A Micromechanical Model for Fracture Process Zone Generation and Microcrack Interactions with a Main Fracture

A Thesis

Submitted in Partial Fulfilment of the Requirements

For the Degree of

Bachelor of Engineering

In

Civil Engineering

By

Dominic Walker

450239612

Supervisor: Dr Itai Einav

School of Engineering

University of Sydney, NSW 2006

Australia

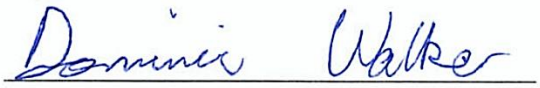
November 2020

DISCLAIMERS

STUDENT DISCLAIMER

The work comprising this thesis is substantially my own, and to the extent that any part of this work is not my own I have indicated that it is not my own by acknowledging the source of that part or those parts of the work. I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure. I understand that failure to comply with the University of Sydney Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under chapter 8 of the University of Sydney By-Law 1999 (as amended).

Author: Dominic Name Surname: Walker

Signed: 

DEPARTMENT DISCLAIMER

This thesis was prepared for the School of Civil Engineering at the University of Sydney, Australia, and describes experiments to induce breakage in a calcareous sand by compression and shearing. The opinions, conclusions and recommendations presented herein are those of the author and do not necessarily reflect those of the University of Sydney or any of the sponsoring parties to this project.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisor, Itai Einav, for his contribution to this research and my development as a researcher. His guidance and encouragement paired with a no-nonsense approach allowed me to develop a new way of thinking in both the generation and evaluation of my ideas as well as how to go about solving problems. Itai's ideas lead to countless intriguing avenues of exploration and added the spice of creativity to this work.

Thanks to Raphael Blumenfeld for the insightful discussion and your comments on this research.

To my family, for their constant and unwavering support throughout my study and throughout this project. Their keen interest in the project, my progress and general presence was the cornerstone for the success in this project, particularly in the face of unique challenges posed by COVID-19.

And, finally to my friends who have been there to support this project with their ideas and computational resources, and for providing me with the motivation to work harder, and achieve more. Special thanks to Leonardo Crespo, Ravi Dissanayake, Anthony Sangster, Olivia Oey, Elyssa Mastroianni, Charlie Wiseman, James Lancaster, Kevin Zou, Matthew Young, Rory McDougall, Lachlan Reid, Langley Ammit, and Ben Korkmann. And, to Edward Farley and Harrison Duncan, thank you for your presence in important moments, for giving me the strength, and creativity to bring to this project to completion.

ABSTRACT

This paper develops a micromechanical model for the development and evolution of microcrack populations ahead of a main fracture. The model combines analytical solutions for crack stress fields, establishes a growth law for individual microcracks and accounts for main fracture – microcrack interactions through the superposition of stresses. The model was first used to analyse the steady state statistics of microcrack geometry and density ahead of the main fracture. Then interactions of the main fracture with individual microcracks was considered and compared to interactions with populations of microcracks. Finally, a sensitivity analysis was done to assess the influence of the microcrack population on the roughness of the main fracture path. It is demonstrated that the distributions of the lengths and orientations for microcracks passing points ahead of the main fracture are well defined and show strong potential for an analytical description. The study of interactions highlighted the importance of considering the microcrack population as a whole, and not simply the effects of microcracks nearest to the main fracture tip. The importance of the population increases as the density of microcracks increases. Additionally, the roughness of the fracture path can be related to the size and density of the microcrack population. More expansive microcrack populations increases fracture path roughness significantly, while increasing microcrack density increases roughness at a decreasing rate.

CONTENTS

List of Figures ..... **vii**

Terminology ..... **xi**

Notation ..... **xi**

1 Introduction **1**

2 Literature Review **2**

2.1 Fracture Mechanics 3

2.2 Microcracks and the Fracture Process Zone 5

2.2.1 Effect of Microcracks 7

2.2.2 Effect of the Process Zone on Structural Response 8

2.2.3 Interpreting the Fracture Process Zone 9

2.3 Methods accommodating the effects of PZ 10

2.3.1 Non-local Continuum Models 10

2.3.2 Discrete Stochastic Models 12

3 Methodology **14**

3.1 Modelling the Main Fracture 14

3.2 Crack Stress Field 15

3.2.1 Analytical Solutions 15

3.2.2 Application of Stress Field Solutions 17

3.3 Voids and Microcracks 17

3.3.1 Voids Distribution 17

3.3.2 Opening Voids and Microcrack Germination 18

3.3.3 Microcrack Growth Law 19

3.3.4 Effective Microcrack Geometry 20

3.4 Crack Interactions 21

3.4.1 Kachanov Method 21

3.4.2 Modified Kachanov Method 22

3.5 Main Fracture Path 24

3.5.1 Path Selection 24

3.5.2 Navigating Microcracks 25

3.5.3 Implementation and Degrees of Freedom 25

3.6 Implementation and Parameters 26
3.6.1 Model Utilisation 26
3.6.2 Parameter Selection 27

4 Results and Discussion 29

4.1 Individuals in a Microcrack Population 29
4.2 Process Zone Properties 31
4.3 Main Fracture Interactions with a Microcrack Population 35
4.3.1 Fracture Trajectory and Path Roughness – Base Case 36
4.3.2 Path Roughness – Sensitivity 39
4.3.3 Crack Tip Stresses along the Fracture Path 40
4.4 Limitations 42
4.5 Valuation of The Model 43
4.6 Future Work – Building the Model and Further Implementation 44

5 Conclusion 46

6 References 48

7 Appendices 54

Appendix A Stress Field Formulae 54
The Static Griffith Crack 54
The Moving Griffith Crack 57
Appendix B Stress Field Plots 60
The Static Griffith Crack 60
The Moving Griffith Crack 62
Weights function 63
Appendix C Derivation for Simplified Kachanov Method for Interactions 64
Appendix D Calculations for Main Fracture Direction and path 65
Fracture Direction 65
Main Fracture Path 66
Appendix E Simulation Overview and Design 67
Overview 67
Design and Procedures 67
Capabilities and Limitations 68
Appendix F Proof of Proportionality of Applied Stress to $\sigma\infty$ 69
Appendix G Discrete Microcracks – Extended Results 70

Appendix H Process Zone Properties – Simulation Grid 70
Appendix I Intercations with Microcrack Populations – Additional Results and Comments 71
Base Case Microcrack Wake and Standard Deviation of Slope 71
Sensitivity Analysis – Inputs 72
Quality of the Gaussian Fit 74
Mohr's Circle Size Distributions 75
Appendix J Python Scripts 76
Input_Parameters.py 76
Main_Script.py 77
Microvoids_Class.py 101
Stresses.py 116

LIST OF FIGURES

Figure 2.1: Concrete core showing cracking from freeze-thaw actions. The 3d crack surface produced using X-ray CT. Source: [13] 2

Figure 2.2: Three force components that act on a crack in an infinite medium of unit thickness. Real cracks experience a combination of these force components or crack modes. Source: [14] 3

Figure 2.3: Acoustic events occurring ahead of a propagating crack. (a) Zone with a high density of recorded acoustic events from direct tensile testing of mortar specimen and crack tip position calculated using LEFM. (b) Near-field acoustic events localising in front of crack from three-point beam bending test. Source: [25, 26] 5

Figure 2.4: Evolution of microcrack formation upon specimen loading. Note in part (d) the inner saturated process zone and outer undamaged zone and the transitional zone separating them. Source: [5] 6

Figure 2.5: Isochromatic patterns showing large stresses about the crack and, in particular, at the crack tip. The patterns are produces by a) constant strain field loading, b) decreasing strain field loading. Source: [27] 6

Figure 2.6: (a)A simple case of interaction between a main fracture and two microcracks near the crack tip. (b) Comparison of results with available research in terms of the stress intensity factor, K. Source: [5] 7

Figure 2.7: Size effect in model mortar beams subject to bending. Source: [3]. 9

Figure 2.8: Kinematic implications with respect to displacement, u, and strain ϵ, for a section taken through the PZ as described by a numerical model. Three model descriptions are presented: (a) cohesive crack model with strong discontinuity, (b) crack band model with weak discontinuity, (a) regularised model with no discontinuity (and continuous differentiability). Source: [35] 11

Figure 3.1: Main fracture geometry and load configuration. 15

Figure 3.2: Contours of major principal stress ($\sigma 1/\sigma\infty$) and line segments parallel to the major principal plane for (a) the static Griffith crack, and (b) the moving Griffith crack with $VMF = 0.5Cs$. The values selected for $\sigma\infty$, VMF and Cs are explained in Section 3.6. Anticipating this section, these values are the same as those used in the *base case* of the sensitivity analysis. 16

Figure 3.3: Void distribution methods for a constant average density of voids: (a) deterministic square grid, (b) stochastic void space. 17

Figure 3.4: Distribution of critical opening stresses, $\sigma crit$, for a population of voids. 18

Figure 3.5: An illustrative sketch of μC growth perpendicular to maximum tensile stress at μC tips. 19
Figure 3.6: Schematic illustration of the calculation for the effective μC geometry. 21
Figure 3.7: Schematic showing the decoupled MF-μC interaction for a single MF-μC interaction. The MF applies stresses on the μC which interprets the MF stresses as traction stresses (acting at infinity) and then transmits stresses back onto the tip of the MF. 23
Figure 3.8: Regions of applicability of the Williams stress field solution used for μC in the interaction model. 24
Figure 3.9: Schematic showing the main fracture in the instant before changing direction to move at an angle θn with respect to the initial MF orientation. 25
Figure 3.10: Schematic representation of the approaches considered for controlling the MF path. 26
Figure 4.1: The evolution of stress state for stresses applied to the tip of main fracture that interacts with singular microcracks. Each curve belongs to a microcrack with a unique displacement, y, above the main fracture major axis. The rectangular stresses $\sigma x, \sigma y, \sigma xy$ shown in (a), (b), and (c) respectively, and are normalised with the stress applied at infinity, σa. Insets for each subplot show an expanded view of the three microcracks are displaced further from the main fracture. 30
Figure 4.2: Results from analysis of a fracture passing individual microcracks with a unique hypothetical displacement, y, above the fracture major axis. The horizontal axis represents the distance travelled since an interaction was first detected. (a) A fictitious direction of main fracture motion determined from stress applied at the tip of the main fracture. (b) Hypothetical fracture path for a fracture that is restricted to straight line motion and interacts with microcracks. 31
Figure 4.3: Sampling points for the pointwise-distributions of microcrack effective length and effective orientation. Horizontally collinear sampling points are considered together to track the evolution of the distributions as points approach the main fracture. 32
Figure 4.4: Kernel density estimation of the effective length (a)-(c) and orientation (d)-(f) distributions for all the microcracks that move along the pathlines indicated in Figure 4.3. Results are grouped by points that are horizontally collinear to demonstrate the evolution of the distributions as microcracks approach the main fracture. 33
Figure 4.5: Results from the analysis of microcrack populations ahead of a fracture. (a) Spatial distribution of the microcrack ratio. (b), (c) The arithmetic mean of effective microcrack orientation and length for all microcracks that pass each point. (d) Zoom in on region in (c) that is less affected by noise. 34

Figure 4.6: Animation frames showing microcracks moving towards the main fracture within the simulation box. The growth of a microcrack is tracked from (a)-(f). The figure is for illustration only, the microcrack growth velocity was doubled to 0.1Cs to exaggerate their size and changes in stress. 36

Figure 4.7: Fracture path of a main fracture restrained to straight line motion through a field of voids. The fracture path has a $5000\times$ vertical exaggeration and the inset has a $200\times$ vertical exaggeration. Note that the length over which the zoom is sampled is equal to the MF length. 37

Figure 4.8: Density distribution of fracture direction and Gaussian fit for the base case simulation parameters. Nonlinear least squares regression was used for fitting the Gaussian. 38

Figure 4.9: Results from the sensitivity analysis for fracture path roughness. 39

Figure 4.10: Density distribution of applied stress measured at the main fracture tip for a fracture interacting with a microcrack population generated using base case simulation parameters. 40

Figure 7.1: Contours for the static Griffith crack stress field solution. 60

Figure 7.2: Angle of Inclination of the $\sigma 2$ direction to the positive x-axis for the Williams stress field solution. This corresponds to the direction in which microcracks would grow according to the maximum tensile stress criterion. 61

Figure 7.3: Contours for the dynamic Griffith crack stress field solution. 62

Figure 7.4: Angle of Inclination of $\sigma 2$ direction to the positive x-axis for the Yoffe stress field solution. This corresponds to the direction in which microcracks would grow according to the maximum tensile stress criterion 63

Figure 7.5: Merging near- and far- stress field solutions along the crack major axis for (a) the static crack (Williams), and (b) the dynamic crack (Yoffe). 63

Figure 7.6: Contours for the dynamic Griffith crack stress field solution at the start of main fracture propagation. Voids are spaced such that the main fracture interacts with one microcrack at a time. Microcracks are shown in the final state which occurs when the main fracture passes. 70

Figure 7.7: Grid on which process zone properties were measured. Data from 2500 voids was collected at each point. 70

Figure 7.8: Crack path through a field of voids and the associated microcrack wake for a base case simulation. Only voids are shown. Red points indicate open voids, black points are closed voids. 71

Figure 7.9: Change in the standard deviation of the θ-distribution along the path of the fracture. The standard deviation changes as the distribution of main fracture trajectories accumulates more values. The standard deviation reaches a constant value, indicating attainment of a steady state. It is at this point that full θ-distribution is assumed. 71

Figure 7.10: Weibull distribution generated with base case parameters for sampling vales of $\sigma crit$.....72
Figure 7.11: Weibull distribution for the (a) maximum and (b) minimum values of shape parameters, m, used in the sensitivity analysis of MF path roughness.....72
Figure 7.12: Weibull distribution for the (a) maximum and (b) minimum values of scale parameter, σw, used in the sensitivity analysis of MF path roughness. 73
Figure 7.14: Void density for the (a) base case, (b) maximum and (c) minimum values of void density, $\rho voids$, used in the sensitivity analysis of MF path roughness. 73
Figure 7.15: Distributions of main fracture orientation, θ, for a sample of sensitivity analyses to compare the quality of the fitted Gaussian PDF. A Cauchy PDF is also fitted to the θ-distribution to compare to the fit of the Gaussian PDF.....74
Figure 7.16: Distribution of the size of Mohr's circle measured along the main fracture path. Increasing the void density shifts the distribution in the positive direction, thereby indicating that larger Mohr's circles are typical in denser microcrack populations. 75

TERMINOLOGY

Main crack axis – the axis parallel to the propagating crack that passes through the crack centre.

Minor crack axis – the axis perpendicular to the propagating crack that passes through the crack centre.

Microcrack ratio – the ratio of microcracks to voids at a point in the coordinate reference system of the main fracture.

NOTATION

a	Main fracture half length
b	Main fracture half height
C_s	Shear wave speed
C_l	Elastic wave speed
d_0	Bažant's characteristic length
E	Elastic modulus
G	Energy release rate
G_c	Critical energy release rate
K_I	Stress intensity factor for mode I crack loading
K_{II}	Stress intensity factor for mode I crack loading
K_{Ic}	Critical stress intensity factor for mode I loading
K_{IIc}	Critical stress intensity factor for mode II loading
$L_{ef,\mu C}$	Effective length of a microcrack
LEFM	Linear Elastic Fracture Mechanics
m	Weibull shape parameter
MF	Main Fracture
MTS	Maximum Tensile Stress (criteria)
PZ	(Fracture) Process Zone
r_R	Distance from the leading tip of the main fracture. That is the tip that opens the material.
V_{MF}	Main fracture velocity
$V_{\mu C}$	Microcrack velocity
γ	Specific surface energy
θ	Direction of main fracture motion.
$\theta_{ef,\mu C}$	Effective microcrack orientation
Λ	Transmission factor for Kachanov interactions

$\Lambda_{i,MF}^{\tau n}$	Transmission factor for microcrack i mode II tractions that induce mode I tractions on the main fracture
$\Lambda_{i,MF}^{nn}$	Transmission factor for microcrack i mode I tractions that induce mode I tractions on the main fracture
$\Lambda_{i,MF}^{\tau\tau}$	Transmission factor for microcrack i mode II tractions that induce mode II tractions on the main fracture
$\Lambda_{i,MF}^{\tau n}$	Transmission factor for microcrack i mode II tractions that induce mode I tractions on the main fracture
λ	Scale parameter for stress field weights function
μC	Micro-Crack
ν	Poissons ratio
ρ_s	Material solid density
ρ_{voids}	Void density
σ_a	The sum of the interaction stresses and the stress applied at infinity
σ_f	Crack tip stress required to propatate a crack
σ_w	Weibull scale parameter
σ_{crit}	Critical stress required to initiate microcrack growth at a void
σ_∞	Stress normal applied at infinity. Induces mode I loading for horizontal main fracture, or mode I and II for arbitrary main fracture orientation.

1 INTRODUCTION

Failure in structural materials is often characterised by the accumulation of damage and distributed cracking. A single crack is represented by an interface that forms in a material due to imposed loads. The growth and propagation of a crack is governed by different phenomena at different length-scales and time-scales [1]. There are at least three length scales over which these distinct behaviours occur. (1) At the atomic level (5-10 Å), the crack tip breaks bonds which releases energy and transfers the larger stress ahead of the crack, this is a discrete problem. (2) In the mesoscale (10 nm – 10 μm) exists material defects such as voids from which microcracks (μC) nucleate. (3) At a 'large enough' length scale (typically $>$ μm) material inhomogeneities and irregularities become negligible and continuum approaches are adopted. For brittle and ductile materials, the zone in which microcracks accumulate, called the fracture process zone (PZ), is in the order of microns and continuum mechanics can be readily applied. However, in quasibrittle media the size of the PZ can be relatively large; for example, in normal-strength concrete the PZ is up to 0.5 m long and in arctic sea-ice up to 5 km long [2].

The effect of the PZ on the overall structural performance is increased variance in measurements of strength and large post-failure softening [2]. When predicting material strength, if microcracking damage is unaccounted for, the strength of material is frequently overestimated, and the failure mode becomes more difficult to predict. At present, the treatment of the process zone is indirect at best, and pseudo-phenomena such as strain softening is introduced to model the effect of the process zone. This is a consequence of limited understanding of the microcrack populations are generated ahead of a fracture. The inability to predict the formation of these precludes the development of theory that directly accounts for their effects. Attempts have been made to measure the size of these process zone in correlational studies and to navigate issues in modelling, yet these approaches lack consistency. For example, some studies declare that the process zone includes 80% of the microcracks either side of the propagation axis [3], others determine the width such that microcracks within three standard deviations of the MF propagation axis are included [4]. Even if these measurements were consistent in literature, their arbitrariness remains. Thus, a consistent, justifiable method of describing the process zone in a geometric sense is needed.

An understanding of the way μC populations are generated and evolve is required to investigate the navigation of a fracture through a process zone. Previous research has investigated the way cracks experience arrays of microcracks [5, 6, 7, 8, 9, 10, 11, 12], but these arrays are not determined by the stresses that produce the microcracks. As a consequence, important

characteristics of the interactions may not be captured, and it is also possible that non-existent phenomena are reported.

The research presented herein seeks to establish a rudimentary micromechanical model for process zone generation and the interactions of the process zone with the main fracture. The model was then used to investigate the properties of the process zone that can be used in the future development of an approach for process zone characterisation. Further, the model was used to analyse the way that microcracks affect the navigation of a main fracture by assessing the roughness of the fracture path, and also the way microcracks induce changes to the stress state at the fracture tip.

2 LITERATURE REVIEW

The crack surface in quasibrittle material is illustrated in Figure 2.1 by the red surface dividing the medium. While the overall orientation of the surface can be aligned vertically with a concave shape in the horizontal plane, the smaller scale shape appears perturbed and random, particularly along the crack front. Given the cracked pre-failure state of this material, fracture mechanics seeks to answer questions regarding residual strength, the load required to propagate the crack and the direction of propagation, and how stresses are redistributed in the cracked solid.

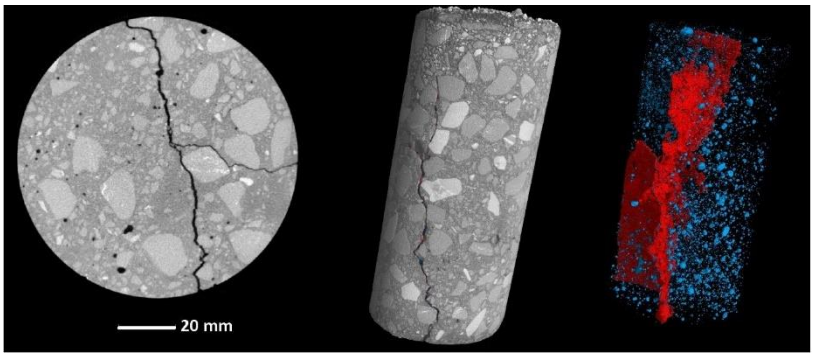


Figure 2.1: Concrete core showing cracking from freeze-thaw actions. The 3d crack surface produced using X-ray CT. Source: [13]

The approach to understanding the three-dimensional problem has historically been to consider the two-dimensional analogue of a crack in a plane. In this more manageable scenario, the forces that cause crack propagation can be divided into three components; in-plane tension, in-plane-shear and out-of-plane shear in the configuration indicated in Figure 2.2 [14].

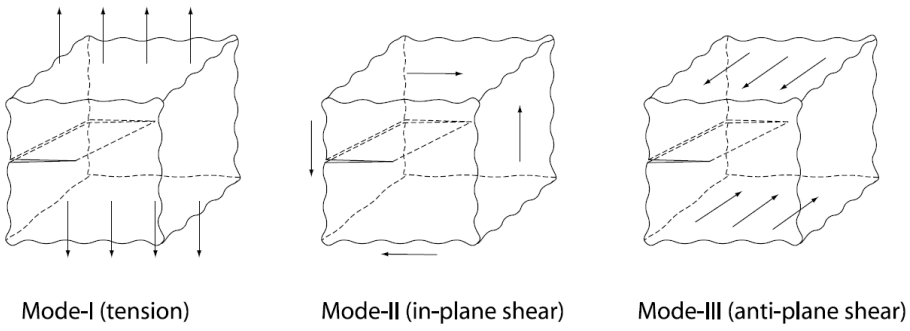


Figure 2.2: Three force components that act on a crack in an infinite medium of unit thickness. Real cracks experience a combination of these force components or crack modes. Source: [14]

2.1 FRACTURE MECHANICS

The first significant step to understanding fracture resistance occurred in 1921 for Mode-I cracking of brittle material by Griffith [15, 16]. Griffith hypothesised that the elastic energy supplied to a crack must at least be equal to the energy required to increase the material surface area [16]. With this he was able to relate the critical stress, σ_{cr}, for crack propagation to the crack geometry and material properties in the equation:

$$\sigma_{cr} a^{1/2} = \left(\frac{2E\gamma}{\pi}\right)^{1/2}, \tag{1}$$

where; $2a$ is the crack length, E is Young's modulus and γ is the specific surface energy. This relied on assuming a perfectly elastic homogenous material up to the point of failure, therefore being useful for brittle material. However, for material that exhibit some sort of permanent damage response such as yielding, opening of microvoids or growth of microcracks, Griffith's theory underestimates strength considerably [17]. Irwin, in the 1950s explained that the additional energy required to produce plastic deformation should be included in the energy balance. That is, energy supplied to the crack, denoted the energy release rate, G, must be greater than the sum of the plastic energy and surface energy required to propagate the crack [18]. This requires an accurate account of the zones where permanent damage processes occur.

Irwin's hypothesis in plane stress can be written mathematically as follows:

$$\sigma_f = \left(\frac{EG_c}{\pi a}\right)^{1/2} = \frac{K_c}{\sqrt{\pi a}}, \tag{2}$$

where; σ_f is the crack tip stress required to propagate the crack, a is the crack length, E is the elastic modulus, and G_c is the critical energy release rate. G_c is equal to the sum of energy required for plastic flow and energy required to increase material surface area. Closely related to G is the stress

intensity factor, K. The critical stress intensity factor K_c is obtained by rearranging Eq. (2) in terms of K_c in terms of the failure stress, σ_f, and the crack length. That is,

$$K_{Ic} = \sigma_f \sqrt{\pi a}. \tag{3}$$

One of Irwin's major contributions was establishing a criterion for crack propagation in terms of the stress intensity factor. Irwin hypothesised that crack propagation occurs when the stress intensity factor reaches a critical value and that the critical values is a material constant [19]. It should be noted that recent research has shown that K_c varies along the crack path [1], however the ease with which K_c can be measured has caused the longevity of stress intensity approaches. Moreover, Irwin recognised if the plastic zone is small enough, the near-field stresses about the crack tip can be fully described in terms of the stress intensity factor. This was shown by Williams (1957) in his modification to the first crack stress field solution proposed by Westergaard (1939). The general form of Williams' solution is presented here for σ_{xx}, the stress component for the axis parallel to the crack path:

$$\sigma_{xx} = \frac{K_I}{\sqrt{2\pi r}} f(\theta) + \frac{K_{II}}{\sqrt{2\pi r}} g(\theta), \tag{4}$$

where; stresses depend only on position in terms of polar coordinates (r, θ) and the stress intensity factor, K. The form of the solution for other Cauchy stress components is similar to Equation (4), but functions f and g are different for each stress component.

Since Irwin, developments have relied heavily on measuring and modifying the stress intensity factor to explain observations related to crack mode, time effects, propagating speed and propagation history [1]. The validity of this approach of explaining new results by more broadly defining K is questionable, considering it was developed assuming quasi-static crack propagation on an infinite plane on a linear elastic material under mode-I cracking [1, 18].

To increase the practicability of Irwin's model, it was assumed that the energy dissipation zone, or the PZ, ahead of the crack is of constant size [18], yet this ignores the rate dependence of the PZ width [1, 5]. This has implications for the validity of linear elastic fracture mechanics (LEFM) which may vary during the evolution of a void to a main fracture (MF); being invalid when the crack is small and PZ is relatively large and then once the crack is big enough LEFM becomes valid. There are further inconsistencies with respect to measurements of K and G which should be self-consistent [20]. This has been explained by the variations in the PZ size which invalidates the use of K altogether [21]. Despite this, the simplification using K has been the prevailing strategy of LEFM due to its convenience in computation and experimentation.

2.2 MICROCRACKS AND THE FRACTURE PROCESS ZONE

In quasibrittle materials such as concrete, microcracks can develop during forming. In this discussion, reference to microcracks specifically denotes those that develop during crack propagation.

The coalescing of microcracks to form a densely populated ‘process zone’ has been well documented experimentally. Cedolin *et al.* [22] and Castro-Montero *et al.* [23] used optical interferometry to show the existence of a process zone as well as the wake of microcracks behind the crack tip. The scanning electron microscope (SEM) and acoustic emission techniques were used in [24, 25, 26, 13, 19], all showing discrete microcracking along the main fracture path. The acoustic emission results of [25] and [26] are both available in Figure 2.3.

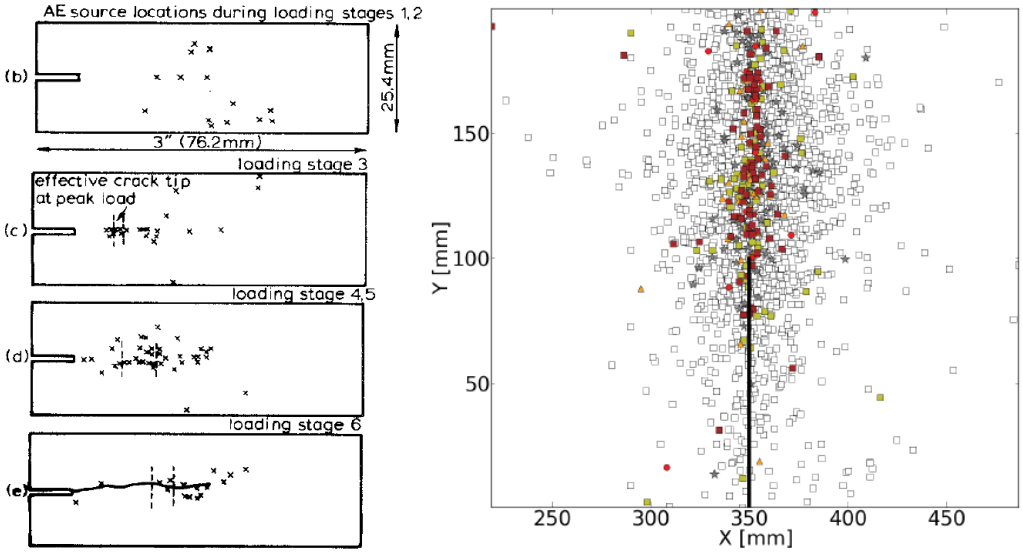


Figure 2.3: Acoustic events occurring ahead of a propagating crack. (a) Zone with a high density of recorded acoustic events from direct tensile testing of mortar specimen and crack tip position calculated using LEFM. (b) Near-field acoustic events localising in front of crack from three-point beam bending test. Source: [25, 26]

The distribution of microcracks can be categorised into three classes. First, a ‘saturated process zone’ which is located about the plane of the crack in Figure 2.3(b). Second, a transitional zone separates the undamaged material from the highly damaged material. Third, outside the transitional zone, only pre-existing microcrack are present [5]. These zones are not clear from Figure 2.3, therefore an illustrative diagram is provided in Figure 2.4.

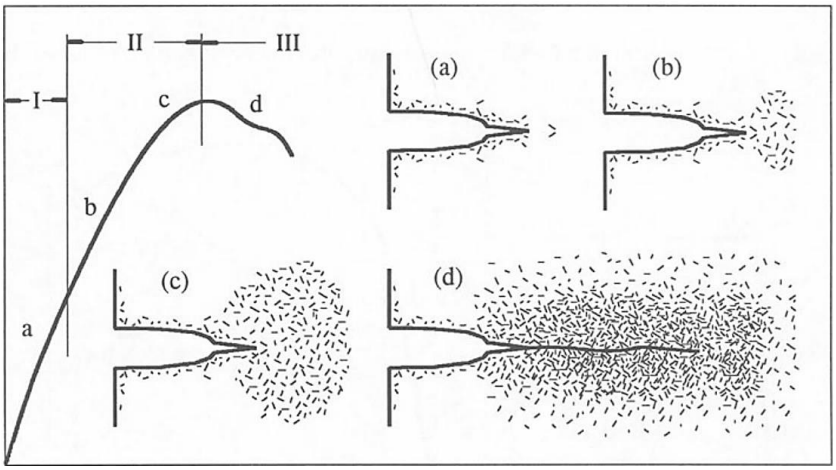


Figure 2.4: Evolution of microcrack formation upon specimen loading. Note in part (d) the inner saturated process zone and outer undamaged zone and the transitional zone separating them. Source: [5]

While the existence of the PZ is well documented, the mechanisms that cause them are nontrivial. It is expected that damage will localise near to the fracture tip due to the large strain strains about the fracture tip. This is reflected in the elastic stress field solution (Equation IV) by observing the stress variation as radial distance from the crack tip, r, tends to zero. High stress levels at the tip are visualised in Figure 2.5 by the convergence of isochrones to the crack tip.

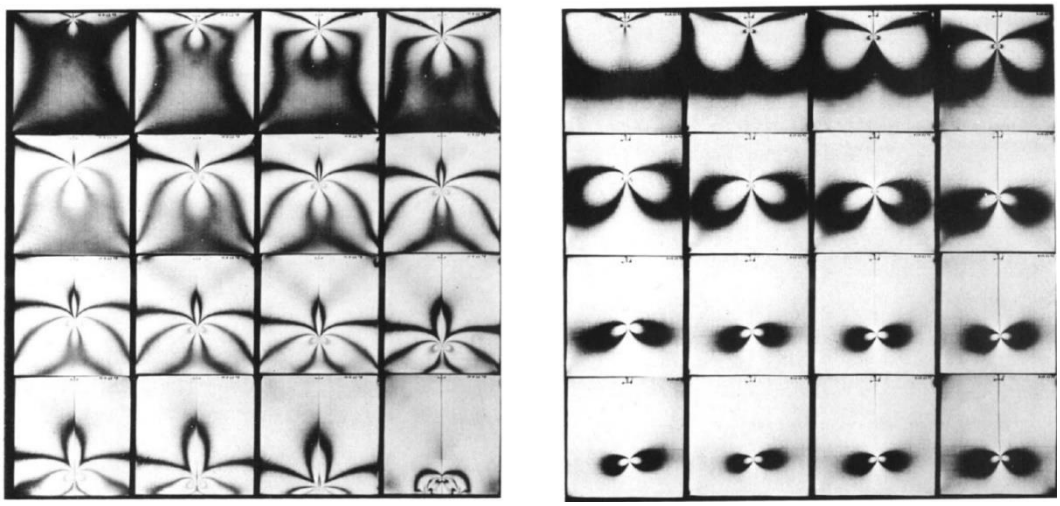


Figure 2.5: Isochromatic patterns showing large stresses about the crack and, in particular, at the crack tip. The patterns are produces by a) constant strain field loading, b) decreasing strain field loading. Source: [27]

Moreover, it is understood that average tensile stresses and the material microstructure control the orientation, shape, length, and distribution of microcracks [28, 5, 3]. It is generally agreed upon that microcracks propagate in a direction that is perpendicular to the average tensile stress direction and either link up with other microcracks or are arrested at voids or heterogeneities in the material [28, 29]. Note that, the complex shape of the stress field precludes a clear assessment of the orientation of microcracks as the average tensile stress direction varies. This means that microcracks that open at different distances in front of the main fracture have greatly varying

orientations and overall shape as they have experienced a different stress history by the time they are close to the main fracture.

2.2.1 EFFECT OF MICROCRACKS

Generating microcracks from a fracture stress field is a self-consistent problem. The stresses that control the formation and characteristics of microcracks are themselves affected by the existence of microcracks. When a crack propagates, the stress field updates through the propagation of shear waves from the crack tip and then through the medium which bring internal and applied stresses back into equilibrium. Quasi-static approaches assume that this return to equilibrium is sufficiently fast. The validity of this assumption depends on the relative speed of the fracture tip velocity and the stress singularity (which does not necessarily coincide with the fracture tip). Blumenfeld [1] explained that the disordered cluster of microcracks forming the process zone scatter shear waves on a wide range of wavelengths. As a consequence, the stress relaxation rate decreases. The effect is most prominent near the crack tip as this is where the densest population of microcracks exist. It was shown by Freund [1] that the stress relaxation rate inside the PZ at crack arrest is an order of magnitude slower than what is expected for a homogenous material. As such, the popular assumption of quasi-static conditions in LEFM is least appropriate at the crack tip. While, enforcing sufficiently slow crack propagation in models will increase the validity of these models, the problem becomes the applicability of LEFM as crack propagation is often a dynamic problem.

Slower shear wave speed in the process zone is responsible for a delayed 'communication' between the crack tip and far field stresses. The crack tip is shielded from global behaviour in what is effectively a stress field cut-off [1]. The shielding effect was studied by [5] using the Boundary Element Method for a main fracture propagating under mode-I loading and approaching parallel microcracks distributed about the main fracture (major) axis. The setup showing a single pair of microcracks is illustrated in Figure 2.6.

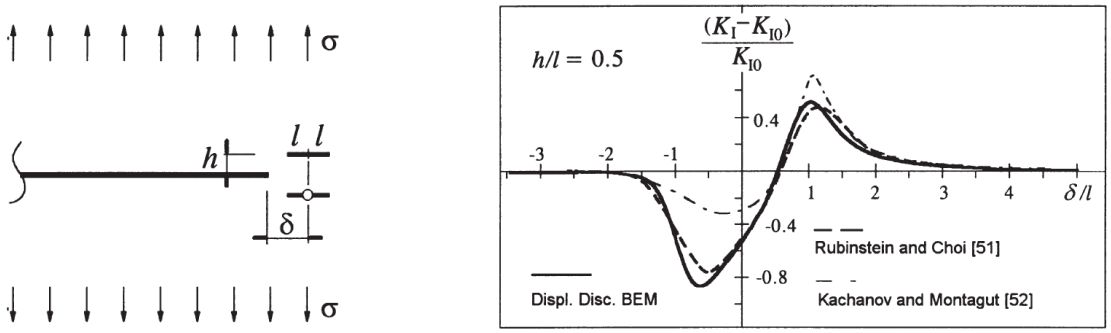


Figure 2.6: (a)A simple case of interaction between a main fracture and two microcracks near the crack tip. (b) Comparison of results with available research in terms of the stress intensity factor, K. Source: [5]

The results highlighted three important phenomena. First, the net effect of microcracks on the main fracture depends on the distance of the microcracks from the main fracture tip. On approach, the stress intensity factor of the main fracture decreases to a minimum, indicating a stress reduction at the main fracture tip. Amplification starts once the main fracture has passed the nearest crack tips of the microcracks. Moreover, microcracks induce shielding over a greater distance and with larger magnitude than amplification as shown in Figure 2.6b where the relative variation of the stress intensity factor K_i is negative for a larger range of δ/l. Since quasi-static models do not account for shielding, as it is an out-of-equilibrium phenomenon, predictions for crack tip stress are incorrect and frequently overestimated [1, 5]. Second, the level of shielding given by the pair of microcracks nearest the crack tip is independent of the microcracks further away, highlighting the highly localised nature of the process zone. This was also shown by Laures and Kachanov [30] using a statistical analysis and numerical modelling finding that the microcracks nearest the crack tip had the greatest influence on the stress intensity factor. Third, if subsets of microcracks in the process zone are considered the results from one subset could predict stress amplification, while another subset would predict shielding. Thus, the process is not simply the sum of its discrete parts, there is a synergy within the microcrack population that needs to be accounted for. Recalling that the cut off between the process zone and the transition zone is ambiguous, the decision of where to draw the line becomes important as to not neglect microcracks that have significant participation in shielding and amplification. Perhaps two widths should be considered; the PZ width and the transitional zone width. Alternatively, if a continuous predictive model for the PZ geometry and properties is available, the discussion of widths may become unnecessary.

2.2.2 EFFECT OF THE PROCESS ZONE ON STRUCTURAL RESPONSE

In brittle materials the PZ is small so LEFM can be applied and the scaling of stresses and strains from small test specimens to larger scale structural elements follows a simple power law [31, 15]. In quasibrittle materials the PZ size is significant compared to a structural dimension, D. The nonlinearity induced by microcracking is significant and the redistribution of stress occurs on a macroscopic scale [15]. As such, there is disagreement between the predicted material strength from results of a small test specimen and the actual structural strength. A typical example of the size effect in a model mortar test specimen is available in Figure 2.7.

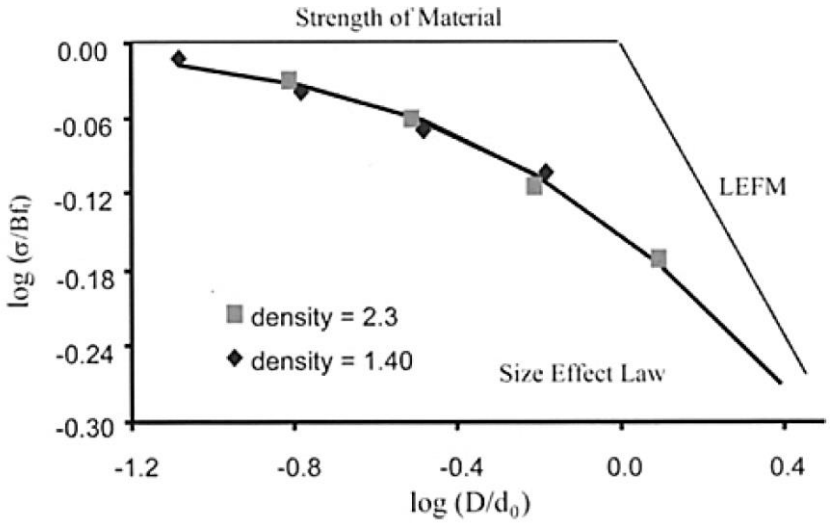


Figure 2.7: Size effect in model mortar beams subject to bending. Source: [3].

2.2.3 INTERPRETING THE FRACTURE PROCESS ZONE

From a modelling perspective, there are three lengths that are involved. The lengths are used to navigate pathological issues of continuum numerical methods and characterise the failure mechanism. First is the *characteristic length*, which defines the transitional point between material strength and fracture energy [3, 2]. This is exactly d_0 in Figure 2.7. Second, the *internal length* which is a tool used for numerical modelling. The internal length is used for spatial averaging in problems where theory predicts a dissipative process theoretically occurring in zero volume [28]. In this case, the singularity is at the crack tip which enforces the internal length to be the minimum possible width of the process zone [32]. Third, the *process zone width*. The question here is about how the process zone should be measured. The challenge can be seen in Figure 2.3 in the ambiguity of where the saturated PZ ends and the transitional zone starts. That is why studies that claim to be able to measure the width of the process zone should be treated with caution. However, for determining an appropriate internal length for modelling purposes a 'calculated' process zone width may be useful as the internal length need only capture the order of magnitude of the process zone and underestimating the internal length will not cause significant numerical errors, though may be more computationally expensive.

Despite the ambiguity, research that has arbitrarily defined the process zone cut-off width has be able to relate the characteristic length, d_0, and the internal length to the process zone width [33]. Haidar et al [3] conducted a correlational study using AE measurements and inverse finite element analysis to estimate the PZ width and compare it to values of d_0 and the characteristic length in literature. The PZ distribution from AE recording and a comparison of the lengths for different test specimens is available in Figure 2.7. The PZ width cut-off included 80% of the

acoustic events. The results indicated when specimens of different density were tested, the proportional change in ‘lengths’ was relatively consistent. This suggests it is possible to predict the transitional point at which material failure goes from strength controlled to fracture mechanics controlled from the geometry of the PZ as determined by material properties. Despite this, better understanding of the meaning of the ratio between the PZ width and characteristic length is required and yet to be clearly explained.

2.3 METHODS ACCOMMODATING THE EFFECTS OF PZ

The process zone is characterised by a region of highly localised strains. On the macroscale this corresponds to strain softening which is a decrease in stiffness of the overall structural response. This is equivalent to locally increasing strain while the rest of the medium is unloaded. In mathematical and numerical models that rely on continuum mechanics, the strain softening region becomes infinitely small which causes the amount of energy dissipated to reduce, in some cases, to zero because no energy can be dissipated in zero volume [34, 35]. Therefore, strain softening in boundary value problems such as crack propagation leads to a loss of ‘well-posedness’. This is because, in the static case, equations of equilibrium loose ellipticity, and in dynamics, equations of motion lose hyperbolicity [28, 34] which corresponds to, in numerical analysis terms, non-convergence as well as non-unique (mesh dependent) solutions.

2.3.1 NON-LOCAL CONTINUUM MODELS

Enrichments to continuum models can be made to restore well-posedness in conditions with discontinuous or highly localised stresses and strains. The approach is to consider, not the crack tip, but the crack-tip ‘dressed’ with the PZ [1]. That is, the singularity near the crack tip is replaced by an effective continuum. There have been three primary approaches to achieve this. The kinematic implications of following approaches are illustrated in Figure 2.8 and explained in the following discussion.

First, the *cohesive crack model* in Figure 2.8a directly incorporated strong localised discontinuous displacements about the crack tip by defining a traction-separation law between the crack faces. While the strain is still localised at the crack faces, the taction separation law ensures that stresses can be transferred to near crack material. While this method has proven useful in its simplicity, it is relegated to crack propagation along a predetermined path under quasi-static

conditions. In losing the detail of the process zone, the cohesive zone model is not useful in predicting dynamic crack behaviour.

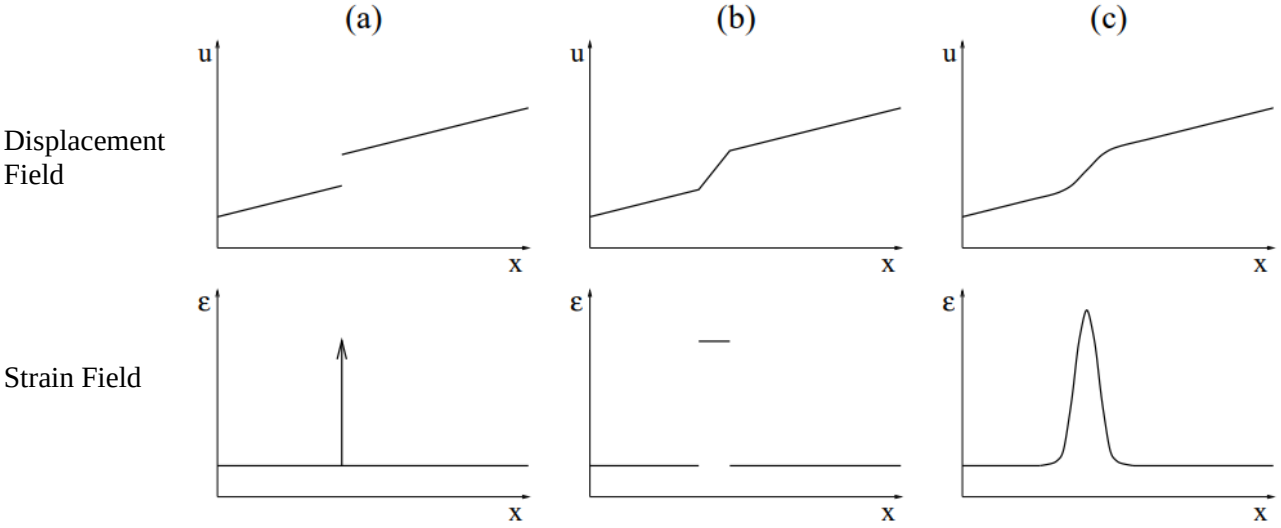


Figure 2.8: Kinematic implications with respect to displacement, u, and strain ϵ, for a section taken through the PZ as described by a numerical model. Three model descriptions are presented: (a) cohesive crack model with strong discontinuity, (b) crack band model with weak discontinuity, (a) regularised model with no discontinuity (and continuous differentiability). Source: [35]

Second, the *crack band model*, as in Figure 2.8b represents the process zone by only allowing localised strains to take place in a band of constant width. The crack band is lined with an a very weak discontinuity on each side over which piecewise continuous strains take place [36, 35]. Kinematic properties of the process zone are captured homogeneously over the width of the crack band. This is equivalent to removing the transitional zone in Figure 2.4 and assuming constant properties over a crack band. This requires establishing an effective process zone width.

Third, *regularised continuum models* as in Figure 2.8c modify strain singularities using a regulator. Regulators are used as a stepping-stone over gaps in knowledge by creating the behaviour that is expected to be taking place. Regulators are not derived from fundamental principles, however, can be well founded in understanding of material behaviour. In crack propagation problems the regulator acts a localisation limiter by using either differential or integral formulations to define a method of smoothing the strain discontinuity. The result is a continuous strain field and a continuously differentiable displacement field about the PZ, as shown in Figure 2.8c [35]. Continuity is generally achieved by using weighted spatial averaging techniques to calculate a non-local counterpart to an internal variable. That is, at each point inside the PZ there is defined a non-local strain, $\overline{\epsilon_{eq}}$, that is a function of either all the strains in the body or strains in the neighbourhood of the point in question. Essentially, this assumes that the strain at a point is a function of the displacement at that point as well as the displacements of all other points in the neighbourhood or entire body [32]. A generalised integral formulation is given as:

$$\overline{\epsilon_{eq}} = \int_V \alpha(x,\xi)\epsilon_{eq}(\xi)\, d\xi \qquad \text{(IV)}$$

where, $\alpha(x,\xi)$ is a non-local weight function that controls spatial averaging and depends on the internal length to define the smoothing domain; ξ is the distance between some neighbouring point and the point, x, under consideration; and V defines the extents of the neighbourhood over which 'smoothing' operations will be applied. Differential formulations are also available.

An example of how this is used is in non-local damage formulations. In damage formulations an internal damage variable, $0 \leq \theta \leq 1$, is used to relate a material loading history to the stress strain response. This is done using a constitutive law whereby accumulation of damage corresponds to a decrease in stiffness. Since the level of damage is a function of the largest strain experienced at a point, damage models can be susceptible to the same problems that continuum fracture mechanics models are with strains localising in an extremely narrow band. In the non-local brand of damage models, the damage variable, $\theta = \theta\left(\overline{\epsilon_{eq}}\right)$, is written as a function of the equivalent non-local strain, $\overline{\epsilon_{eq}}$. The stiffness of the material becomes a function of the non-local strains, however the stresses are determined using the local strains [34, 35].

The ability of non-local models to capture real behaviour is dependent on the internal length. Mathematically, the internal length controls the size of the area over which the weight function conducts averaging. Therefore, it is paramount that an appropriate internal length is selected. Moreover, these non-local continuum models assume a constant internal length. The present consensus is that this is not the case [28]. As such, if methods involving an internal length are to be used, more robust models are required that can capture the variation in process zone size. This being the case, available models can still be improved with better predictions of the extents of the PZ combined with consistent treatment of the 'transitional zone' when assigning a process zone width. These constitute important contributions in predicting fracture behaviour.

In numerical models, non-local formulations have given rise to the adaptive meshes which update and refine at zones with large strains. The advantage of this refinement is limited by the current knowledge of process-zone development and the continuum accommodation of microcracks.

2.3.2 DISCRETE STOCHASTIC MODELS

In real materials, the coalescence of disordered microcracks in a developing process zone is a non-deterministic, non-local, non-linear and non-equilibrium process [37]. While non-local continuum models create a fictitious continuum description of strains about the PZ much information about the characteristics of the PZ's constituent microcracks is either homogenised and

lost or undetected. Discrete stochastic approaches treat each microcrack individually and do not require assumptions with respect to crack shapes, sizes, distributions of weak spots [29]. This allows the anisotropic process zone structural response to be retained. Microcracks open based on a probability distribution which is a function of some initiation stress level and interaction laws between microcracks and the main fracture [29]. The benefit of discrete stochastic models is the direct treatment of the PZ. This is countered by computationally expensive numerical simulations required to treat each microcrack individually. At present computational capabilities are insufficient for stochastic approaches to be an industry viable crack analysis method.

Improved understanding of the process zone distribution and the mechanism of microcrack interactions would decrease the computational cost of discrete models by decreasing the number simulations required. Moreover, a predictive model for the PZ geometry and microcrack orientation and length may allow for an effective representation in continuum mechanics which would allow more accurate solutions without the need for computationally expensive discrete models.

3 METHODOLOGY

This research was intended to develop a method for predicting μC populations and reveal fundamental aspects of the MF-PZ interaction. The principal governing the approach is one of simple assumptions. Despite the critique of continuum approaches and linear elastic constitutive models in Section 2, the simplicity in implementation and basic assumptions serve the purpose of this research.

Overall, the main goal was to observe the generation of μCs due to the presence of a MF and develop a method for predicting μC populations and the fracture path. This was done by constructing a blended conceptual-computational model which was implemented in a computer simulation using a Python (3.7.6) Program. The model borrows elements from continuum models and discrete stochastic models. Namely, stresses are calculated from analytical solutions based on LEFM (Section 3.2) and μCs are given discrete treatment.

The sections that follow describe the components of the model (Section 3.1-3.5) and the implementation (Section 3.6). Elements of the Python program are also described, however specific details of the Python scripts and the procedures the main simulation engine are described in Appendix E and Appendix J.

3.1 MODELLING THE MAIN FRACTURE

A two-dimensional finite Griffith Crack moves with velocity, V_{MF}, in an infinite plate made of linear elastic, isotropic material that is subjected to mode-I loading, σ_∞, that is applied uniformly at infinity. The MF is modelled as an ellipse of constant length, $2a$, and height, $2b$, such that $a \gg b$. That is, the leading tip of the MF and the trailing tip move at constant velocity. The geometry of the situation and the configuration of the applied loads is illustrated in Figure 3.1.

The stress field ahead of the MF are applied to μC potential locations, referred to as voids, which become μCs if stresses are large enough (Section 3.3). MF-μC interactions are considered for μCs within the simulation box. For interactions, stresses applied to μCs which then transmit their own stresses back onto the MF (Section 3.4). The interaction stresses are used to update the fracture direction (Section 3.5).

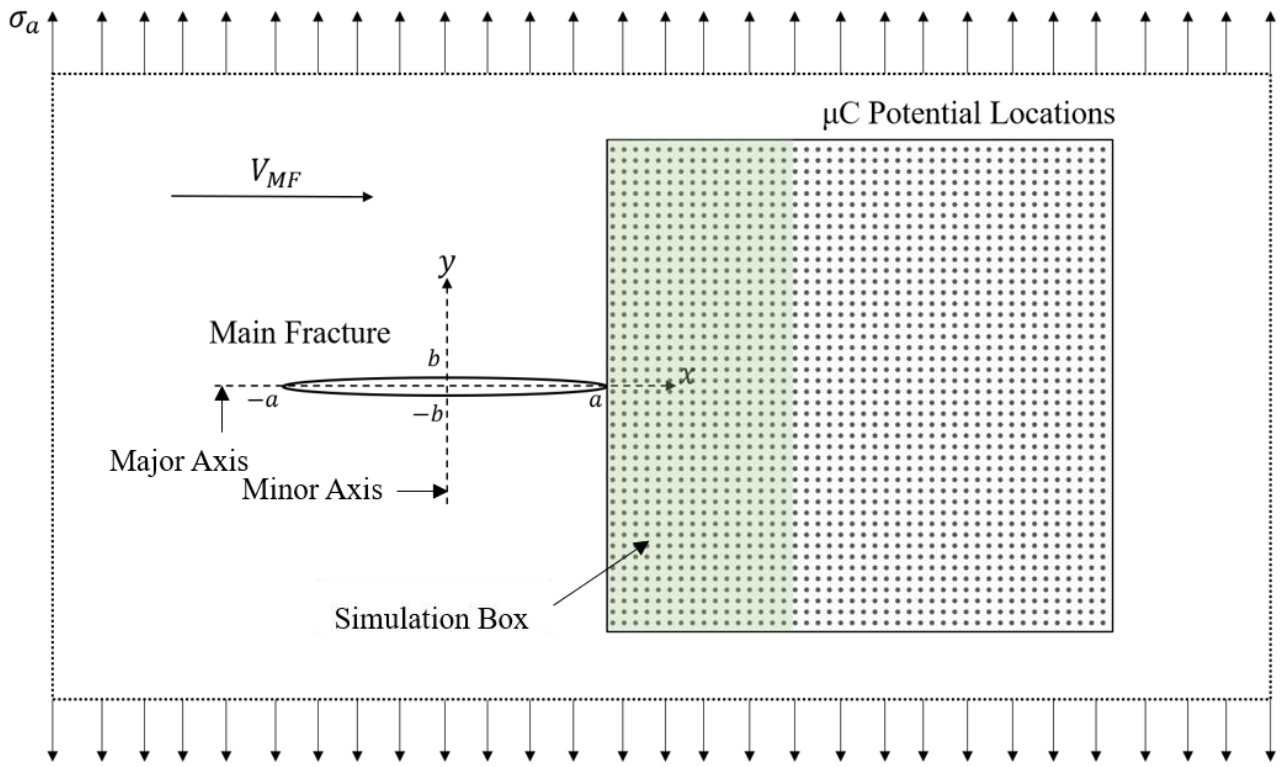


Figure 3.1: Main fracture geometry and load configuration.

3.2 CRACK STRESS FIELD

Stress fields form the basis of the model proposed here. The objects in the system are only recognised by other objects through the perturbations in stress field that they induce. Deformations that result from the propagation of the MF or the generation of the μC population are ignored.

3.2.1 ANALYTICAL SOLUTIONS

Analytical solutions to the stress field ahead of the Griffith Crack are available for a static crack and a moving crack for each crack opening mode (Modes I, II and III) [38]. The dynamic stress field derived by Yoffe (1951) [39] converges to the Williams (1957) [40] static crack stress field as crack velocity approaches zero. The solutions are available for both plane strain and plane stress. Plane stress is considered here.

The analytical stress field solutions produced by Yoffe and Williams are appropriate for near-crack tip stresses and do not satisfy boundary conditions at infinity. The applicability is limited to a region where the distance from the leading tip, r_R, is such that $r_R \ll a$. Far field solutions have been solved to account for the boundary conditions at infinity [38]. The near- and far-field equations are long and heavy, therefore they are produced in Appendix A.

A full description of the stress field at any point ahead of the crack requires that the near-field solution and far-field solution are combined. A continuous exponential weights function is used for merging the two stress fields in the transitional region between the near- and far-field. The weights function is reproduced in here,

$$\sigma = \mathrm{e}^{-\lambda r_R/a} \cdot \left(\sigma_{near} - \sigma_{far}\right) + \sigma_{far} \tag{5}$$

where; σ_{near} and σ_{far} represents stress components the near field solution and far field solution, respectively, and $\lambda = 3$ is a scaling parameter.

The selection of λ was arbitrary, although the idea was to ensure that the near-tip stress field solution contributed a significant proportion of the stresses where $r_R < 1.2a$. An example demonstrating the application of the weights function to the near-field and far-field stresses along the horizontal line, $y = 0$, is available in Appendix B. The arbitrariness of the exponent in Eq. (5) must be appreciated. An alternative to be implemented in a future version of the model is to tune the stress field using the output of an equivalent ABAQUS model.

Contours for the major principal stress calculated using the full stress field solution of the static and dynamic Griffith crack are sketched in Figure 3.2 for an identical applied stress, σ_a. The orientation of the major principal plane is also shown. Additional plots for the Cauchy stress tensor and principal stresses are available in Appendix B.

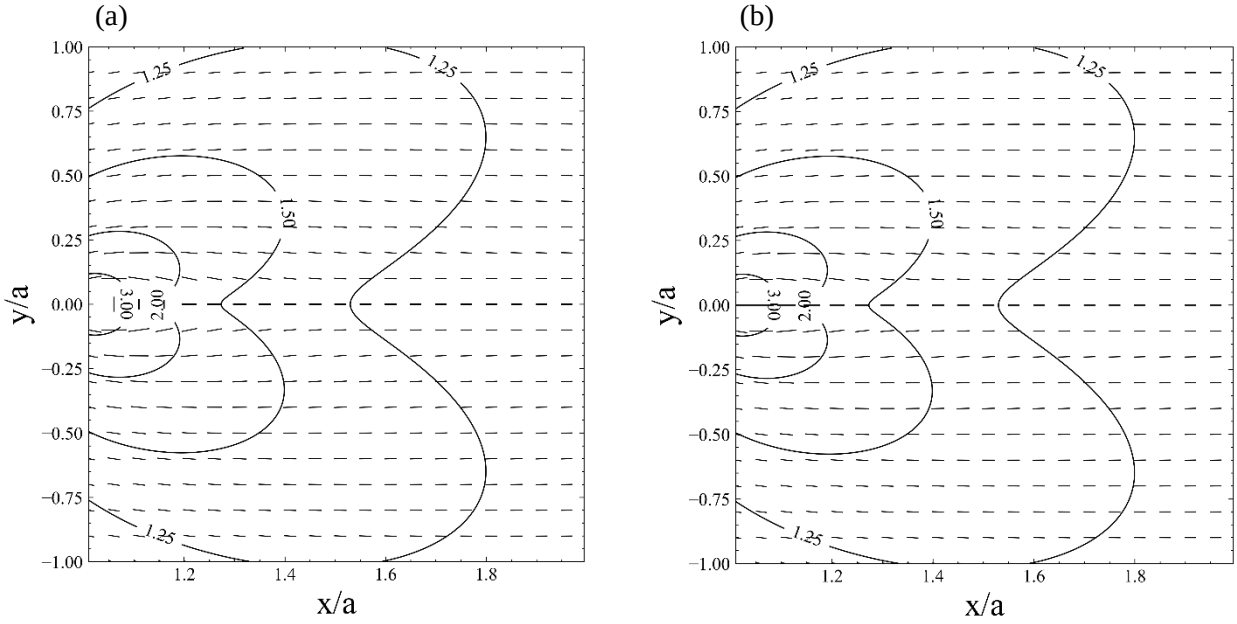


Figure 3.2: Contours of major principal stress (σ_1/σ_∞) and line segments parallel to the major principal plane for (a) the static Griffith crack, and (b) the moving Griffith crack with $V_{MF} = 0.5C_s$. The values selected for σ_∞, V_{MF} and C_s are explained in Section 3.6. Anticipating this section, these values are the same as those used in the *base case* of the sensitivity analysis.

3.2.2 APPLICATION OF STRESS FIELD SOLUTIONS

The MF is modelled as a moving Griffith crack, thus the perturbation of the stress field ahead of the MF is generated from the equations for the moving Griffith crack in Appendix A.

Changes in stress resulting from μCs are modelled as static Griffith cracks using the static Griffith crack formulae in Appendix A. This was done because unbalanced stresses emerge in the Yoffe stress field solution for large enough crack velocities, however, for μCs it was assumed that μC growth velocity is relatively small compared to the MF.

3.3 VOIDS AND MICROCRACKS

3.3.1 VOIDS DISTRIBUTION

A void is a defect from which μCs can germinate. Voids can be generated from forming, material stress history or time-dependent degradation. These processes are uncontrolled leading to irregular void orientation and geometry, and stochastic void locations. The simulation was designed with the capability of distributing voids in a deterministic grid or a stochastic voids space. This research uses a square grid for locating voids to decrease the simulation length required to achieve steady state results. A schematic of the possible void distribution routines is available in Figure 3.3.

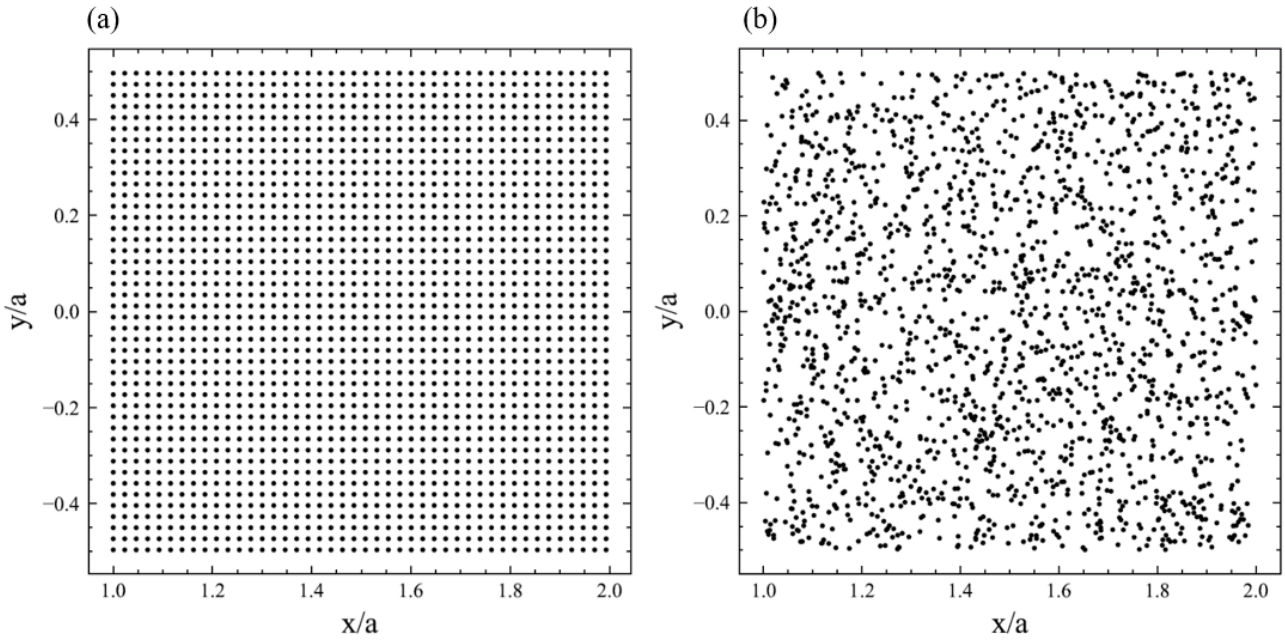


Figure 3.3: Void distribution methods for a constant average density of voids: (a) deterministic square grid, (b) stochastic void space.

3.3.2 OPENING VOIDS AND MICROCRACK GERMINATION

It is hypothesised that within a voids population there is varying resistance to μC nucleation. To model a heterogenous void population, Weibull Survival probability is used to calculate a critical opening stress, σ_{crit}, represents the resistance of a void to opening. The inverse Weibull Survival Function for calculating σ_{crit} is written,

$$\sigma_{crit} = \sigma_w \left[\ln\left(\frac{1}{RV}\right)\right]^{1/m}, \tag{6}$$

where, σ_w is a scaling parameter that determines the location of the distribution, m is shape parameter that controls the spread and shape of the distribution, and RV is a random variable sampled from the interval $[0, 1]$. For each void, RV is selected from a pseudo-random number generator. Parameters m and σ_w are obtained through material testing.

In the model, voids in the simulation box are *open*ed when the major principal stress, σ_1, at the location of a void, exceeds the critical opening stress, σ_{crit}. Physically described, if the tensile stress applied to the void is large enough, the induced stress concentrations at opposite sides of the void will develop into tension cracks. While, in real materials the void strength may change for different orientations of maximum tensile stress, for simplicity it is assumed that σ_{crit} is constant property of each void. An example of the distribution of σ_{crit} for a population of voids is provided in. The parameters used for the example are $\sigma_w = 3 \times 0.0001E$ and $m = 10$.

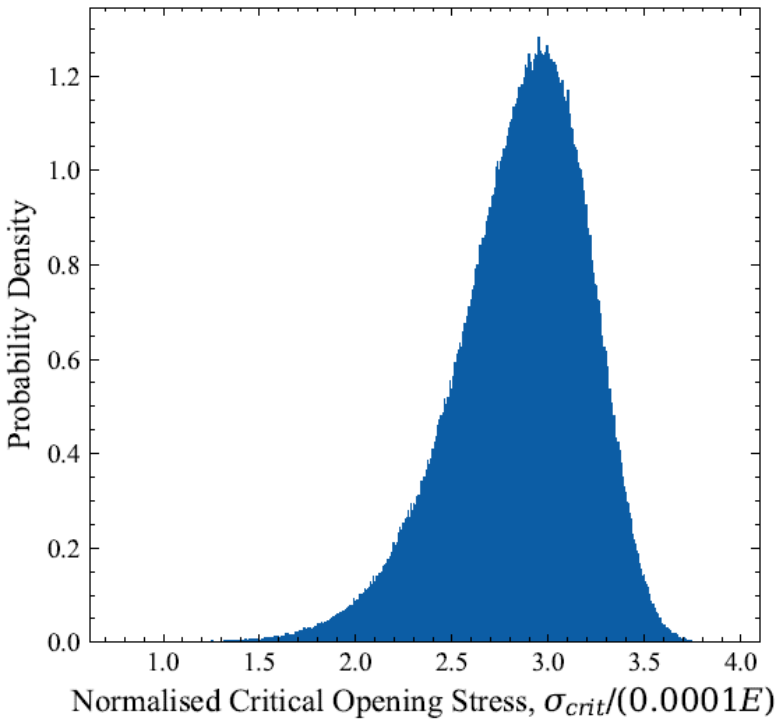


Figure 3.4: Distribution of critical opening stresses, σ_{crit}, for a population of voids.

Weibull statistics are commonly used in failure analysis due to the versatility of the of the shape of the distribution. This allows the model to capture a material-specific distribution of defect strength. However, this assumes that published values for the Weibull shape parameter, m, are appropriate at the meso-scale. While, the Weibull Modulus is widely regarded as a material constant, growing experimental that suggests otherwise [41, 42]. Thus, to take full advantage of the Weibull distribution, the Weibull Modulus should be determined through a meso-scale analysis of μCs. Considering the illustrative approach of this research, the use of ball-park values for input parameters is sufficient to demonstrate fundamental behaviour.

3.3.3 MICROCRACK GROWTH LAW

The properties of μC kinematics are controlled by a growth law. The design of the growth law comprises criteria for the speed and direction μC growth.

Growth Direction

The direction of μC growth follows the Maximum Tensile Stress (MTS) criterion. This means that a microcrack propagates in a direction that is perpendicular to the major principal stress direction, as shown schematically in Figure 3.5. The stresses are measured at the location of the μCs nucleating void. This was done because the size of μCs, the difference in stress between each μC crack tip is small, even when considering the large MF stress gradients near the MF tip.

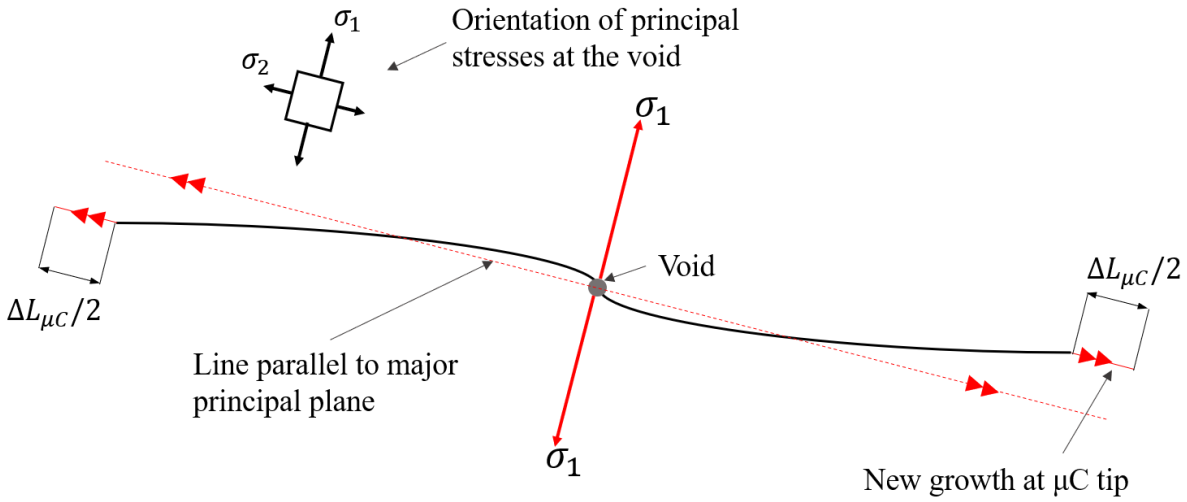


Figure 3.5: An illustrative sketch of μC growth perpendicular to maximum tensile stress at μC tips.

The line segments in Figure 3.2 represent the direction in which a growing μC would propagate at each point. A differentiation is made here between the line segments in Figure 3.2 and μC potential plot. Figure 3.2 is not a μC potential plot as it shows the increments of growth that a μC can accumulate at each point.

An advantage of the MTS criterion is in the ability to determine crack growth direction through assessment of the stress field alone. Other methods are available; however, these are based

on maximum dissipation of energy and the strain energy density which requires development of a constitutive law and knowledge of the strain field. Moreover, the implementation of other crack path criteria is more complex and requires additional assumptions.

Crack Tip Speed

Open voids follow a piecewise continuous μC velocity law. If the stress state at an open void is such that $\sigma_1 \geq \sigma_{crit}$, the μC grows at a constant velocity of magnitude $0.01C_s$, where C_s is the shear wave speed. If $\sigma_1 < \sigma_{crit}$, the crack is arrested. Mathematically, this is written,

$$V_{\mu C} = \begin{cases} 0 & \sigma_1 < \sigma_{crit} \\ 0.01C_s & \sigma_1 \geq \sigma_{crit} \end{cases}.$$

In general, σ_1 increases as μCs approach the MF, therefore, μC growth is continuous until the μC passes the leading tip of the MF. Thus, it is unlikely that crack arrest will occur at any significant extent. The magnitude of the growth velocity was arbitrarily selected to ensure μCs remain micro in comparison to the MF size.

The growth rate of cracks is non-trivial. The presence of complex stress fields precludes a simple law for μC growth, even when other defects are ignored, and a perfectly homogenous elastic continuum is assumed. Formula that describe the crack velocity have been developed by Bouchbinder et al [43] and Dulaney and Brace [44] in terms of limiting velocity, fracture energy and stress intensity factors utilising a Mott energy balance approach. However, the applicability of these equations at the meso-scale is questionable due to energy losses that are unaccounted for in the Mott energy balance at crack initiation [44]. The development of the constant μC growth velocity was done without attempting to manufacture the correct growth velocity. The assumed model is intentionally simple, albeit later versions may incorporate one of the above velocity solutions.

3.3.4 EFFECTIVE MICROCRACK GEOMETRY

The direction that μC crack tips grow depends on the orientation of the major principal stress. As a result, μCs generated by the simulation can be curvilinear. An effective μC geometry needs to be determined for data collection and to calculate stresses in the interaction mechanism (Section 3.4). The effective μC length, $L_{ef,\mu C}$, is calculated as the distance between the μC crack tips. The effective orientation, $\theta_{ef,\mu C}$, is the angle of inclination of the line used to measure μC length measured from the positive x-axis for the MF. The set-out showing the calculation of the effective μC geometry is available in Figure 3.6.

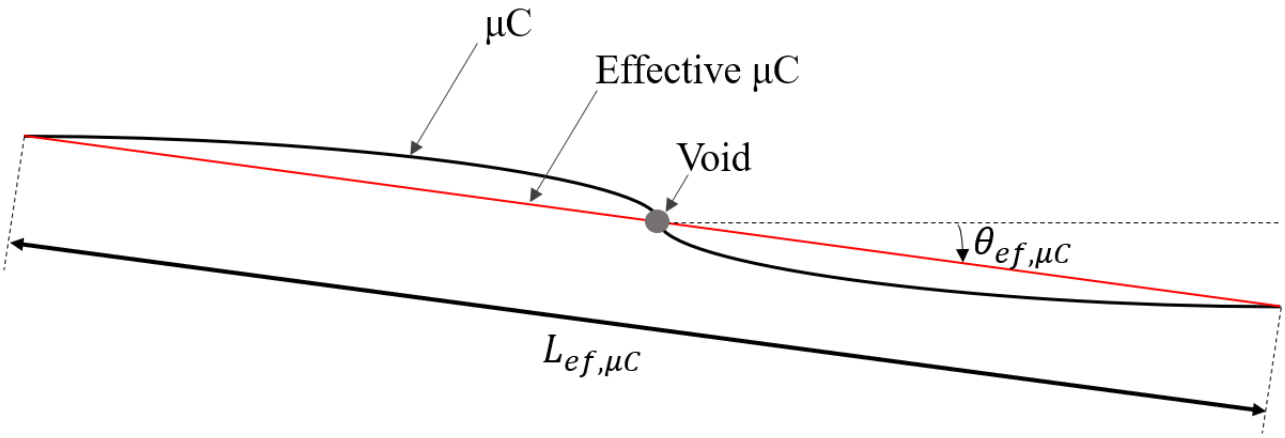


Figure 3.6: Schematic illustration of the calculation for the effective μC geometry.

Since μCs are relatively straight, the use of the crack tips to determine length and orientation will yield a negligible error.

3.4 CRACK INTERACTIONS

3.4.1 KACHANOV METHOD

Interactions are modelled using the Kachanov method which relies on superposition and self-consistency to generate the average stress applied to each crack in the system [45, 46]. The self-consistency approach avoids the need for iteration through the use of transmission factors, Λ. For a system of m cracks, say, the problem becomes one of a system of $2m$ linear equations of length equal to the number of cracks in the interaction. The coefficients of the linear system are the transmission factors, and the variables are the unknown normal and shear tractions, p and τ, on each crack. An example of a pair of linear equations for the normal and shear stress applied to crack, k, is written

$$\begin{aligned}\langle p_k \rangle &= p_{\infty,k} + \sum_{\substack{i=1 \\ i\neq k}}^{m} \Lambda_{ik}^{nn}.\langle p_i \rangle + \Lambda_{ik}^{\tau n}.\langle \tau_i \rangle \\ \langle \tau_k \rangle &= \tau_{\infty,k} + \sum_{\substack{i=1 \\ i\neq k}}^{m} \Lambda_{ik}^{n\tau}.\langle p_i \rangle + \Lambda_{ik}^{\tau\tau}.\langle \tau_i \rangle\end{aligned} \tag{7}$$

Or in compact form,

$$\langle \boldsymbol{t}_k \rangle = \langle \boldsymbol{t}_k^{\infty} \rangle + \sum_{\substack{i=1 \\ i\neq k}}^{m} \boldsymbol{\Lambda}_{ik} \cdot \langle \boldsymbol{t}_i \rangle \tag{8}$$

where, p_∞, τ_∞ are the normal and shear tractions from remote loading (this depends on the orientation of crack k with respect to the applied load), and each Λ_{ik} is a transmission factor to translate normal and shear tractions applied to crack i to the normal and shear stress that crack k experiences as a result. Note that shear stress applied to crack i induce both normal and shear tractions at crack k. The same is true for normal stresses.

The power of the Kachanov method is that it does not require iterating which leads to significant savings in computation time. The approach is particularly accurate for the determination of effective elastic properties and has reasonable accuracy for calculating stress intensity factors [47]. However, accuracy decreases for cracks that are space less than the smallest crack half-length, a, away from each other [47]. Additionally, the model does not capture the phenomenon of crack coalescence (when cracks link up), although the net effect on the stress field when crack intersect is still captured reasonably well [47].

3.4.2 MODIFIED KACHANOV METHOD

Only interactions between each μC and the MF are considered, the inter-μC interactions are ignored. Consequently, the linear system for interactions reduces to a single linear combination of the stresses applied to the MF by the individual μCs. Essentially, the problem simplifies to a simple superposition of stresses. A derivation of the simplified form is available Appendix C.

Traditionally, the Kachanov method is used to calculate average applied normal and shear stresses, however the approach is applied herein to calculate rectangular stresses, $\boldsymbol{\sigma} = (\sigma_x, \sigma_y, \sigma_{xy})$. The stresses are calculated at the leading tip of the MF. If the crack is permitted to change direction, the MF coordinate system used to measure σ_x, σ_y and σ_{xy} changes. Restrictions regarding MF rotation and the degrees of freedom of the MF motion are discussed in Section 3.5.3.

The MF-μC interaction in vectorized form, with $\boldsymbol{\sigma} = (\sigma_x, \sigma_y, \sigma_{xy})$ in the MF coordinate axes, is written,

$$\boldsymbol{\sigma}_{MF} = \boldsymbol{\sigma}_\infty + \sum_i \left(\boldsymbol{\sigma}^I_{\mu C,i} + \boldsymbol{\sigma}^{II}_{\mu C,i}\right) \tag{9}$$

where, $\boldsymbol{\sigma}_\infty$ is the stress applied at infinity, $\boldsymbol{\sigma}^I_{\mu C,i}$ and $\boldsymbol{\sigma}^{II}_{\mu C,i}$ are the stresses at the MF resulting from the average normal and shear tractions applied on μC i.

The sequencing of the interaction calculation is as follows: (1) The MF experiences the load, $\boldsymbol{\sigma}_\infty$, applied to the material, (2) The stress field that is produced by the MF is calculated assuming no defects are present, (3) the average stress along each μC is calculated and resolved into traction stresses for mode I and mode II loading of the μC (this depends on the orientation of the μC in

question), (4) the μC induced stresses at the MF leading tip are calculated for each μC and the result of all μCs is superimposed. The procedure for calculating interaction stresses is illustrated for a single MF-μC interaction in Figure 3.7.

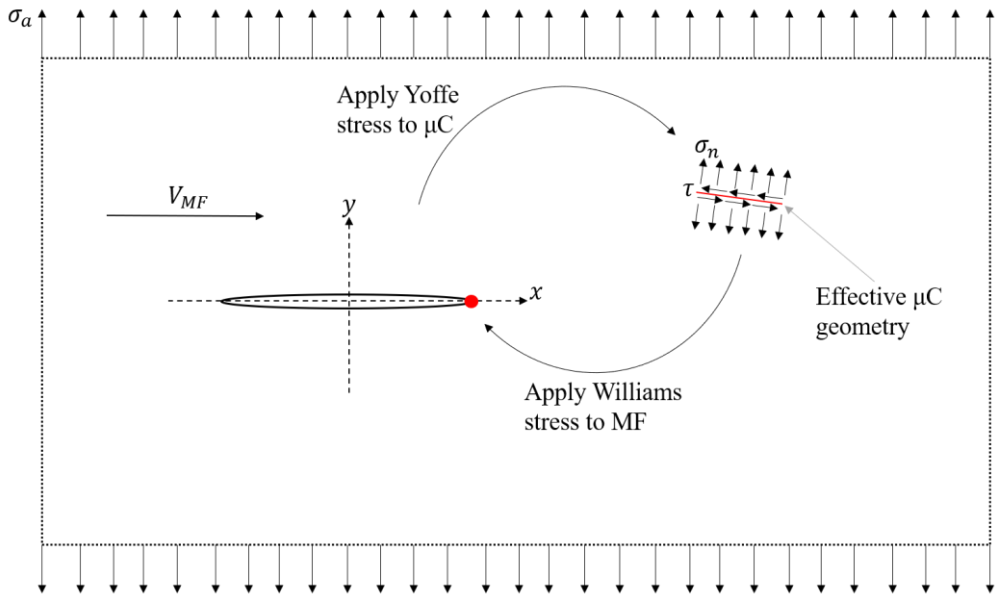


Figure 3.7: Schematic showing the decoupled MF-μC interaction for a single MF-μC interaction. The MF applies stresses on the μC which interprets the MF stresses as traction stresses (acting at infinity) and then transmits stresses back onto the tip of the MF.

The modified Kachanov method ignores the interactions between μCs. However, as noted in Section 2.2.1, the interaction that occur within the PZ characterise its overall effect and therefore should be considered in an interaction model that describes the effect of the PZ on the MF. This can be incorporated in a later version of the model with the full Kachanov interaction method to include inter-μC interactions. For the objectives of this model the fully decoupled interaction approach is suitable for studying the MF as the complex interactions (possibly involving both shielding and amplification) are avoided.

Implementation

The implementation of the modified Kachanov method in the simulation considers only the μCs that are inside the simulation box ahead of the MF. Additionally, since the analytical stress field solutions are not valid in the region between the crack tips, only the stresses resulting from the nearest crack tips of each μC are considered, and interactions that occur behind the near μC tip are ignored. The regions of applicability of the Williams stress field solution are visualised in Figure 3.8.

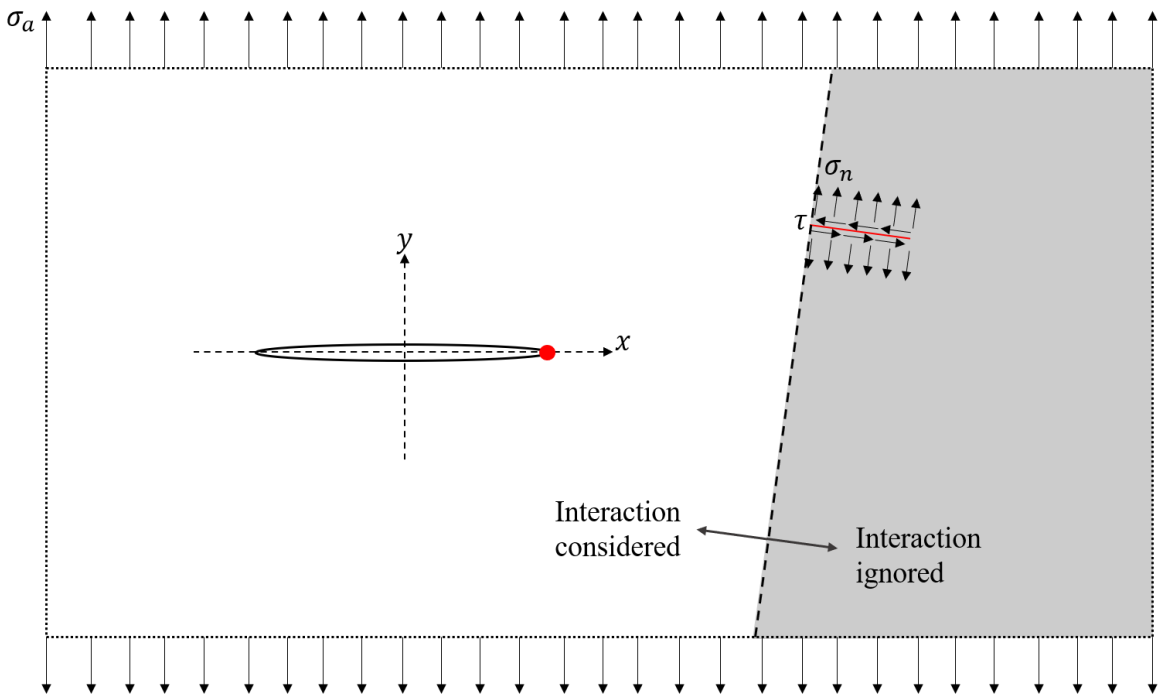


Figure 3.8: Regions of applicability of the Williams stress field solution used for μC in the interaction model.

A distinction must be made between the stresses that are applied to the MF and the stresses that control the MF direction. Interaction stresses and the remote stress, collectively called the *applied stress*, are to be used for determining the MF direction of motion, while the MF stress field is generated by the remote stress alone. This may include mode II loading in addition to mode I if the MF is permitted to rotate.

3.5 MAIN FRACTURE PATH

3.5.1 PATH SELECTION

At each timestep, the MF direction of motion, θ, is determined by applying the MTS criterion to the sum of the applied stress at infinity, σ_∞, and the interaction stresses calculated at the MF tip (σ_a). The angle θ is measured from the positive x-axis of the MF in its initial orientation. This is depicted in Figure 3.9 and the formula for calculating θ is provided in Appendix D.

While it is more realistic to consider the stress field that results from the applied stresses, since Yoffe's solution assumes propagation according to the MTS the result is the same [39]. Thus, even when considering multi-mode loading the applied tractions to the MF give the same direction of propagation as the overall stress field under the MTS assumption.

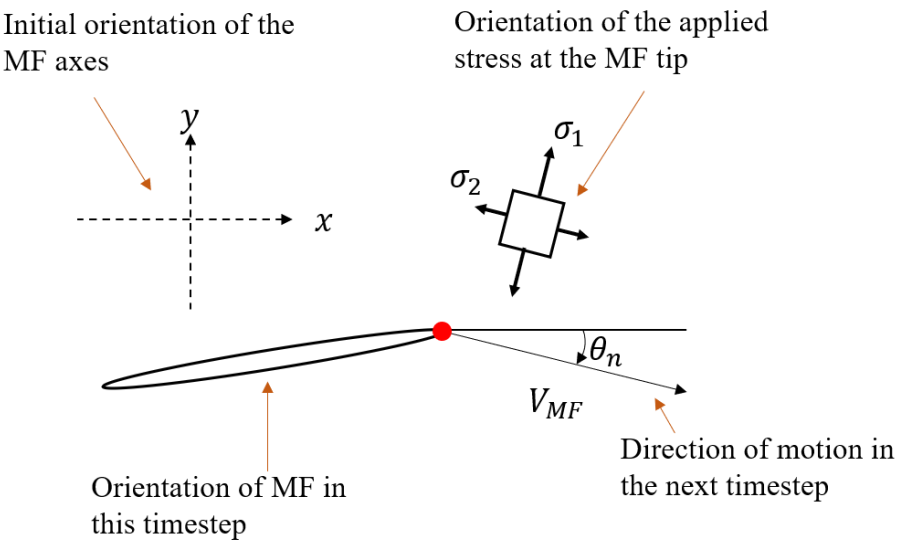


Figure 3.9: Schematic showing the main fracture in the instant before changing direction to move at an angle θ_n with respect to the initial MF orientation.

3.5.2 NAVIGATING MICROCRACKS

Since objects are modelled by the perturbations they create on the stress field, when the MF encounters μCs along its path it simple crosses over them without coalescing with them. Ignoring coalescence avoids additional assumptions required to determine where the MF exits the μC or the stress changes that occur while the MF is jumping between the entry and exit points. Despite this approach, the Kachanov method yields reasonable results for intersecting cracks [47].

3.5.3 IMPLEMENTATION AND DEGREES OF FREEDOM

The MF has an orientation, $\theta(t)$, and a speed, V_{MF}. In the simulation, there three possible approaches for moving the MF were considered. In each approach a different level of restraint is applied to the propagation direction of the MF. The approaches are listed here and illustrated in Figure 3.10.

Approach 1: The MF is restricted to horizontal motion and zero rotation.

Approach 2: The MF is restricted to horizontal motion and rotation is permitted.

Approach 3: The MF propagates radially in the direction that it is oriented in.

In Approach 1, θ is a pseudo-orientation that is calculated using the MTS criterion and is recorded at each time step. In Approach 2 and 3, θ is the actual orientation of the MF.

The use of these approaches is to navigate possible instability issues resulting from μCs that the MF may cross. Results presented herein assume Approach 1 is used.

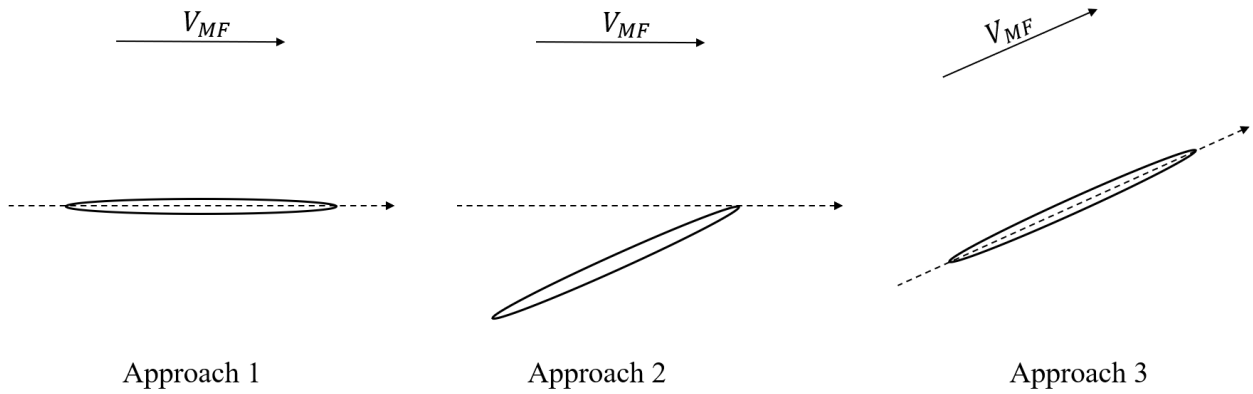


Figure 3.10: Schematic representation of the approaches considered for controlling the MF path.

Note that, in Approach 1, the remote stress, σ_∞, is always applied perpendicular to the MF direction of motion. However, if Approach 2 and 3 are considered, then the normal applied stress at infinity will translate to normal and shear stress if the MF is inclined.

3.6 IMPLEMENTATION AND PARAMETERS

3.6.1 MODEL UTILISATION

The model was used study PZ formation and MF-μC interactions. The testing regime involved a base case parameter set and sensitivity analysis parameters.

To demonstrate the operation of the simplified Kachanov interaction mechanism and the behaviour of individual μCs, the simulation was run with a single μC located at varying vertical displacements above the MF major axis. To achieve a qualitative description of the PZ, an exploratory analysis is conducted using the base case parameter set. It was expected that the outcome will provide insight into possible approaches for predictive PZ models. In each sensitivity analysis, a single parameter was changed, and the results were compared to the base case. The focus was to capture a broad spectrum of behaviour. Thus, parameters were varied by up to an order of magnitude. The parameters involved in the sensitivity analysis include, (a) the voids density, to study the effect of different levels of interaction, while maintaining the size and shape of the PZ, (b) Weibull parameters, m and σ_w, to assess the effect of process zone size and shape. The values selected for each parameter are outlined in Section 3.6.2.

3.6.2 PARAMETER SELECTION

The parameters involved in the model and the values used for the simulation are summarised in Table 1. Material parameters were based on typical values expected for a 24 MPa concrete. However, given the model assumptions (perfectly linear elastic, homogenous, isotropic material), the medium that is studied is rather fictitious and the results do not attempt to model how a 24 MPa concrete specimen would behave.

Parameters that control the μC population were the subjects of the sensitivity analysis. The μC population is dependent on the void population, void strength and the stress that is applied to them. As such the parameters involved in the sensitivity analysis include void density, ρ_{voids}, the Weibull distribution parameters, σ_w and m, that define the distribution from which σ_{crit} is sampled from, and the MF size and velocity for the stress distribution. The applied stress, σ_∞, is also important in the stress distribution, however it is held constant in this research.

The applied stress, σ_a, and MF half-length, a, were selected to ensure that the fracture toughness, K_{Ic}, calculated using Eq. (3) exists within the range of values for concrete. In this case, K_{Ic} is 0.79 $MPa \cdot m^{0.5}$ for the base case in Table 1, which is within the range of available experimental results [48, 49, 50, 51].

The Yoffe stress field solution assumes that the crack velocity does not exceed the shear wave speed, C_s. The μC velocity was selected as to ensure the maximum possible length of a μC is small compared to the MF size. Void density in the base case was set by tuning the simulation to obtain a reasonably sized μC population.

The Weibull shape parameter, m, was selected by considering the distribution experimental values for concrete. Since an appropriate value for σ_w was not known, it was selected using the rationale that follows. The material is already stressed to a certain level. If voids had extremely low strength, then microcracking would be dense everywhere and there would be no PZ. For the PZ to develop stress level must increase and the only location where the stress levels are sufficiently high enough are ahead of the MF. Moreover, considering the stress intensity factor K_I is proportional to $\sqrt{a}$, and since μCs are small, μCs that experience the same background stresses as the MF will have a smaller stress intensity factor. To reach the critical stress intensity factor, the stress about the μC must increase. The additional stress is supplied by the MF. Therefore, Weibull scale parameters σ_w was selected by ensuring the distribution critical stresses was such that voids would not open for stresses similar or smaller magnitude the applied stress, σ_∞, that is propagating the MF. While it is not possible to prevent $\sigma_{crit} < \sigma_\infty$ for every single void, the distribution was set such that few voids would have such low strength.

Table 1: Summary of simulation parameters and values for the base case and sensitivity analysis.

Parameter Description	Parameter	Base Case	Sensitivity Analysis***
Material Parameters			
Elastic Modulus	E (MPa)	20,000	-
Poisson's Ratio	ν (-)	0.25	-
Solid Material Density	ρ_s (kg/m^3)	2000	-
Weibull Scaling Parameter	$\sigma_w/E \times 10^{-4}$ (-)	3	2, 4
Weibull Shape Parameter	m (-)	10	4, 24
MF Geometry and Applied Stress			
MF HALF Length	a (m)	0.05	0.01, 0.1
Applied Load	$\sigma_\infty/E \times 10^{-4}$ (-)	1	-
Kinematic Parameters			
Normalised MF Velocity	V/C_s (-)*	0.5	-
Normalised μC Velocity	$V_{\mu C}/C_s$ (-)	0.05	-
Voids Parameters			
Void Density	$\rho_{voids} \times 10^6$ $(voids/m^3)$	0.75	0.075, 7.5
Simulation Parameters			
Time-Step	dt (s)	50×10^{-9}	50×10^{-9}**
Propagation Approach	-	1	-
Voids Distribution Method	-	Deterministic Grid	-

* C_s is the shear wave speed (m/s).

** time-step is adjusted to improve resolution of results when the MF velocity is different from the base case value.

***Each value in the *Sensitivity Analysis* column represents a unique sensitivity analysis, whereby all other values for the simulation are taken from the *base case* column.

4 RESULTS AND DISCUSSION

4.1 INDIVIDUALS IN A MICROCRACK POPULATION

Interactions between the MF and individual μCs were considered to compare the behaviour of μCs at heights of 0.01a, 0.05a, 0.1a and 0.5a (denoted $\mu C_{0.01}$, $\mu C_{0.05}$, $\mu C_{0.1}$, $\mu C_{0.5}$) above the MF major axis. Voids strength, σ_{crit}, was set at $1.25\sigma_{\infty}$ to allow μCs time to grow (Approach 1 MF motion).

The applied stress, $\boldsymbol{\sigma_a} = (\sigma_{a,x}, \sigma_{a,y}, \sigma_{a,xy})$, that was calculated at the main fracture tip is shown for each microcrack in Figure 4.1. The *main fracture distance travelled* represents the distance travelled since an interaction was first registered with the MF. It can be seen that the time for interaction varies for each μC.

Moving away from the MF tip the decrease in interaction stresses ($\sigma_a - \sigma_{\infty}$) occurs over orders of magnitude and different regimes of interaction behaviour occur. The variation of stresses for $\mu C_{0.01}$ demonstrates a regime that is distinct from the other μCs for all stress components. The stress variations for $\mu C_{0.05}$, $\mu C_{0.1}$, and $\mu C_{0.5}$ can be obtained from each other from shifts and stretches of the curves in Figure 4.1. While, for $\mu C_{0.1}$, stresses follow different curves and even different signs altogether. For example, in Figure 4.1b, the interactions with $\mu C_{0.01}$ increase stress at the MF tip, whereas $\mu C_{0.05}$, $\mu C_{0.1}$, $\mu C_{0.5}$ generally induce a stress reduction. This shows that the distance of a μC from the MF not only controls the strength of interaction but also the way the interaction takes place.

In this section, and for the remainder of the results, applied stresses (interactions stresses + σ_{∞}) are normalised with the stress applied at infinity, σ_{∞}. This was done because in the Yoffe and Williams stress field solutions, stresses are proportional to the applied stress, σ_{∞}. This means stresses applied to μC are proportional to σ_{∞}, and by extension, the stresses that microcracks apply back onto the MF are also proportional σ_{∞}. If σ_{∞} is varied, the resulting stresses from individual interactions simply scale by the proportion of the change in σ_{∞}. A proof supporting this explanation is provided in Appendix F.

While the MF is restricted to straight line motion and zero rotation, the stresses components in Figure 4.1 can be used to calculate the orientation of the major principal plane and thus a fictitious MF direction of motion, θ, can be obtained. This allows for observation of θ along the horizontal MF path and enables calculation of a hypothetical vertical displacement of the MF as shown in Figure 4.2. The formulae for calculating θ and vertical displacement of the MF is lengthy

and is available in Appendix D. The similarity between the shear stresses in Figure 4.1c and the calculated MF orientation in Figure 4.2 is expected since the shear stresses induce the change in MF direction in *Approach 1* MF propagation. This is because, the remote stresses applied to the fracture are in the x- and y- directions (at large x and large y; $\sigma_y = \sigma_1 = \sigma_\infty$ and $\sigma_x = \sigma_2 = 0$), therefore, shear stresses change the orientation of the principal plane. The amount that the orientation changes depends on the size of the Mohr's circle ($\sigma_1 - \sigma_2$).

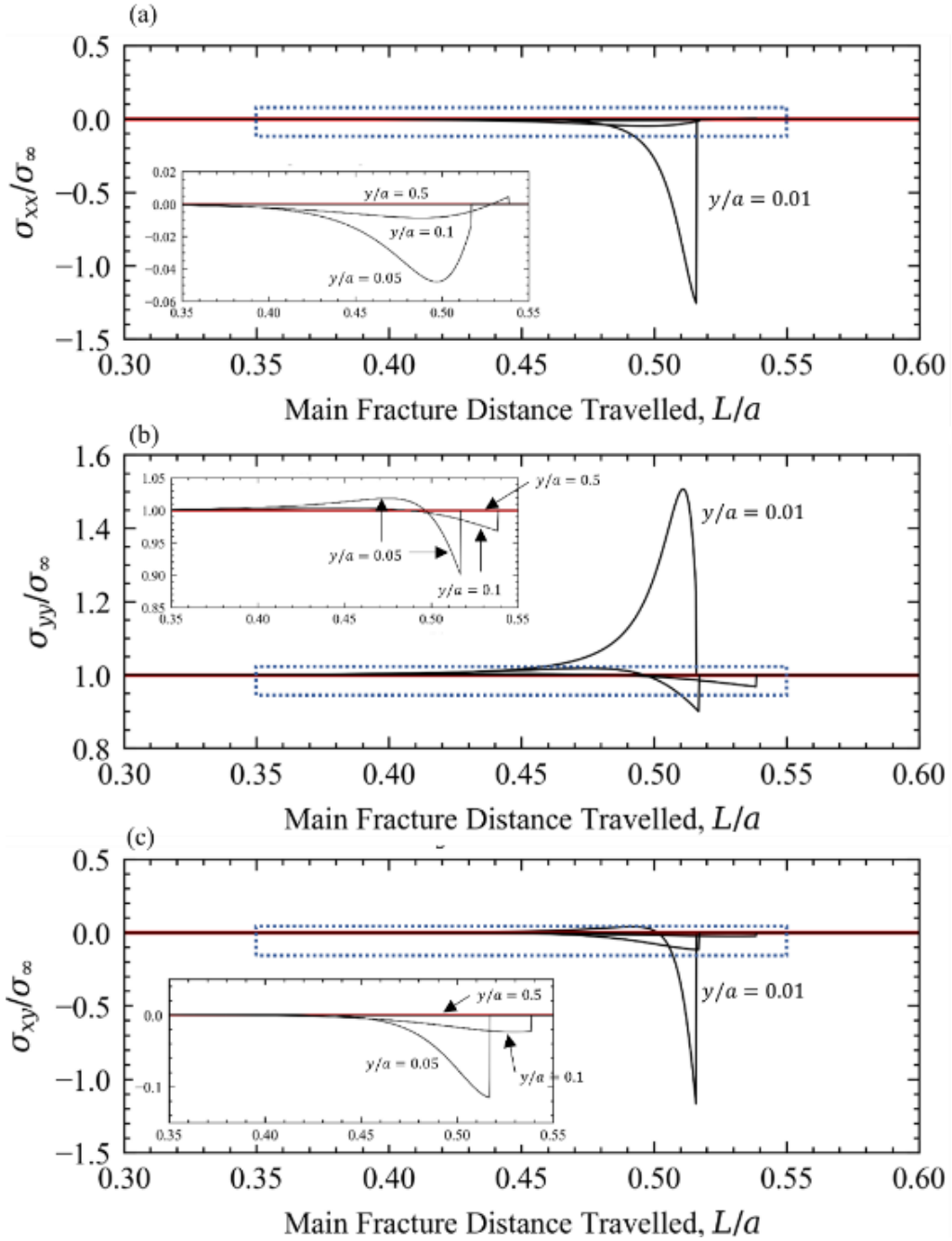


Figure 4.1: The evolution of stress state for stresses applied to the tip of main fracture that interacts with singular microcracks. Each curve belongs to a microcrack with a unique displacement, y, above the main fracture major axis. The rectangular stresses $\sigma_x, \sigma_y, \sigma_{xy}$ shown in (a), (b), and (c) respectively, and are normalised with the stress applied at infinity, σ_a. Insets for each subplot show an expanded view of the three microcracks are displaced further from the main fracture.

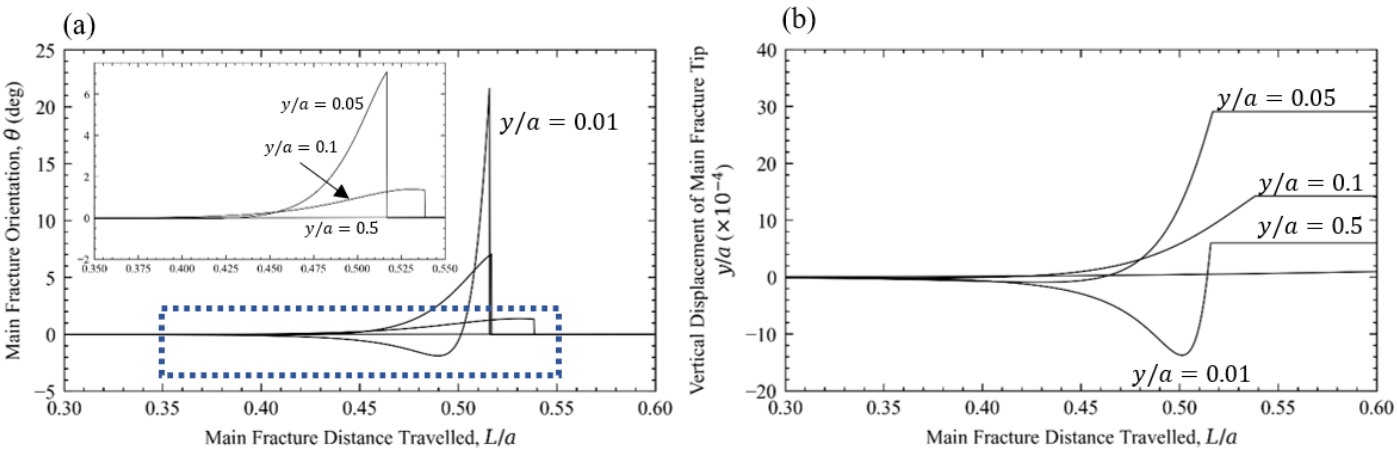


Figure 4.2: Results from analysis of a fracture passing individual microcracks with a unique hypothetical displacement, y, above the fracture major axis. The horizontal axis represents the distance travelled since an interaction was first detected. (a) A fictitious direction of main fracture motion determined from stress applied at the tip of the main fracture. (b) Hypothetical fracture path for a fracture that is restricted to straight line motion and interacts with microcracks.

The MF path follows different regimes of motion depending on the vertical displacement on interacting μCs. Nearer μCs induce significant repulsion followed by a short and sharp attraction, whereas μCs further from the MF induce attraction over most if not all of the interaction. Similar to the stresses, the amount that μCs attempt to change the course of the MF, indicated in the magnitudes of the curves in Figure 4.2a, decreases rapidly (over orders of magnitude) as μCs are located at larger distances above the MF, with negligible interaction potential of the μC that moves along the horizontal line $y/a = 0.5$.

To summarise, μCs with small vertical displacement from the MF major axis display a different regime of interaction compared to μCs that are further away. Near μC increase the normal applied stress on the MF and display strong repulsion followed by attraction of the MF as it approaches. Moving further away from the MF major axis, μCs attract the MF for the duration the interaction and reduce the normal applied stress at the MF tip.

4.2 PROCESS ZONE PROPERTIES

A fine grid containing 250,000 voids at 100 unique y values located above the MF major axis was passed through the MF simulation box to investigate the evolution of μC length and orientation and the density of μCs ahead of the MF. The grid of points that represents the locations visited by voids is produced in Appendix H. The size of the simulation box was optimised so that the process zone was fully contained. This decreases computational expense of considering voids that are far from the MF, and also mitigates potential issues related to voids with $\sigma_{crit} < \sigma_{\infty}$ forming μCs that grow at constant velocity at unreasonably low stresses.

Distributions of Microcrack Properties along Pathlines

At each grid point, the ratio of the number of open voids and the total number of voids to visit that grid point is calculated. This ratio is referred to as the *μC ratio*, which is defined as the ratio of microcracks to voids at a point in the coordinate reference system of the main fracture. At each point distributions can be produced for μC effective length and effective orientations of the μCs that are amassed by the 2500 voids that pass each point. To investigate the evolution these distributions as μCs approach the MF, their variation was analysed at evenly spaced points along horizontal lines as indicated in Figure 4.3. The horizontal lines represent pathlines as μCs move along them toward the MF. The distributions at points along the same pathline were plotted together, as shown in Figure 4.4. The curves that represent each distribution were calculated using a Gaussian Kernel Density Estimator with the bandwidth (smoothness) of each distribution determined using Scott's Rule.

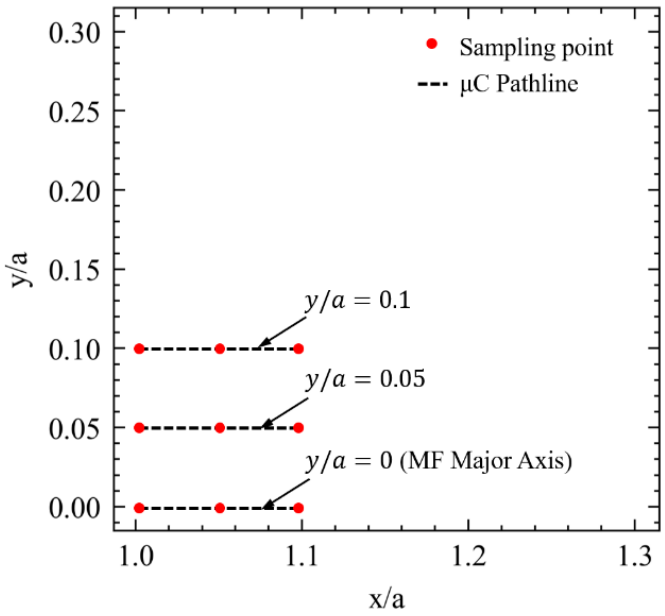


Figure 4.3: Sampling points for the pointwise-distributions of microcrack effective length and effective orientation. Horizontally collinear sampling points are considered together to track the evolution of the distributions as points approach the main fracture.

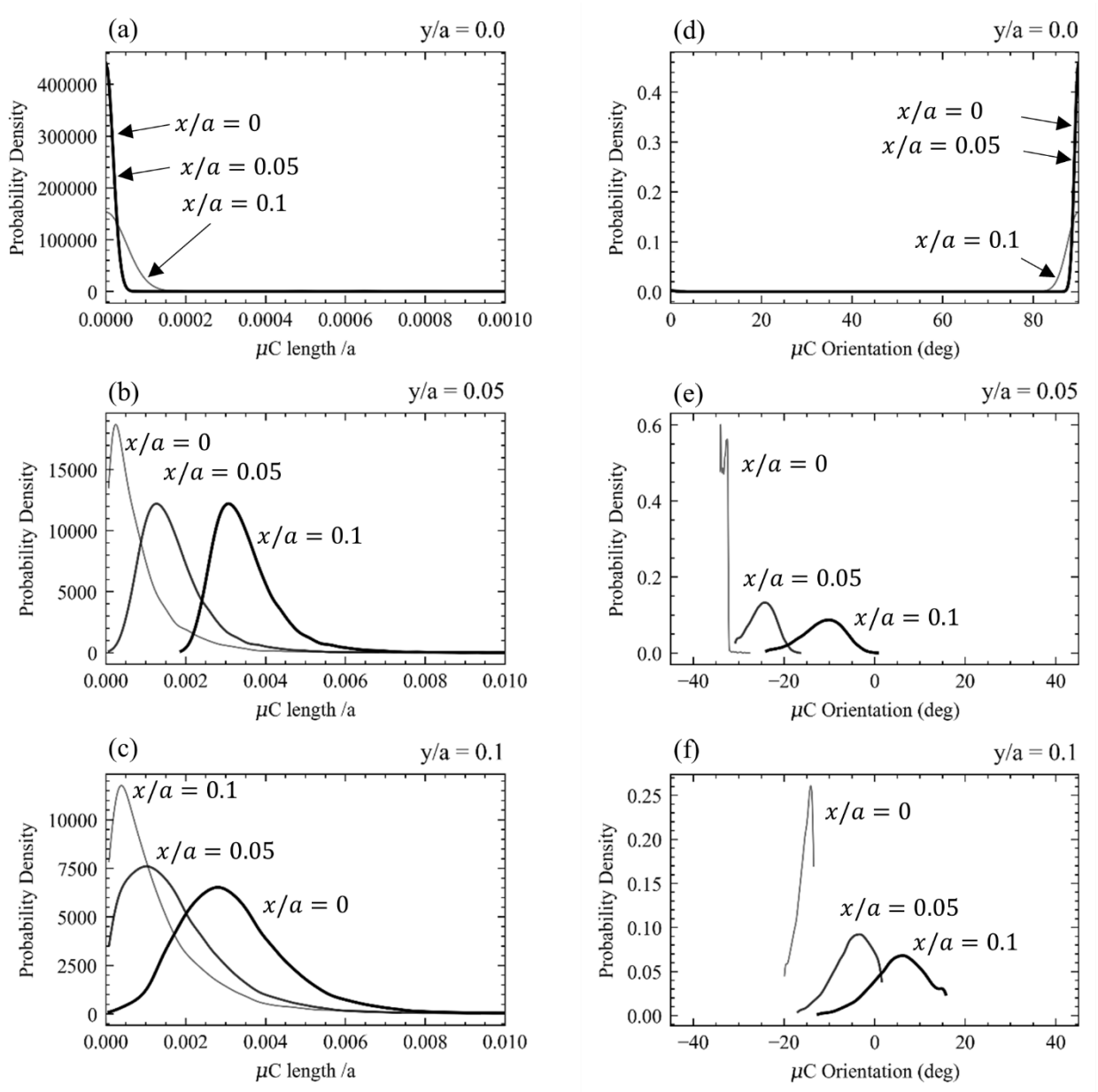


Figure 4.4: Kernel density estimation of the effective length (a)-(c) and orientation (d)-(f) distributions for all the microcracks that move along the pathlines indicated in Figure 4.3. Results are grouped by points that are horizontally collinear to demonstrate the evolution of the distributions as microcracks approach the main fracture.

For all distributions for length and orientation, as μCs approach the MF, distributions approach an ultimate state.

The location and spread of the data changes, but the shape of the distribution remains relatively consistent as existing μCs grow and more μCs are added to the distributions while moving towards the MF. Moreover, the shape of the distributions along different pathlines are similar.

Spatial Distribution of Mean Microcrack Properties and the Microcrack Ratio

The arithmetic mean of length and orientation distributions was calculated at each point in the simulation box and plotted with the microcrack ratio in Figure 4.5 to visualise the spatial variation of μC geometry and population properties.

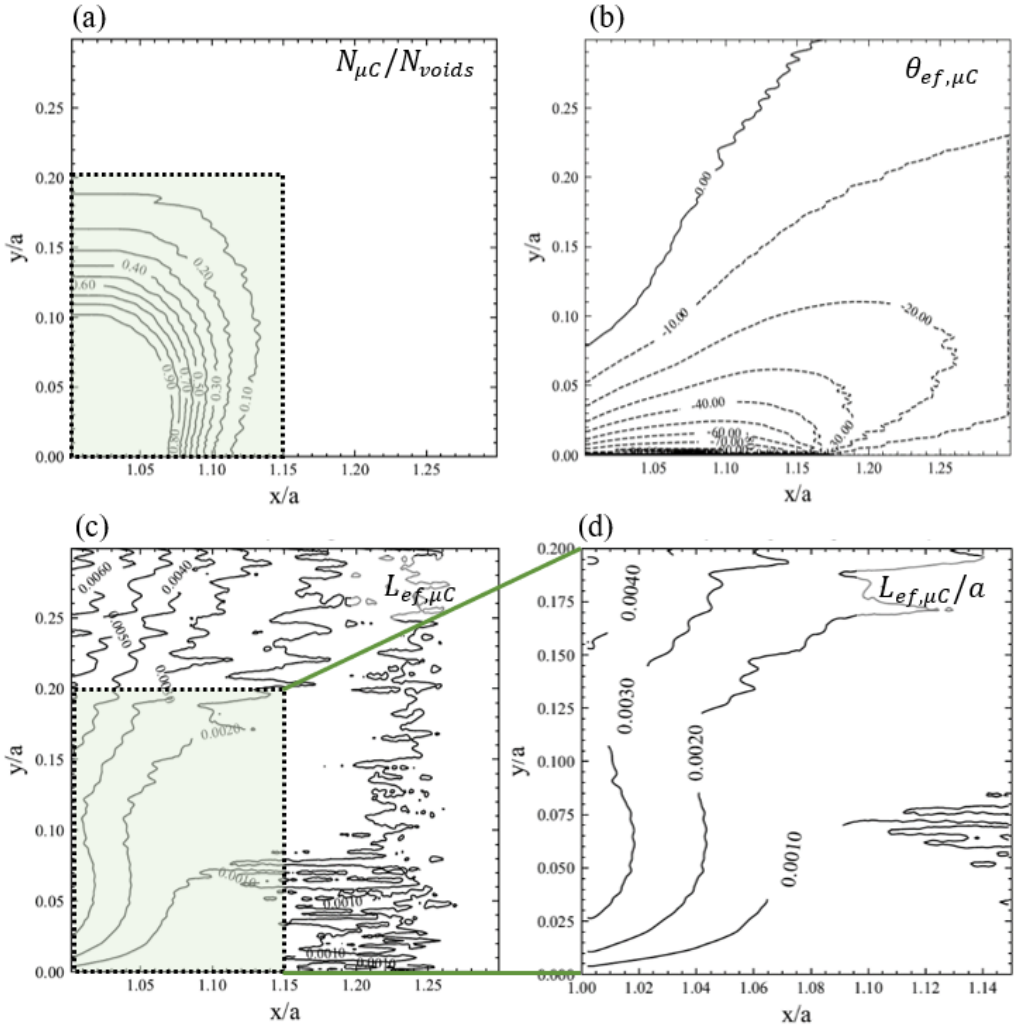


Figure 4.5: Results from the analysis of microcrack populations ahead of a fracture. (a) Spatial distribution of the microcrack ratio. (b), (c) The arithmetic mean of effective microcrack orientation and length for all microcracks that pass each point. (d) Zoom in on region in (c) that is less affected by noise.

Figure 4.5a & b are reminiscent of the contours for MF principal stress in Figure 3.2a and the rotation to σ_2 direction (Figure 7.4, Appendix B). This is most prominent at larger values of x/a. Although, when approaching the MF, μCs age and the distribution of μCs develops resulting in deviations. The same can be said for Figure 4.5d where $y/a < 0.1,\ x/a < 1.1$, however outside this region the level of noise is significant and μCs appear longer. This is a result of voids with $\sigma_{crit} < \sigma_\infty$ opening upon entry into the simulation box and the associated μCs then growing a $V_{\mu C}$ for the lifetime inside the simulation box. Herein lies the problem with a constant $V_{\mu C}$, since when $\sigma_{crit} < \sigma_\infty$, μCs become unreasonably large by the time they reach the MF tip. However, when enough μCs open, this effect is smoothed out as seen near the MF tip ($y/a \to 0,\ x/a \to 0$). This is

a possible source for additional noise, particularly when considering crack interactions which is further discussed in Section 4.4.

Process Zone Properties and A New Characterisation of the Process Zone

Figure 4.5b, c, d allow for an understanding of the changes in μC properties in space, while Figure 4.4 provides the distributions of all the probable μC properties of the μCs at each point. The results allow for a characterisation of the μC properties that comprise μC populations and the distributions in particular show the way these properties evolve over time. A Population Balance Equation (PBE) can be used to provide an analytical description of the evolution the μC populations that belong to the distributions in Figure 4.4. PBEs describe the mean behaviour of the population thereby giving the typical properties of a μC at each point along a pathline. This can be combined with the μC ratio to generate a mean μC population state depending on material and loading conditions. By being able to predict the μC populations, the amount of energy dissipated in the formation and evolution of μC populations can also be predicted. If μC-dissipated energy can be incorporated in Irwin's energy release rate, G_c, then the strength predictions of Linear Elastic Fracture Mechanics can be improved for quasi-brittle materials.

An analytical solution for μC population generation allows for a description of the PZ in continuum mechanics. Currently, discrete stochastic μC models give better strength estimates compared to continuum models. As was discussed in Section 2.3.2, discrete models are computationally expensive, and therefore avoided in practice. Analytical description of the PZ would allow for the incorporation of μC populations in continuum approaches. To achieve this further research is required to link μC populations to the structural response.

4.3 MAIN FRACTURE INTERACTIONS WITH A MICROCRACK POPULATION

The MF-μC interactions for a population of μCs was investigated to analyse the impact of μC populations on the MF trajectory and the applied stresses at the MF tip. The base case parameters in Table 1 are used for the analysis of stress changes and to establish a description of the MF path roughness. A sensitivity analysis was done for the MF path roughness to analyse the changes in the roughness parameter.

The size of the simulation box was set to enclose the PZ in full and allow additional distance for μCs with low critical opening stress, σ_{crit}. The length of MF propagation was set to obtain steady state results for the standard deviation, σ_θ, of the main fracture trajectory measured at each point. For the base case, steady state was achieved after a MF propagation distance of 50a. This was

not a constant length and some sensitivity analyses required longer propagating for steady state results. Results were declared steady state when the standard deviation of the MF directions accumulated along the path, denoted, σ_θ, reached a constant value. The running calculation of σ_θ for the base case is provided in Appendix I to demonstrate the attainment of steady state.

For all simulations, voids were distributed in a regular square grid such that the MF major axis was equidistant to the nearest rows of voids above and below the MF. This was done to avoid asymmetry and to avoid voids along the centreline from dominating the stresses and introducing periodicity into the results from the regular spacing of voids. Additionally, since the accuracy of the Kachanov method diminishes for near the MF tip, the reliability of the model is improved by avoiding μCs along the centreline for *Approach 1* type MF propagation.

4.3.1 FRACTURE TRAJECTORY AND PATH ROUGHNESS – BASE CASE

Microcrack Populations and the Process Zone

The grid used for μC potential locations and the Weibull distribution used to for assigning a critical void-opening stress to each μC were presented in the Methodology, Figure 3.3a and Figure 3.4, respectively. A representation of the process zone for the base case parameters is presented in Figure 4.6. The μCs in the figure have been exaggerated to be double their actual size in the base case for better visualisation.

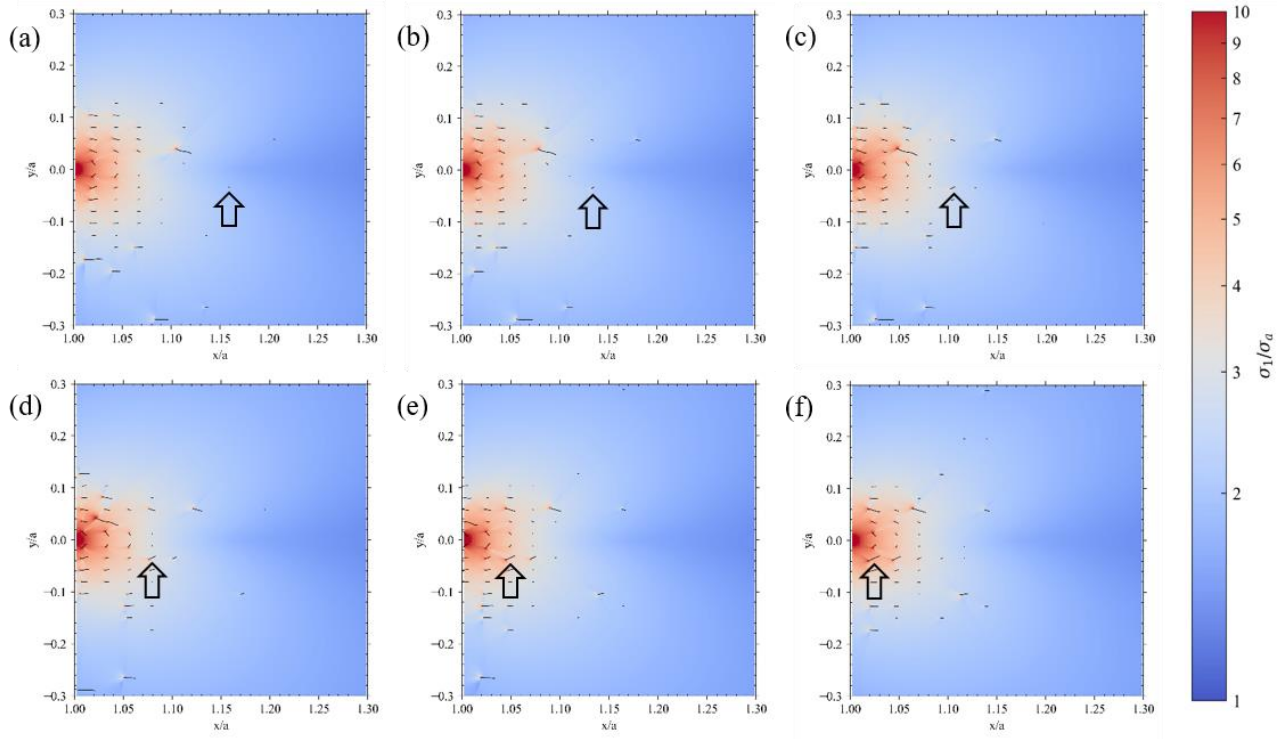


Figure 4.6: Animation frames showing microcracks moving towards the main fracture within the simulation box. The growth of a microcrack is tracked from (a)-(f). The figure is for illustration only, the microcrack growth velocity was doubled to 0.1Cs to exaggerate their size and changes in stress.

The animation slides in Figure 4.6 highlight the lengths and orientations of μCs that were investigated in Section 4.2 as well as the instantaneous changes in asymmetry of the μC population that interact with the MF. The μCs at each point can be thought of as a random sample from similar distributions to those shown in Figure 4.4 for μCs moving along different pathlines.

Interestingly, the PZ appears ordered unlike the schematic representation in Figure 2.4. The closer μCs are to the MF major axis, the more inclined they are. This suggests that μCs may also limit lateral motion of the MF since, if coalescence of the MF with μCs is considered, when a MF enters into a μC, the exit point will be closer to the original MF major axis. Interactions in this research are limited to stress field perturbations, the displacements and notions of coalescence are ignored.

Visualising the Fracture Path

The theoretical vertical displacement of the MF was calculated using the fictitious MF orientations calculated from applied stresses. This was done by integrating the vertical incremental displacements that precede each iteration as follows,

$$y_n = \sum_{i \leq n} \Delta y_i = \sum_{i \leq n} V_{MF} \cdot \Delta t \cdot \sin \theta_i = V_{MF} \cdot \Delta t \cdot \sum_{i \leq n} \sin \theta_i, \tag{10}$$

where; θ_i are all the MF orientations up to and including the current point, Δt is the timestep, and V_{MF} is the MF velocity. An example of a path generated using base case parameters is provided in Figure 4.7. It is important to note that the MF path produced by measuring stresses along the line $y = 0$, since the MF was restricted to *Approach 1* propagation.

The roughness of the MF occurs at length scales that are significantly smaller than the MF. This can be understood by considering the zoomed inset in Figure 4.7 corresponds to a sample length equal to the MF length and the vertical scale has a 200x exaggeration. The randomness of the MF path exists on multiple length scales. Peaks and troughs are visible at both the global and local scale.

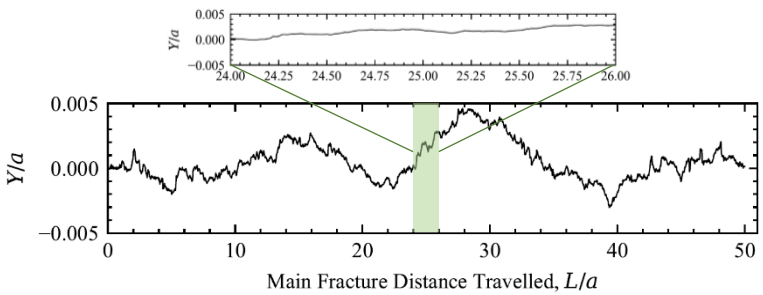


Figure 4.7: Fracture path of a main fracture restrained to straight line motion through a field of voids. The fracture path has a $5000\times$ vertical exaggeration and the inset has a $200\times$ vertical exaggeration. Note that the length over which the zoom is sampled is equal to the MF length.

Fracture Path Roughness Model

To investigate the roughness of the MF path, the hypothetical crack orientations from each iteration were used to generate the density distribution in Figure 4.8. A Gaussian distribution was fit to the θ density data. The Gaussian fit was achieved by fixing the mean at zero and allowing a single degree of freedom, the standard deviation. The fitting procedure implemented the Python module *lmfit* which uses nonlinear least-squares minimisation. The assessment and discussion for the quality of the Gaussian fit is provided in Section 4.3.2. The standard deviation, σ_θ, of the fitted Gaussian was used to quantify roughness for comparison in the sensitivity analysis in Section 4.3.2. Additionally, the validity of using σ_θ relies on the quality of fit of the Gaussian. Basing comparisons on, σ_θ, is sufficient for this exploratory analysis, however, there exist established roughness measures such as the roughness exponent, ζ. Future research should incorporate these roughness measures to enable comparisons to available experimental and numerical results.

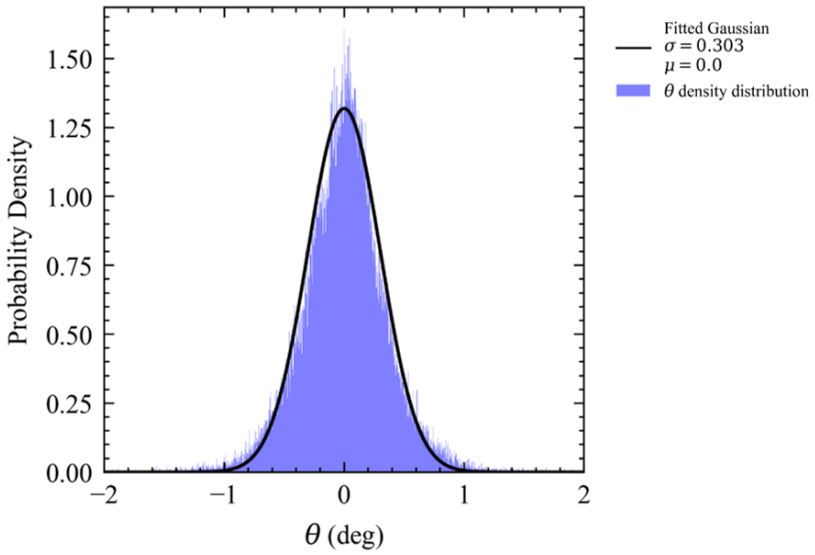


Figure 4.8: Density distribution of fracture direction and Gaussian fit for the base case simulation parameters. Nonlinear least squares regression was used for fitting the Gaussian.

The fitted Gaussian in Figure 4.8 demonstrates a reasonably fit for the density distribution of MF orientations, although this diminishes towards $\theta = 0$. A Gaussian was also fitted to the θ-distributions for the sensitivity analyses in Section 4.3.2 and a sample of them are provided in Appendix I. These also demonstrate a varying closeness of the fit with some distributions matching well and others being poorly described by its fitted Gaussian. In particular, deviations occur about $\theta = 0$ and at the tail ends of the distribution. It is possible that greater densities at zero are related to the immediate response of the MF orientation to the applied loads. This means that, regardless of the orientation in the previous iterations, if interactions become insignificant, the MF will immediately resume a horizontal orientation induced by the remote stress, σ_∞. Thus, it is reasonable to expect over-reporting of the probability density at zero. This can be verified by relaxing the restrictions on MF motion.

4.3.2 PATH ROUGHNESS – SENSITIVITY

The influence of the PZ on the MF trajectory was analysed using a sensitivity analysis with the void density, ρ_{voids}, Weibull parameters, m (shape parameter) and σ_w (scale parameter), and, the MF velocity, V_{MF}. The aim was to investigate a spectrum of behaviour; therefore, parameters were distributed over a broad range. In each sensitivity run, a single parameter was changed, while the remaining parameters reflected the base case values. The fracture path roughness parameter was calculated, and the results are presented in Figure 4.9.

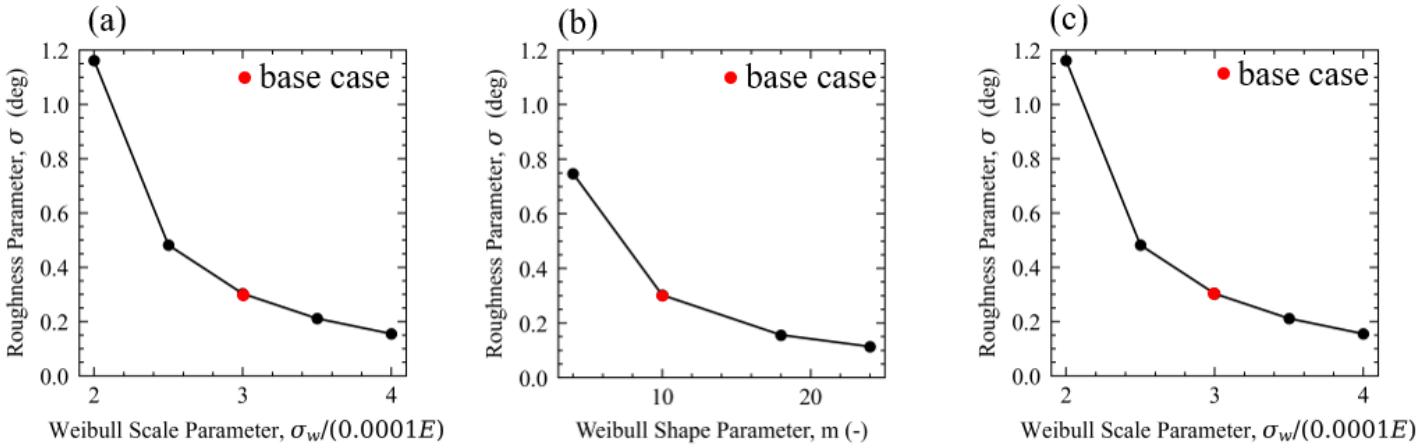


Figure 4.9: Results from the sensitivity analysis for fracture path roughness.

Voids Density

As expected, the roughness of the hypothetical MF path increased with increasing voids density. The effect of changing the void density are proportional changes to the absolute μC density, while the relative (spatial) density (comparing regions) of μCs remains constant. Therefore, changing ρ_{voids} directly changes the intensity of the interactions of the μC population with the MF. Since, the increase in roughness parameter is at a decreasing rate, the sensitivity of the MF to lateral deviations from its path decreases as ρ_{voids} increases. This is explained in terms of interaction stresses in Section 4.3.3. From a geometric perspective, as ρ_{voids} is increased the homogeneity of the PZ increases and the symmetry of the stresses being applied to the MF tip increases. The shear stresses that cause MF to deviate from its path become more frequently balanced by shear stresses originating from μCs on the opposite side of the MF axis. To obtain a non-negligible level of symmetry, a lower ρ_{voids} is required for a regular square voids grid compared to a stochastic grid. To assess the effect of symmetry, curtailing the MF path roughness, future analysis should consider stochastic void distribution.

Weibull Parameters

The Weibull parameters control the extents of the distribution and density of the μC population. Increasing m reduces the spread of the Weibull distribution and increasing σ_w shifts the

distribution of critical stresses to larger values; both of which have the effect of decreasing μC distribution size. The Weibull distributions for σ_{crit} that result from changing m and σ_w are available in Appendix I for the extreme values considered in Figure 4.9. As expected, the MF path roughness decreases as the μC population localises near the MF tip. As can be seen for smaller values of m and σ_w, the MF roughness increases significantly. Decreasing m and σ_w corresponds to larger PZs due to voids opening at lower stresses, and more μCs with $\sigma_{crit} < \sigma_\infty$. It is important to note that distributions which have a larger density of values with $\sigma_{crit} < \sigma_\infty$ have increased vulnerability to the assumption of a constant $V_{\mu C}$ as larger number of μCs open immediately upon entry into the simulation box. These μCs grow in regions of low stress for much of their lifetime and become relatively large when they pass the MF tip. This has the effect of adding and obfuscating the real behaviour. If a stress dependent μC velocity is used, the effect of low strength voids can be solved. This would also lead to more realistic effective μC geometry as more importance would be given to the orientation and magnitude of the stresses near the MF tip.

4.3.3 CRACK TIP STRESSES ALONG THE FRACTURE PATH

The effect of a population of μCs on the applied stresses at the MF tip were considered for the base case parameters. It is reiterated that applied stress refers to the sum of the stress at infinity, σ_∞, and μC interaction stresses calculated at the MF tip. The distributions for applied rectangular stress components, the major principal stress and the size of Mohr's circle $(\sigma_1 - \sigma_2)$, measured over the course of the MF propagation are produced in Figure 4.10.

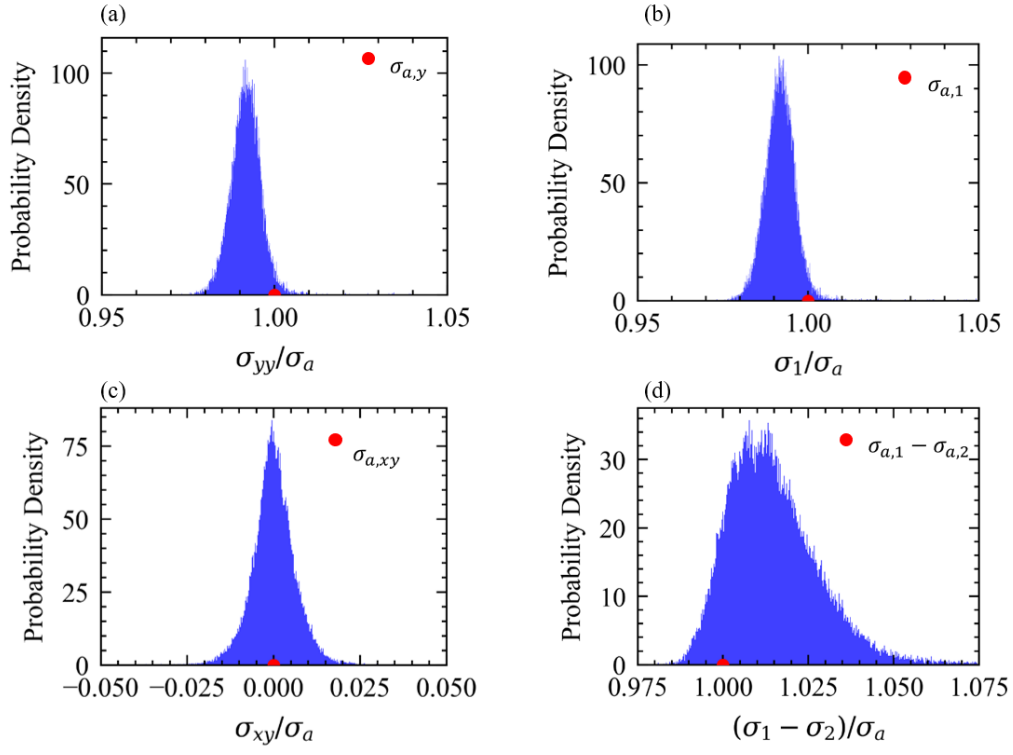


Figure 4.10: Density distribution of applied stress measured at the main fracture tip for a fracture interacting with a microcrack population generated using base case simulation parameters.

Individual Microcracks, the Microcrack Population and Shielding/Amplification

From Figure 4.10, μC population reduces σ_{yy} and σ_1 below the remote stress σ_∞. This is compared with the results for individual μCs in Figure 4.1b, whereby near tip μCs significantly increase normal stresses, and μCs with at least slightly larger displacements reduce normal stress by a smaller magnitude. The stress reduction induced by the μC population shows that the collective impact on the MF stresses of μCs outside the inner PZ region outweighs the large stress changes induced by near-tip μCs. Although, in some instances, the stress in the y-direction and the maximum normal stress increase.

While shielding and amplification are thought to be an out-of-equilibrium elastic wave scattering effect of μCs [1, 5], this research shows that shielding and amplification are present in crack interactions that rely on elasticity and superposition. Thus, understanding the effect of the PZ may not require the complex analysis for waves through a population of discontinuities to adequately reflect the stress state at the MF tip. That being the case, a PZ generation model such as the one developed in this research would be fundamental aid in generating the defect populations required to study of such phenomena. Research is required to determine the proportion of shielding is attributed to reduced shear wave speed and the amount that is caused through interactions with μCs at equilibrium.

Use of the model:

If the shear wave speed can be measured as a function of μC density, or through more detailed description of the μC population (such the PBEs for PZ description), out of equilibrium phenomena can be incorporated in models for evaluating the shielding effect.

Main Fracture Sensitivity to Individual Microcracks

As seen in Figure 4.10d, interactions with μCs generally increase the diameter of the Mohr's circle for applied stresses. The larger the Mohr's circle, the larger the shear stresses required to achieve a unit rotation of a stress element. This suggests that the sensitivity of the MF to individual μCs decreases. as the μC population size increases since changes in shear stress that the additional μC induce less rotation of the MF. This explains why the roughness parameter increases at a decreasing rate for denser μC populations in Figure 4.9a.

This can be checked by considering the distributions for $\sigma_1 - \sigma_2$ produced using the extreme values of ρ_{voids} are produced in Appendix I. They show that as ρ_{voids} increases, the distribution for the diameter of Mohr's circle shifts to larger values, indicating that the reduced sensitivity becomes even more extreme as μC population density increases. Thus, denser μC populations require

consideration of the whole population as the discreteness inherent in the μC population becomes less important.

Previous research that has identified the strong intensity of near μC interactions have neglected the importance of phenomena, concluding that their strong interactions precludes continuum representation of the PZ [52]. Although, it is likely that the observations are of an extreme case of strong localisation of μCs, possibly due to large values of m or σ_w or dynamic effects. In these situations, the ratio of the inner μC population to the outer μC population is larger and thus they dominate. This research shows that for large enough, dense enough μC populations, consideration of the entire population is important. Descriptions of such populations lend themselves to continuum representation and thus allow for opportunities of direct account of the μC populations within continuum mechanics.

4.4 LIMITATIONS

The parameters in Table 1 were inspired by the properties of a 24 MPa concrete, however the results were not designed to reflect how this material actually behaviours. With the purpose of an exploratory analysis and an investigation of regimes of behaviour, this research is designed to set the stage for further research into μC populations, ways of modelling them and the types of interactions that arise between a fracture and a μCs that precede it. To achieve this, a relatively simple approach with rather basic assumptions was undertaken. Undoubtedly, the assumptions have the potential of obfuscating the real behaviour and manufacturing apparent phenomena.

Stress Independent Crack Velocity

The velocity that the MF and μCs propagate at is assumed constant in the model. However, in reality, crack growth in perfectly elastic, homogeneous, isotropic materials, depends on the applied stress, crack size, and dynamic instability. Assuming constant crack velocity can lead do unrealistic situations whereby unreasonably levels of energy is dissipated. In the case of μCs growing in low stress levels, the energy being dissipated in their growth will far exceed the level predicted by applying energy conservation arguments to the new surface added by the crack. And, for a crack growing in regions of large stresses, less energy would be dissipated by the crack growth. In addition to the implications for the conservation of energy, the issues of crack growth being insensitive to stress level resulted in adding noise to the results. This was because μCs that come from voids with $\sigma_{crit} < \sigma_{\infty}$ grow a significant amount in regions of low stress before they reach the larger stresses imposed by the MF. These μCs, being relatively larger than their neighbours, induced noise seen in Figure 4.5c, d, and modified interaction stresses. The effect on

the latter was not quantified, although it is hypothesised that as ρ_{voids} increases the effects would be less significant.

Solutions for crack velocity are available which can improve the arbitrariness of the μC growth velocity and allow for more realistic crack growth for μCs in relatively low stress regions. An example of a solution with relatively simple implementation is Mott's derivation of crack velocity [53]. The solution requires, (a) an initial crack size, this can be directly determined from σ_{crit} and the material's fracture toughness, K_I, by rearranging Equation (3) for a, and (b) the crack terminal velocity, which can be the shear waves speed or the limiting crack velocity calculated in [54]. Another potential candidate for implementation is the solution by Bouchbinder et al [43], which requires a few more assumptions but is able to account the stress level. In addition to implementation in the μC growth law, this second solution can be used to assign a reasonable magnitude for V_{MF}.

Ignoring Energy Balance

Without account of the energy dissipated by the μC population, the system violates the conservation of energy. Despite this, Biner [11] showed that, for μCs with random nucleation, orientation and size, the energy expended in producing μC populations did not contributed significantly to fracture toughness and that elastic interaction controlled the system. Therefore, while future research can make more direct account of energy dissipation, with all else the same, the results are unlikely to be significantly different to the outcomes of this research.

4.5 VALUATION OF THE MODEL

Existing models that consider μC populations consider μCs of constant length with either deterministic or stochastic positions and orientation [5, 6, 7, 8, 9, 10, 11, 12]. The stress dependent orientations of μCs and the evolution of μC populations are not considered. In doing so, these studies skip over gaps in knowledge for the generation of μCs and attempt to investigate the effect of μC populations. The micromechanical model developed here establishes a method of direct treatment of individual μCs. This allows for more insightful analysis of the phenomena caused by μCs. For example, μC populations generated in this research have demonstrated the importance of consideration of the population effect as shown by the large shielding effect which was the opposite effect that the nearest μCs induce. In previous research, such large-scale shielding was not observed as either near tip μCs were considered or stochastic orientations precluded clear observation of the interactions between μC populations and a MF [11]. Moreover, contrary to the contention that the PZ is disorder, this research has demonstrated a level of organisation in the μC population, and more significantly, that the population is characterizable, as shown in Section 4.2.

Another advantage is in the analysis of interactions. Previously, interactions have been considered in terms of the pseudo-tractions applied to a MF or calculation of the Stress Intensity Factor, K, measured when passing a MF through an array of μCs. In this research, the simulation was run until a steady state was achieved. This revealed well defined distributions for applied stress and MF orientation. This highlights the potential for continuum description of the response of the MF to μC populations.

4.6 FUTURE WORK – BUILDING THE MODEL AND FURTHER IMPLEMENTATION

There are significant opportunities for further development and implementation of the model developed herein. These are discussed here. The results of this research have revealed some other interesting avenues for exploration, which were discussed in Sections 4.1 - 4.4.

An interesting notion from this research has been the sensitivity of the MF to μCs both as a population and via individual interactions. If the variation of the *interaction density* (interaction stress per unit area) could be determined ahead of the MF, deeper understanding the geometry of the PZ can be obtained. This could lead to a stress-based characterisation of regions in the μC population such as the PZ core, transitional zones and the region in which interactions are negligible. It was discussed in Section 2.2 that the transitional zone between the densely packed μCs near the MF tip and the μCs on the PZ periphery was difficult to define. This was due to the discreteness of μCs, and an inability to determine which μCs are important. The solution has been to determine PZ width distribution of microcracking events. However, this research has demonstrated more complex behaviour at play whereby the transitional zone can have the opposite effect on the MF compared to the μCs near the MF tip. Thus, the relative size and densify of the transitional zone and the inner PZ are also important. Thus the questions about the validity of current approaches to PZ width measurement are well justified since PZs of different size could have vastly different interactions as near-tip μCs may dominate population behaviour. Thus, defining the regions of the PZ using some *interaction density* measure would allow for a more consistent measure definition of the regions in the PZ. For example, the inner PZ and transitional zone boarder could be drawn at the point where μCs no longer induce initial repulsion. While the boarder would change location, the effect of the two regions it separates is the same.

Incorporation of Inter-μC Interactions

For simplicity in the interaction mechanism and computational efficiency, this research neglected the interactions that occur within μC populations. This research has shown the importance

of interactions in the way the orientation and magnitudes of stresses are modified. Moreover, in the same way that the MF is affected in interactions with μC populations, the μCs that comprise these populations are themselves affected. Thus, consideration must be made for inter-μC interactions. This can be achieved using the Kachanov Method described in Section 3.4.1. It is likely that accounting for inter-μC interactions will result in disruption to the order that was observed in the analysis of PZ properties as the orientation and magnitude of stresses will no-longer be fully determined from a single MF, but rather it will be the intersection of a large number of stress field solutions. As such, the validity of the results obtained when neglecting inter-μC interactions must be checked. However, given the small size of μCs it is also possible that the stress field generated by the MF will win out and ensure some level organisation prevails in the PZ.

Generalising Material Defects to Pre-existing Microcrack Populations

The defects that transmit stress that were considered in research were limited to μCs. However, in real material, there may be pre-existing μCs owing to the stress history or the effects from forming the material. These defects may be large compared to the MF-induced μC population, and their orientations may be stochastic or aligned in some direction. The potential for pre-existing cracks to modify the production of the PZ or even guide the propagation of the PZ is significant. Careful account needs to be made for non-MF-induced μCs when it comes to predicting failure modes of material. The situation where MF propagation is no longer μC controlled, and existing μCs take over needs to be determined.

Comparison of Roughness Measures with Experimental Crack Paths

The resolution of length and time scale at which for cracks propagate and the PZ forms preclude real time observations. Further, observations of the μC wake provide little information about the sequencing involved in the germination and growth of μCs. However, measurements for the roughness of a crack have been achieved rather successfully. A major advancement of this research has been in the development of a roughness characterisation for the MF based on the PZ that preceded it. The significance lies in the fact that it can be measured. Thus, future research is required into establishing physical verification of the presence and effect of the PZ on the MF by utilising roughness as the measurable, quantifiable indicator.

5 CONCLUSION

This study developed a micromechanical model to investigate the process zone microcrack population characteristics and the interactions of microcrack populations with a main fracture. The micromechanical model was developed by combining elastic solutions for crack stress fields, Weibull survival to model the resistance to microcracking, a simplified Kachanov method to develop interactions and utilisation of the maximum tensile stress criterion to direct crack growth. The model was implemented in a Python program and simulations and interactions were analysed at the resolution of individual microcracks as well as a population of microcracks. The geometric characteristics and density of microcracking for these interacting populations was also studied.

The study of microcrack population characteristics in the process zone aimed to investigate the evolution of properties of the constituent microcracks and the potential for analytically producing microcrack populations. The evolution of the microcrack length and orientation microcracks following the same pathline yielded consistent and well-defined distributions which could possibly be described by a set of population balance equations. This is a subject of future research.

An analysis of the interactions highlighted the importance of considering the overall effect microcrack populations, not simply the nearest microcracks. The nearest microcracks tend to exhibit strong repulsion and attraction forces as well as increase the maximum tensile stress, whereas microcracks at greater displacements are predominantly attractive and reduce the maximum tensile stress at the main fracture tip. The effect of the microcrack population was predominantly stress reduction, which is the opposite effect to that of near microcracks alone. The population effect becomes more important the larger the population of microcracks as microcracks increase the Mohr's circle at the main fracture tip resulting in the individual contribution of microcracks to the shear stresses required to change the main fracture direction to decrease.

A sensitivity analysis was used to study the effect of changing the parameters that control process zone generation on roughness of the main fracture path as it navigates microcrack populations. As the density of microcrack nucleation points increases, and by extension the density of microcracks increases, the roughness of the main fracture path increase. This increase in roughness was at a decreasing rate which is in agreement with decreases in the sensitivity of a main fracture to individual microcracks. Moreover, as the size of the process zone decreased due to higher voids strength, microcracks concentrated about the main fracture tip and the roughness of the fracture oath decreased. The main fracture was highly sensitive to increases in process zone size.

Finally, this research establishes fracture path roughness as a point of comparison for experimentation and simulations that model the process zone. This navigates challenge of viewing the process zone formation and evolution due the relatively small length and time scale at which cracks propagate and microcrack populations develop.

6 REFERENCES

[1] R. Blumenfeld, “Dynamics of fracture propagation in the mesoscale: Theory,” *Theoretical and Applied Fracture Mechanics,* vol. 30, no. 3, pp. 209-223, 1998.

[2] Z. P. Bažant, “Design of quasibrittle materials and structures to optimize strength and scaling at probability tail: An apercu,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences,* vol. 475, no. 2224, 2019.

[3] K. Haidar, G. Pijaudier-Cabot, J. F. Dubé and A. Loukili, “Correlation between the internal length, the fracture process zone and size effect in model materials,” *Materials and Structures/Materiaux et Constructions,* vol. 38, no. 276, pp. 201-210, 2005.

[4] A. Tarokh, R. Y. Makhnenko, A. Fakhimi and J. F. Labuz, “Scaling of the fracture process zone in rock,” 2017.

[5] G. GLADWELL, Nonlinear Crack Models for Nonmetallic Materials, Kluwer Academic Publishers, 1999.

[6] A. Chudnovsky, A. Dolgopolsky and M. Kachanov, “Elastic interaction of a crack with a microcrack array-I. Formulation of the problem and general form of the solution,” 1987.

[7] M. Kachanov and E. Montagut, “MARK KACHANOV and ERIC MONTAGUT Department of M~hani~i Engineering, Tufts University, Medford, MA 02355,,” 1986.

[8] M. Kachanov and A. Chudnovsky, “hr. A&. Engng Sci. Vol. 21, No. 8, pp. Printed,” 1983.

[9] Y. P. Li, L. G. Tham, Y. H. Wang and Y. Tsui, “A modified Kachanov method for analysis of solids with multiple cracks,” 2003.

[10] E. L. E. Montagut and J. P. Laures, “Mechanics of crack-microcrack interactions,” 1990.

[11] S. B. Biner, “A numerical analysis of crack propagation in microcracking ceramic and ceramic composites,” *Journal of Non-Crystalline Solids,* vol. 177, no. C, pp. 36-45, 1994.

[12] V. P. Tamuzs, V. E. Petrova, A. S. Vavakin, B. Budiansky, A. G. Evans, Y. Fu, M. Ortiz, Y. Murakami and P. Mekhanika, “On macrocrack–microdefect interaction,” 2002.

[13] A. du Plessis and W. P. Boshoff, “A review of X-ray computed tomography of concrete and asphalt construction materials,” *Construction and Building Materials,* vol. 199, no. April, pp. 637-651, 2019.

[14] A. T. Zehnder, Fracture Mechanics, vol. 6, 2012.

[15] Z. Bazant, “Size effect on structural strength: a review,” *Archive of Applied Mechanics,* vol. 69, pp. 703-725, 1999.

[16] G. C. Sih, Mechanics of Fracture Initiation and Propagation, Kluwer Academic Publishers, 1991.

[17] K. A. Macdonald, Fracture and Fatigue of Welded Joints and Structures, Woodhead Publishing, 2011.

[18] F. Erdogan, “Fracture mechanics,” *International Journal of Solids and Structures,* vol. 37, pp. 171-183, 2000.

[19] F. Erdogan, Crack Propagation Theories, 1967.

[20] D. W. Cof, K. Li and J. Li, “Utilization of Modified Linear Elastic Fracture Mechanics To Characterize the Fracture Resistance of Paper,” *Advances in Pulp nd Paper Research, Cambridge 2013,* no. September 2013, pp. 637-672, 2013.

[21] A. Livne, E. Bouchbinder and J. Fineberg, “Breakdown of linear elastic fracture mechanics near the tip of a rapid crack,” *Physical Review Letters,* vol. 101, no. 26, 2008.

[22] L. Cedolin, S. Dei Poli and I. Iori, “Tensile Behavior of Concrete,” *Journal of Engineering Mechanics,* vol. 113, no. 3, 1987.

[23] A. Castro-Montero, S. P. Shah and R. A. Miller, “Strain Field Measurement in Fracture Process Zone,” *Journal of Engineering Mechanics,* vol. 11, no. 116, 1990.

[24] L. J. Struble, P. E. Stuntzman and E. R. Fuller, “Microstructural Aspects of the Fracture of Hardened Cement Paste,” *American Ceramic Society,* vol. 72, no. 12, pp. 2295-2299, 1989.

[25] A. Maji and S. P. Shah, “Process Zone and Acoustic-Emission Measurements in Concrete,” *Experimental Mechanics,* pp. 27-33, 1986.

[26] D. Grégoire, L. Verdon, V. Lefort, P. Grassl, J. Saliba, J.-P. Regoin, A. Loukili and G. Pijaudier-Cabot1, “Mesoscale analysis of failure in quasi-brittle materials: comparison

between lattice model and acoustic emission data," *International Journal for Numerical and Analytical Methods in Geomechanics,* vol. 30, no. 39, pp. 1639-1664, 2015.

[27] W. B. Bradley and A. S. Kobayashi, "Fracture dynamics-a photoelastic investigation," *Engineering Fracture Mechanics,* vol. 3, no. 3, 1971.

[28] G. Pijaudier-Cabot and D. Grégoire, "A review of non local continuum damage: Modelling of failure?," *Networks and Heterogeneous Media,* vol. 9, no. 4, pp. 575-597, 2014.

[29] K. K. Skrzypek and A. W. Ganczarski, Anisotropic Behaviour of Damaged Materials, vol. 9, F. Pfeiffer and P. Wriggers, Eds., Die Deutsche Bibliothek, 2003.

[30] J. P. Laures and M. Kachanov, "Three-dimensional interactions of a crack front with arrays of penny-shaped microcracks," *International Journal of Fracture,* vol. 48, no. 4, pp. 255-279, 1991.

[31] Z. Bažant and J. Planas, Fracture and Size Effect in Concrete and Other Quasibrittle Materials, W. F. Chen, Ed., 1998.

[32] G. P.-C. Zdenek P. Bazant, "MEASUREMENT OF CHARACTERISTIC LENGTH OF NONLOCAL CONTINUUM," *Journal of Engineering Mechanics,* vol. 115, no. 4, pp. 775-767, 1989.

[33] J. Mazars and G. Pijaudier-Cabot, "From damage to fracture mechanics and conversely: A combined approach," *International Journal of Solids and Structures,* vol. 33, no. 20-22, pp. 3327-3342, 1996.

[34] M. Jirásek, "Nonlocal Theories in Continuum Mechanics," *Acta Polytechnica,* vol. 44, no. 5-6, 2004.

[35] M. Jirásek, "Nonlocal damage mechanics," *Revue européenne de génie civil,* vol. 11, no. 7-8, pp. 993-1021, 2007.

[36] Z. P. Bažant, Mechanics of distributed cracking, vol. 39, 1986, pp. 675-705.

[37] D. Krajcinovic, M. Basista and D. Sumarac, "Micromechanically Inspired Phenomenological Damage Model," *Journal of Applied Mechanics,* vol. 58, no. 2, pp. 305-310, 1 6 1991.

[38] J. Weertman, Dislocation Based Fracture Mechanics, Singapore: World Scientific Publishing, 1998.

[39] E. H. Yoffe, “LXXV. The moving griffith crack,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* vol. 42, no. 330, pp. 739-750, 1951.

[40] A. T. Zehnder, “Lecture Notes on Fracture Mechanics,” Ithaca, NY, 2007.

[41] L. Graham-Brady, “Statistical characterization of meso-scale uniaxial compressive strength in brittle materials with randomly occurring flaws,” *International Journal of Solids and Structures,* vol. 47, no. 18-19, pp. 2398-2413, 2010.

[42] L. Afferrante, M. Ciavarella and E. Valenza, “Is Weibull's modulus really a material constant? Example case with interacting collinear cracks,” *International Journal of Solids and Structures,* vol. 43, no. 17, pp. 5147-5157, 2006.

[43] E. Bouchbinder, J. Fineberg and M. Marder, “Dynamics of simple cracks,” *Annual Review of Condensed Matter Physics,* vol. 1, pp. 371-395, 2010.

[44] E. N. Dulaney and W. F. Brace, “Velocity Behavior of a Growing Crack,” *Journal of Applied Physics,* vol. 31, p. 2233, 1960.

[45] M. Kachanov, “A Simple Technique of Stress Analysis in Elastic Solids with Many Cracks,” *International Journal of Fracture,* vol. 28, 1985.

[46] M. Kachanov, “Elastic Solits with Many Cracks: A Simple Method of Analysis,” *International Journal of Solids and Structures,* vol. 23, no. I, pp. 23-43, 1987.

[47] M. Kachanov, “On the problems of crack interactions and crack coalescence,” *International Journal of Fracture,* vol. 120, no. 3, pp. 537-543, 2003.

[48] Y. S. Jenq and S. P. Shah, “A Fracture toughness criterion for concrete,” *Engineering Fracture Mechanics,* vol. 21, no. 5, pp. 1055-1069, 1985.

[49] S. A. Hamoush and H. Abdel-Fattah, “The fracture toughness of concrete,” *Engineering Fracture Mechanics,* vol. 53, no. 3, pp. 425-432, 1996.

[50] E. Alyama, “A prediction formula for fracture toughness of concrete,” no. OCTOBER, pp. 213-222, 2005.

[51] D. Benarbia and M. Benguediab, “Determination of stress intensity factor in concrete material under Brazilian disc and three-point bending tests using finite element method,” *Periodica Polytechnica Mechanical Engineering,* vol. 59, no. 4, pp. 199-203, 2015.

[52] K. B. Broberg, Cracks and Fracture, Dublin, 1999.

[53] A. Rabinovitch, “On the Mott derivation of crack velocity,” *Philosophical Magazine Letters,* vol. 70, no. 4, pp. 231-233, 1994.

[54] C. Guerr, J. Scheibert, D. Bonamy and D. Dalmas, “Understanding fast macroscale fracture from microcrack post mortem patterns,” *Proceedings of the National Academy of Sciences of the United States of America,* vol. 109, no. 2, pp. 390-394, 2012.

[55] X. Y. Long, C. Jiang, X. Han, W. Gao, X. G. Wang and M. Z. Hou, “Probabilistic crack trajectory analysis by a dimension reduction method,” *Fatigue and Fracture of Engineering Materials and Structures,* vol. 40, no. 1, pp. 12-26, 2017.

[56] E. Katzav and M. Adda-Bedia, “Stability and roughness of tensile cracks in disordered materials,” *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics,* vol. 88, no. 5, 2013.

[57] D. Sutula, P. Kerfriden, T. van Dam and S. P. Bordas, “Minimum energy multiple crack propagation. Part III: XFEM computer implementation and applications,” *Engineering Fracture Mechanics,* vol. 191, pp. 257-276, 2018.

[58] D. Roylance, “Introduction To Fracture Mechanics.,” 2001.

[59] R. H. J. Peerlings, R. De Borst, W. A. M. Brekelmans and J. H. P. De Vree, “GRADIENT ENHANCED DAMAGE FOR QUASI-BRITTLE MATERIALS,” vol. 39, pp. 3391-3403, 1996.

[60] B. Patzák and M. Jirásek, “Process zone resolution by extended finite elements,” *Engineering Fracture Mechanics,* vol. 70, no. 7-8, pp. 957-977, 2003.

[61] X. Gao, G. Koval and C. Chazallon, “A size and boundary effects model for quasi-brittle fracture,” *Materials,* vol. 9, no. 12, pp. 1-20, 2016.

[62] Z. Bažant and J.-L. Le, Probabilistic Mechanics of Quasibrittle Structures: Strength, Lifetime, and Size Effect, 2017.

[63] Y. H. Tai, M. W. Brown and J. R. Yates, “A new solution for 3D crack extension based on linear elastic stress fields,” *Engineering Fracture Mechanics,* vol. 78, no. 8, pp. 1602-1613, 2011.

[64] T. Davis, D. Healy, A. Bubeck and R. Walker, “Stress concentrations around voids in three dimensions: The roots of failure,” *Journal of Structural Geology*, vol. 102, pp. 193-207, 2017.

[65] E. N. Dulaney, “The velocity behavior of a growing crack,” 2010.

7 APPENDICES

Appendix A STRESS FIELD FORMULAE

In stress field formulae, σ_a refers to the stress applied at infinity. This is σ_∞ in the body of the thesis.

THE STATIC GRIFFITH CRACK

Far-Field

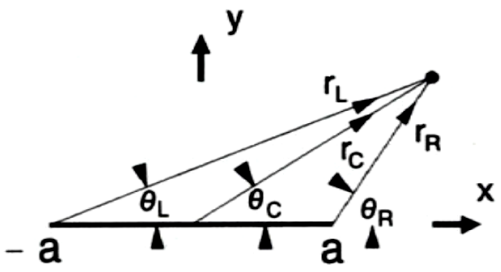


Figure 1.8. Coordinates for general stress field of Griffith-Inglis and Zener-Stroh-Koehler cracks.

1.2.1. Mode I Crack Stress Field

In the coordinate system given in Fig. 1.8 (with the subscript L referring to the left hand crack tip, R to the right hand crack tip and C to the center of the crack) the general stress field of a Griffith-Inglis mode I crack in a linear elastic solid is

$$\sigma_{xx} = -\sigma_A + \sigma_A[r_C/2r^*]\{2\cos(\theta^*-\theta_C) + 2\sin\theta_C\sin\theta^* - \sin\theta_R\sin(\theta^*+\theta_R-\theta_C) - \sin\theta_L\sin(\theta^*+\theta_L-\theta_C)\},$$

$$\sigma_{yy} = \sigma_A[r_C/2r^*]\{2\cos(\theta^*-\theta_C) - 2\sin\theta_C\sin\theta^* + \sin\theta_R\sin(\theta^*+\theta_R-\theta_C) + \sin\theta_L\sin(\theta^*+\theta_L-\theta_C)\},$$

$$\sigma_{xy} = \sigma_A[r_C/2r^*]\{\sin\theta_R\cos(\theta^*+\theta_R-\theta_C) + \sin\theta_L\cos(\theta^*+\theta_L-\theta_C) - 2\sin\theta_C\cos\theta^*\},$$

$$\tfrac{1}{2}(\sigma_{xx}-\sigma_{yy}) = -\tfrac{1}{2}\sigma_A + \sigma_A[r_C/2r^*]\{2\sin\theta_C\sin\theta^* - \sin\theta_R\sin(\theta^*+\theta_R-\theta_C) - \sin\theta_L\sin(\theta^*+\theta_L-\theta_C)\},$$

$$\tfrac{1}{2}(\sigma_{xx}+\sigma_{yy}) = -\tfrac{1}{2}\sigma_A + \sigma_A[r_C/r^*]\cos(\theta^*-\theta_C), \qquad \sigma_{z\omega} = -\sigma_A[r_C/r^*]\sin(\theta^*-\theta_C). \tag{1.10}$$

$$r^* = \sqrt{(r_L r_R)}, \qquad \theta^* = \tfrac{1}{2}(\theta_L+\theta_R),$$
$$r_C = \sqrt{[x^2+y^2]}, \qquad \theta_C = \tan^{-1}[y/x],$$
$$r_L = \sqrt{[(x+a)^2+y^2]}, \qquad \theta_L = \tan^{-1}[y/(x+a)],$$
$$r_R = \sqrt{[(x-a)^2+y^2]}, \qquad \theta_R = \tan^{-1}[y/(x-a)],$$

and $-\pi \le \{\theta_C, \theta_L, \theta_R, \theta^*\} \le \pi$. All other stress components are equal to zero except the stress component σ_{zz}. The stress component σ_{zz} is equal to $\sigma_{zz} = \nu(\sigma_{xx}+\sigma_{yy})$. The stress given by eq. (1.10) is the sum of the applied stress and the stress arising from all of the dislocations on the crack plane. The applied stress σ_A is a tensile stress σ_{yy}. At large distances from the crack: $\sigma_{yy} \to \sigma_A$, $\sigma_{zz} \to \nu\sigma_A$ and $\sigma_{xx} \to \sigma_{xy} \to \sigma_{z\omega} \to 0$ as $r \to \infty$.

1.2.2. Mode II Crack Stress Field

For the Griffith-Inglis mode II crack the stress field is

$$\sigma_{xx} = -\sigma_{yy} - 2\sigma_A[r_C/r^*]\sin(\theta^*-\theta_C),$$

$$\sigma_{yy} = \sigma_A[r_C/2r^*]\{\sin\theta_R\cos(\theta^*+\theta_R-\theta_C) + \sin\theta_L\cos(\theta^*+\theta_L-\theta_C) - 2\sin\theta_C\cos\theta^*\},$$

$$\sigma_{xy} = \sigma_A[r_C/2r^*]\{2\cos(\theta^*-\theta_C) + 2\sin\theta_C\sin\theta^* - \sin\theta_R\sin(\theta^*+\theta_R-\theta_C) - \sin\theta_L\sin(\theta^*+\theta_L-\theta_C)\},$$

$$\tfrac{1}{2}(\sigma_{xx}-\sigma_{yy}) = -\sigma_A[r_C/r^*]\sin(\theta^*-\theta_C) - \sigma_A[r_C/2r^*]\{\sin\theta_R\cos(\theta^*+\theta_R-\theta_C) + \sin\theta_L\cos(\theta^*+\theta_L-\theta_C) - 2\sin\theta_C\cos\theta^*\},$$

$$\tfrac{1}{2}(\sigma_{xx}+\sigma_{yy}) = -\sigma_A[r_C/r^*]\sin(\theta^*-\theta_C), \quad \sigma_{z\omega} = -\sigma_A[r_C/r^*]\cos(\theta^*-\theta_C). \tag{1.11}$$

All other stress components are equal to zero except the stress component $\sigma_{zz} = \nu(\sigma_{xx}+\sigma_{yy})$. At large distances from the crack: $\sigma_{xy} \to \sigma_A$ and $\sigma_{xx} \to \sigma_{yy} \to \sigma_{zz} \to 0$ as $r \to \infty$. Figures 1.12 are contour plots of constant maximum shear stress magnitude for the mode II Griffith-Inglis crack calculated with the above equations. The plots are presented for different distance scales.

1.2.4. Mode I Near Tip Stress Field

The general stress fields for the modes I, II and III cracks given by eqs (1.10), (1.11) and (1.12) may be difficult to visualize because the various terms are expressed with the combination of four systems, r^*, θ^*, r_L, θ_L, r_R, θ_R, r_C, θ_C. In the near tip region this problem is lessened. The terms in the equations for the stress components are expressed in only one coordinate system.

Close to the right hand crack tip, where $r_R \ll a$, the mode I crack stress field reduces to

$$\sigma_{xx} = \{K_I/(2\pi r)^{1/2}\}\cos\tfrac{1}{2}\theta\,[1 - \sin\tfrac{1}{2}\theta\sin\tfrac{3}{2}\theta],$$
$$\sigma_{yy} = \{K_I/(2\pi r)^{1/2}\}\cos\tfrac{1}{2}\theta\,[1 + \sin\tfrac{1}{2}\theta\sin\tfrac{3}{2}\theta],$$
$$\sigma_{xy} = \{K_I/(2\pi r)^{1/2}\}\sin\tfrac{1}{2}\theta\cos\tfrac{1}{2}\theta\cos\tfrac{3}{2}\theta,$$
$$\tfrac{1}{2}(\sigma_{xx}-\sigma_{yy}) = -\{K_I/(2\pi r)^{1/2}\}\cos\tfrac{1}{2}\theta\sin\tfrac{1}{2}\theta\sin\tfrac{3}{2}\theta,$$
$$\tfrac{1}{2}(\sigma_{xx}+\sigma_{yy}) = \{K_I/(2\pi r)^{1/2}\}\cos\tfrac{1}{2}\theta,$$
$$\sigma_{z\omega} = -\{K_I/(2\pi r)^{1/2}\}\sin\tfrac{1}{2}\theta. \qquad (1.13)$$

The subscript R has been dropped in these expressions for the near tip stress components. It is to be understood that r and θ are measured from the right hand crack tip. The azimuthal angle θ is restricted to the angular range $-\pi \le \theta \le \pi$. The term K_I is $K_I = \sigma_{AI}(\pi a)^{1/2}$ where σ_{AI} is the applied tensile stress σ_{yy} in the y-direction.

In a cylindrical coordinate system (see Appendix D for the transformation equations) this mode I near tip stress field is expressed as

1.2.5. Mode II Near Tip Stress Field

The near tip mode II stress field is

$$\sigma_{xx} = -\{K_{II}/(2\pi r)^{1/2}\}\sin\tfrac{1}{2}\theta\,[2 + \cos\tfrac{1}{2}\theta\cos\tfrac{3}{2}\theta],$$
$$\sigma_{yy} = \{K_{II}/(2\pi r)^{1/2}\}\sin\tfrac{1}{2}\theta\cos\tfrac{1}{2}\theta\cos\tfrac{3}{2}\theta,$$
$$\sigma_{xy} = \{K_{II}/(2\pi r)^{1/2}\}\cos\tfrac{1}{2}\theta\,[1 - \sin\tfrac{1}{2}\theta\sin\tfrac{3}{2}\theta],$$
$$\tfrac{1}{2}(\sigma_{xx}-\sigma_{yy}) = -\{K_{II}/(2\pi r)^{1/2}\}\sin\tfrac{1}{2}\theta\,[1 + \cos\tfrac{1}{2}\theta\cos\tfrac{3}{2}\theta],$$
$$\tfrac{1}{2}(\sigma_{xx}+\sigma_{yy}) = -\{K_{II}/(2\pi r)^{1/2}\}\sin\tfrac{1}{2}\theta,$$
$$\sigma_{z\omega} = -\{K_{II}/(2\pi r)^{1/2}\}\cos\tfrac{1}{2}\theta. \qquad (1.14)$$

Far-Field

$$\beta_S = \sqrt{1 - V^2/C_S^2}, \quad \beta_L = \sqrt{1 - V^2/C_L^2},$$
$$\beta_{2S} = \sqrt{1 - V^2/2C_S^2}, \; \beta_{2L} = \sqrt{1 - V^2/2C_L^2},$$

where $C_S = \sqrt{\mu/\rho_S} \equiv \sqrt{G/\rho_S}$ is the shear wave velocity, $C_L = \sqrt{(\lambda + 2\mu)/\rho_S} = C_S\sqrt{2(1-\nu)/(1-2\nu)}$ is the longitudinal wave velocity and ρ_S is the density of the solid. (The Lamé constants λ and μ are given in Appendix D.)

The stress field results are given below. In these stress field equations the variables ρ, R etc. are defined by the relationships

$$\rho = \sqrt{x^2 + \beta_S^2 y^2}, \qquad \rho_1 = \sqrt{(x+a)^2 + \beta_S^2 y^2},$$
$$\rho_2 = \sqrt{(x-a)^2 + \beta_S^2 y^2}, \qquad \rho^* = \sqrt{\rho_1 \rho_2},$$

$$R = \sqrt{x^2 + \beta_L^2 y^2}, \qquad R_1 = \sqrt{(x+a)^2 + \beta_L^2 y^2},$$
$$R_2 = \sqrt{(x-a)^2 + \beta_L^2 y^2}, \qquad R^* = \sqrt{R_1 R_2},$$

$$\psi = \arctan\{\beta_S\, y/x\}, \qquad \psi_1 = \arctan\{\beta_S\, y/(x+a)\},$$
$$\psi_2 = \arctan\{\beta_S\, y/(x-a)\}, \qquad \psi^* = \tfrac{1}{2}(\psi_1 + \psi_2),$$

$$\phi = \arctan\{\beta_L\, y/x\}, \qquad \phi_1 = \arctan\{\beta_L\, y/(x+a)\},$$
$$\phi_2 = \arctan\{\beta_L\, y/(x-a)\}, \qquad \phi^* = \tfrac{1}{2}(\phi_1 + \phi_2). \qquad (9.10)$$

9.2.1. Mode I Crack

The stress field of the mode I Yoffe crack is

$$\sigma_{xy} = \frac{\sigma_A \beta_L \beta_{2S}^2}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{R}{R^*} \sin(\phi^* - \phi) - \frac{\rho}{\rho^*} \sin(\psi^* - \psi) \right\},$$

$$\sigma_{yy} = \sigma_A + \frac{\sigma_A}{\beta_S \beta_L - \beta_{2S}^4} \left\{ -\beta_{2S}^4 \frac{R}{R^*} \cos(\phi^* - \phi) + \beta_S \beta_L \frac{\rho}{\rho^*} \cos(\psi^* - \psi) + \beta_{2S}^4 - \beta_S \beta_L \right\},$$

$$\sigma_{xx} = \frac{\sigma_A}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \beta_{2S}^2 (2\beta_{2L}^2 - \beta_{2S}^2) \frac{R}{R^*} \cos(\phi^* - \phi) - \beta_S \beta_L \frac{\rho}{\rho^*} \cos(\psi^* - \psi) - \beta_{2S}^2 (2\beta_{2L}^2 - \beta_{2S}^2) + \beta_S \beta_L \right\},$$

$$\sigma_{z\omega} = -\frac{\sigma_A \beta_L}{\beta_S \beta_L - \beta_{2S}^4} \frac{V^2}{4\alpha C_S^2} \left\{ \frac{\rho}{\rho^*} \sin(\psi^* - \psi) \right\}. \qquad (9.11)$$

For the mode I crack, as well as for the mode II crack below, all other stress components are equal to zero except the stress component σ_{zz} which is equal to $\sigma_{zz} = \nu(\sigma_{xx} + \sigma_{yy})$.

9.2.2. Mode II Crack

For the Yoffe mode II crack the stress field is

$$\sigma_{xy} = \sigma_A + \frac{\sigma_A}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \beta_S \beta_L \frac{R}{R^*} \cos(\phi^* - \phi) - \beta_{2S}^4 \frac{\rho}{\rho^*} \cos(\psi^* - \psi) - \beta_S \beta_L + \beta_{2S}^2 \right\},$$

$$\sigma_{yy} = \frac{\sigma_A \beta_S \beta_{2S}^2}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \beta_L \frac{R}{R^*} \sin(\phi^* - \phi) - \beta_S \frac{\rho}{\rho^*} \sin(\psi^* - \psi) \right\},$$

$$\sigma_{xx} = -\frac{\sigma_A \beta_S}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \beta_L (2\beta_{2L}^2 - \beta_{2S}^2) \frac{R}{R^*} \sin(\phi^* - \phi) - \beta_S \beta_{2S}^2 \frac{\rho}{\rho^*} \sin(\psi^* - \psi) \right\},$$

$$\sigma_{z\omega} = -\frac{\sigma_A \beta_{2S}^2}{\beta_S \beta_L - \beta_{2S}^4} \frac{V^2}{4\alpha C_S^2} \left\{ \frac{\rho}{\rho^*} \cos(\psi^* - \psi) - 1 \right\}. \qquad (9.12)$$

Near-Field

9.2.4. Mode I Near Tip Stress Field

$$\sigma_{xy} = \frac{K_I}{\sqrt{2\pi r_R}} \frac{\beta_L \beta_{2S}^2}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{\sin\frac{1}{2}\phi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_L^2 \sin^2\theta_R}}} - \frac{\sin\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\},$$

$$\sigma_{yy} = -\frac{K_I}{\sqrt{2\pi r_R}} \frac{1}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{\beta_{2S}^4 \cos\frac{1}{2}\phi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_L^2 \sin^2\theta_R}}} - \frac{\beta_S \beta_L \cos\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\},$$

$$\sigma_{xx} = \frac{K_I}{\sqrt{2\pi r_R}} \frac{1}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{\beta_{2S}^2 (2\beta_{2L}^2 - \beta_{2S}^2) \cos\frac{1}{2}\phi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_L^2 \sin^2\theta_R}}} - \frac{\beta_S \beta_L \cos\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\},$$

$$\sigma_{z\omega} = -\frac{K_I}{\sqrt{2\pi r_R}} \frac{\beta_L}{\beta_S \beta_L - \beta_{2S}^4} \frac{V^2}{4\alpha C_S^2} \left\{ \frac{\sin\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\}. \qquad (9.14)$$

9.2.5. Mode II Near Tip Stress Field

$$\sigma_{xy} = \frac{K_{II}}{\sqrt{2\pi r_R}} \frac{1}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{\beta_S \beta_L \cos\frac{1}{2}\phi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_L^2 \sin^2\theta_R}}} - \frac{\beta_{2S}^4 \cos\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\},$$

$$\sigma_{yy} = \frac{K_{II}}{\sqrt{2\pi r_R}} \frac{\beta_S \beta_{2S}^2}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{\beta_L \sin\frac{1}{2}\phi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_L^2 \sin^2\theta_R}}} - \frac{\beta_S \sin\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\},$$

$$\sigma_{xx} = -\frac{K_{II}}{\sqrt{2\pi r_R}} \frac{\beta_S}{\beta_S \beta_L - \beta_{2S}^4} \left\{ \frac{\beta_L (2\beta_{2L}^2 - \beta_{2S}^2) \sin\frac{1}{2}\phi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_L^2 \sin^2\theta_R}}} - \frac{\beta_S \beta_{2S}^2 \sin\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\},$$

$$\sigma_{z\omega} = -\frac{K_{II}}{\sqrt{2\pi r_R}} \frac{\beta_{2S}^2}{\beta_S \beta_L - \beta_{2S}^4} \frac{V^2}{4\alpha C_S^2} \left\{ \frac{\cos\frac{1}{2}\psi_2}{\sqrt{\sqrt{\cos^2\theta_R + \beta_S^2 \sin^2\theta_R}}} \right\}. \qquad (9.15)$$

Appendix B STRESS FIELD PLOTS

THE STATIC GRIFFITH CRACK

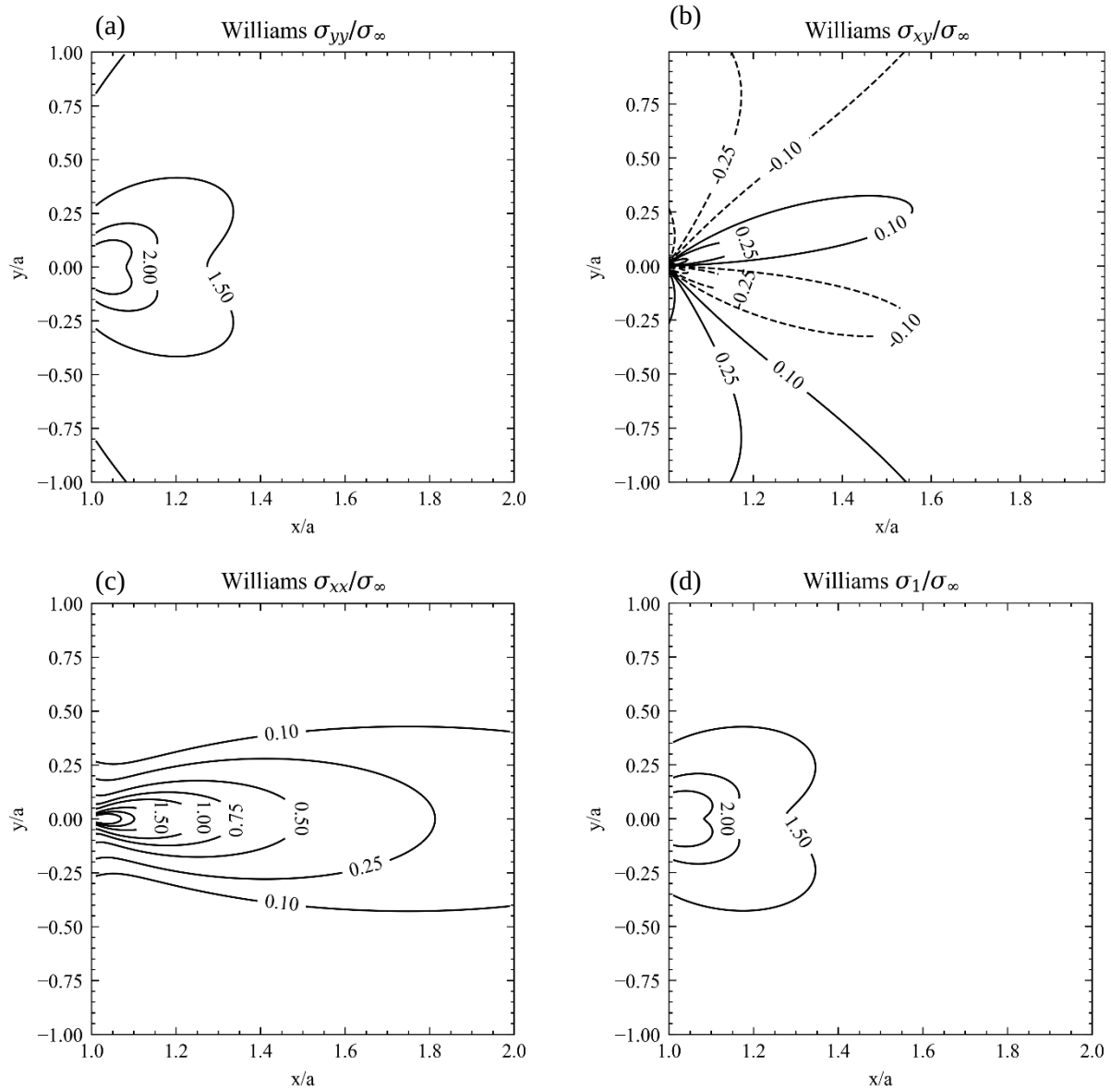


Figure 7.1: Contours for the static Griffith crack stress field solution.

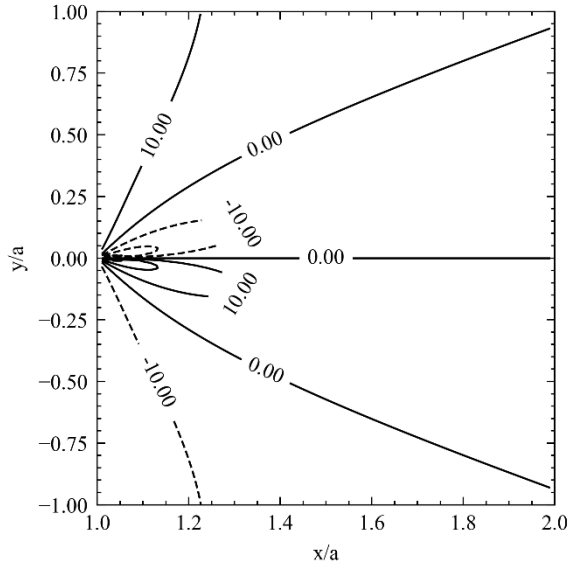


Figure 7.2: Angle of Inclination of the σ_2 direction to the positive x-axis for the Williams stress field solution. This corresponds to the direction in which microcracks would grow according to the maximum tensile stress criterion.

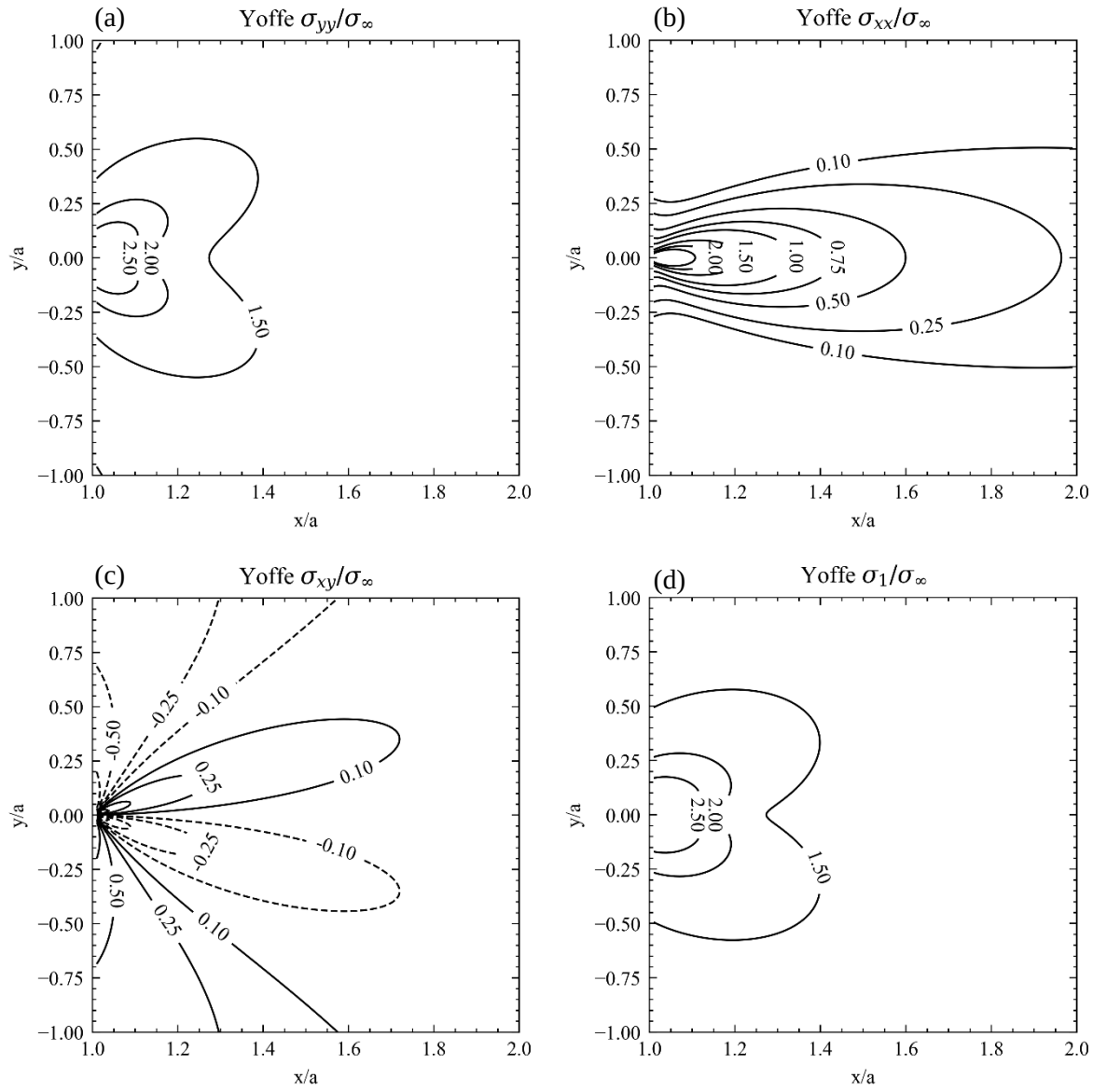


Figure 7.3: Contours for the dynamic Griffith crack stress field solution.

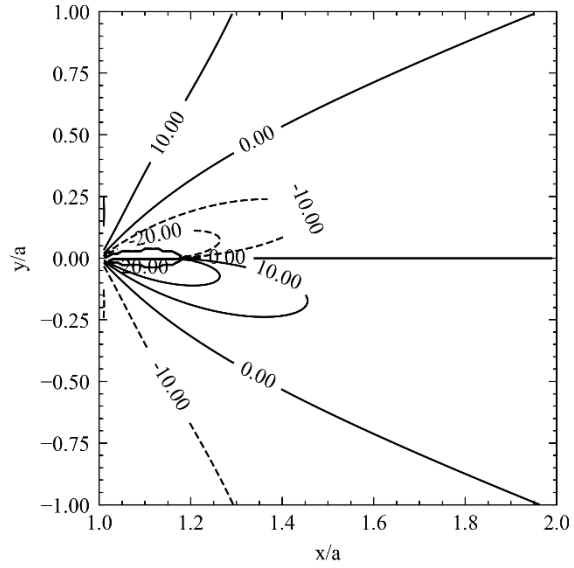


Figure 7.4: Angle of Inclination of σ_2 direction to the positive x-axis for the Yoffe stress field solution. This corresponds to the direction in which microcracks would grow according to the maximum tensile stress criterion.

WEIGHTS FUNCTION

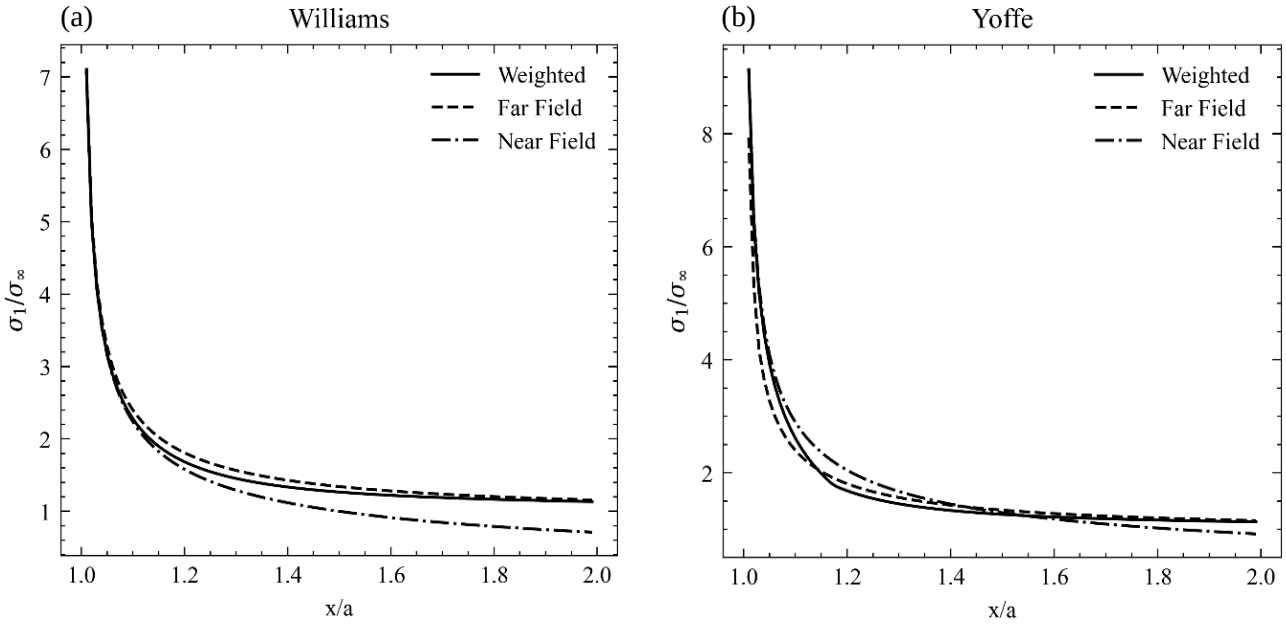


Figure 7.5: Merging near- and far- stress field solutions along the crack major axis for (a) the static crack (Williams), and (b) the dynamic crack (Yoffe).

Appendix C DERIVATION FOR SIMPLIFIED KACHANOV METHOD FOR INTERACTIONS

The Kachanov method for crack interactions uses the principal of superposition combined with a self-consistent method to calculate the average traction applied to a crack which is affected by the presence of nearby cracks. Mathematically, by the Kachanov method, the average traction applied to some crack, MF, that experiences a normal stress applied at infinity, σ_∞, and stresses from field of m other cracks is given by,

$$\sigma_{MF} = \sigma_\infty + \sum_{i=1}^{m} \Lambda_{i,MF}^{nn} . \langle \sigma_i \rangle + \Lambda_{i,MF}^{\tau n} . \langle \tau_i \rangle \tag{11}$$

where,

σ_∞ = traction applied far from main fracture

σ_{MF} = average traction applied to crack MF

$\langle \sigma_i \rangle$ = average normal traction applied to crack i

$\langle \tau_i \rangle$ = average shear traction applied to crack i

$\Lambda_{i,MF}^{nn}$ = transmission factor for normal stresses applied to crack i

$\Lambda_{i,MF}^{\tau n}$ = transmission factor for shear stresses applied to crack i

The normal/shear transmission factor for each crack, i, is the average normal/shear traction applied to crack MF, when each crack, i, us loaded by a unit stress.

There is an Equation (11) for each crack in the system. If it is assumed crack interactions only occur between the MF and some field of n μCs, then the average tractions on each crack (σ_i, τ_i) come directly from the stress field solution for crack MF. Given this assumption and since the stress field solutions in Section Appendix A are all directly proportional to the applied stress (that is, for each stress component, $\sigma = \sigma_a \cdot f(loading, geometry, kinematic\ parmaeters)$), then the stresses applied to the crack MF can be re-written as

$$\sigma_x = \sigma_{\infty,x} + \sum_{i=1}^{m} \left(\sigma_{x,i}^{I} + \sigma_{x,i}^{II}\right)$$

$$\sigma_y = \sigma_{\infty,y} + \sum_{i=1}^{m} \left(\sigma_{y,i}^{I} + \sigma_{y,i}^{II}\right)$$

$$\sigma_y = \sigma_{\infty,y} + \sum_{i=1}^{m} \left(\sigma_{y,i}^{I} + \sigma_{y,i}^{II}\right) \tag{12}$$

Or, in condensed form, the interaction traction, $t^{(k)}(x)$

$$\boldsymbol{t}^{(MF)} = \boldsymbol{t}_0^{(MF)} + \Delta\boldsymbol{t}^{(k)} = \boldsymbol{t}_0^{(MF)} + \sum_{i=1}^{m} \boldsymbol{t}^{(i)} \tag{13}$$

where, $\boldsymbol{t} = \left(\sigma_x, \sigma_y, \sigma_{xy}\right) = (\sigma_t, \sigma_n, \sigma_\tau)$

Note that σ_x is not a traction stress. Only σ_y and σ_{xy} are traction stresses.

Note that it is assumed that the coordinate system is appropriate to the crack, MF.

Appendix D CALCULATIONS FOR MAIN FRACTURE DIRECTION AND PATH

FRACTURE DIRECTION

The calculation of θ is as follows:

a) Calculate the rotation to the principal plane

$$\alpha = 0.5 \operatorname{atan}\left(\frac{2\sigma_{xy}}{\sigma_{xx} - \sigma_{yy}}\right)$$

This gives the smallest rotation of the stress element required to obtain the principal orientation.

b) Calculate θ

$$\theta = \alpha + \pi/2 \cdot \frac{|\sigma_{xy}|}{\sigma_{xy}} \cdot \langle \sigma_x - \sigma_y \rangle^0$$

where, $\langle \sigma_x - \sigma_y \rangle^0$ controls when $\pm\pi/2$ is applied, and $\frac{|\sigma_{xy}|}{\sigma_{xy}}$ is used to determine which of $\pm\pi/2$ is applied to ensure θ is acute.

MAIN FRACTURE PATH

The calculation of the vertical displacement of the main fracture is calculated as

$$y_i = \sum_{j \le i} \Delta L \cdot \sin \theta_j \tag{14}$$

where, θ_j is the instantaneous MF orientation of all previous interactions, and ΔL is the displacement of the MF at each iteration.

Appendix E SIMULATION OVERVIEW AND DESIGN

OVERVIEW

There are three scripts that that are involved in the simulation. A description of the function of each script is presented here.

Main_Script.py

The main body of the simulation and all post-processing code is in the engine.

Micro_VoidCrack.py

The properties of each void, geometry of each μC and the functions that control the growth of each μC and the interaction mechanism are contained in the Micro_VoidCrack Class.

stresses.py

The Yoffe and Williams analytical solutions for mode I and Mode II cracking and the method for merging stress field solutions are defined.

DESIGN AND PROCEDURES

General Simulation Structure: (Represent this in a flow chart)

1. Define/import input parameters
2. Set simulation settings (time-step, simulation length, frame of reference, approach, stress measuring method for interactions)
3. Distribute voids
4. Generate critical opening stresses
5. Enter Simulation
    a. Rotate stress element so that it is inline with instantaneous MF axes
    b. For each void
        i. If the void is in the stress space
            1. Open the voids that should open on this iteration
            2. For all open voids

Growing the voids:

a. Calculate stress applied by MF
b. Get the new locations of the μC crack tips

For interactions:

c. Obtain the effective geometry of the μC
d. Calculate stresses applied by the MF onto the μC
e. Convert the stresses into the mode-I and mode-II cracking stresses
f. Calculate the stress applied to the MF by the μC
g. Add result to cumulative sum of stresses from all the μCs considered so far in the present iteration

Record μC Data

h. Append the
    i. Void opened/closed status
    ii. Effective orientation
    iii. Effective length

    to the data frame storing location-based information on voids.

c. Determine principal stress and principal direction of pseudo-traction stress applied to th MF.
d. Calculate the incremental change in direction, $\Delta\theta$, from the rotation of a stress element to the principal direction
e. Determine direction of motion, $\theta_n = \theta_{n-1} + \Delta\theta$
f. move the MF to the next location OR Move the voids to the next location
g. Boolean to check if simulation should continue.
    i. If yes, repeat step 5

    If no, exit simulation

6. Save and post-process simulation output.

CAPABILITIES AND LIMITATIONS

Specifying Frame of Reference

Two frames of reference were implemented in the simulation; (1) the Lagrangian flow field specification, whereby the observer moves with the MF and watches the defects pass by (being on a bus and watching the world go by), and (2) the Eulerian flow field, whereby the observer is stationary with the material and watches crack pass through the material (e.g. watching live updates of the bus's location on a map). The selection of the frame of reference was based on computational expense and the best form for working with output data. A Lagrangian flow field specification was used in the PZ analysis as it allowed for data to be directly associated with the position in front of the MF, whereas a Eulerian flow field specification was used for the interaction analysis due to computational savings and improved visualisation of the result.

Appendix F PROOF OF PROPORTIONALITY OF APPLIED STRESS TO σ_∞

The Yoffe and Williams stress field solutions are such that stresses are proportional to the applied stress, σ_∞. This means stresses applied to microcracks are proportional to σ_∞, and by extension, the stresses that microcracks apply back onto the MF are also proportional σ_∞. Therefore, stresses for interactions are normalised by the stress applied to the main fracture.

This is still the case for mixed mode loading caused by an inclined MF, except the stress of proportionality becomes the far field major principal stress, which is σ_∞. (This is relevant to approach 2 & 3 for crack propagation)

Stress field of MF applied to μCs

$$\left(\sigma_x, \sigma_y, \sigma_{xy}\right) = \left(F(\sigma_\infty), G(\sigma_\infty), H(\sigma_\infty)\right) = \sigma_\infty(f, g, h)$$

Stresses μCs apply back onto the MF

$$\begin{aligned}\left(\sigma_{x,\mu C}, \sigma_{y,\mu C}, \sigma_{xy,\mu C}\right) &= \left(F'\left(\sigma_x, \sigma_y, \sigma_{xy}\right), G'\left(\sigma_x, \sigma_y, \sigma_{xy}\right), H'\left(\sigma_x, \sigma_y, \sigma_{xy}\right)\right) \\ &= \left(F'\left(\sigma_\infty \times (f, g, h)\right), G'\left(\sigma_\infty \times (f, g, h)\right), H'\left(\sigma_\infty \times (f, g, h)\right)\right) \\ &= \sigma_\infty\left(f'(f, g, h), g'(f, g, h), h'(f, g, h)\right) \\ = \sigma_\infty(f', g', h')\end{aligned}$$

In the case of in inclined crack with angle θ to the horizontal, under mixed mode loading for a unidirectional uniform stress applied at infinity (i.e. $\sigma_\infty = \sigma_{\infty,y}$), the normal and shear stresses are calculated as;

$$\sigma_{\infty,n} = \sigma'_y = \frac{\sigma_x+\sigma_y}{2} - \frac{\sigma_x-\sigma_y}{2}\cos(2\theta) - \tau_{xy}\sin(2\theta) = \frac{\sigma_\infty}{2} + \frac{\sigma_\infty}{2}\cos(2\theta) = \frac{\sigma_\infty}{2}(1 + \cos(2\theta)) \propto \sigma_\infty,$$

$$\sigma_{\infty,\tau} = \sigma'_{xy} = -\frac{\sigma_x-\sigma_y}{2}\sin(2\theta) + \tau_{xy}\cos(2\theta) = -\frac{\sigma_\infty}{2}\sin(2\theta) \propto \sigma_\infty,$$

■

which are proportional to σ_∞.

Appendix G DISCRETE MICROCRACKS – EXTENDED RESULTS

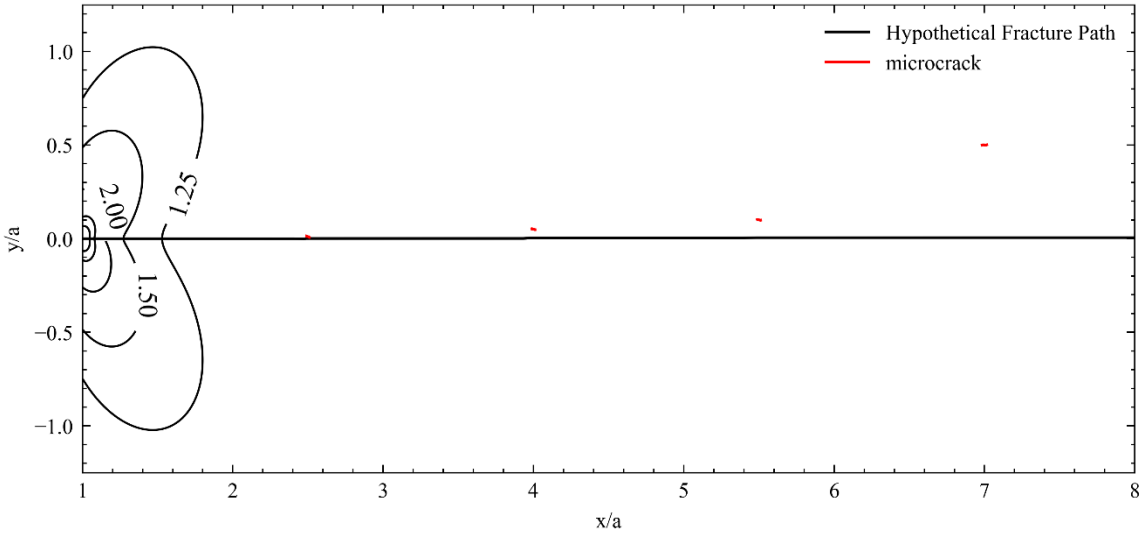


Figure 7.6: Contours for the dynamic Griffith crack stress field solution at the start of main fracture propagation. Voids are spaced such that the main fracture interacts with one microcrack at a time. Microcracks are shown in the final state which occurs when the main fracture passes.

Appendix H PROCESS ZONE PROPERTIES – SIMULATION GRID

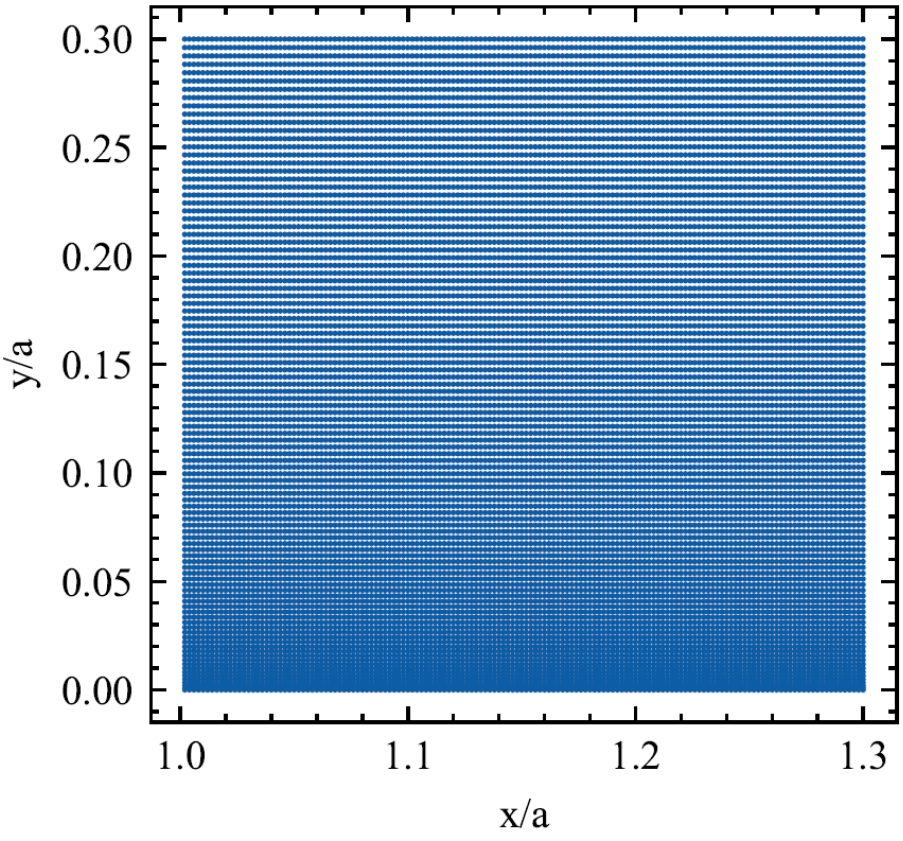


Figure 7.7: Grid on which process zone properties were measured. Data from 2500 voids was collected at each point.

Appendix I INTERCATIONS WITH MICROCRACK POPULATIONS – ADDITIONAL RESULTS AND COMMENTS

BASE CASE MICROCRACK WAKE AND STANDARD DEVIATION OF SLOPE

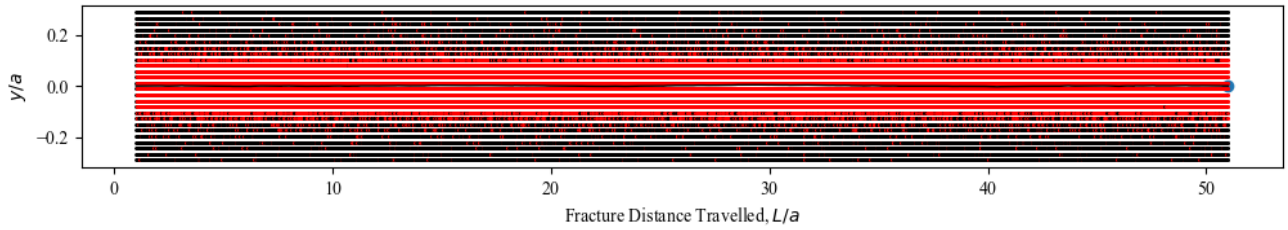


Figure 7.8: Crack path through a field of voids and the associated microcrack wake for a base case simulation. Only voids are shown. Red points indicate open voids, black points are closed voids.

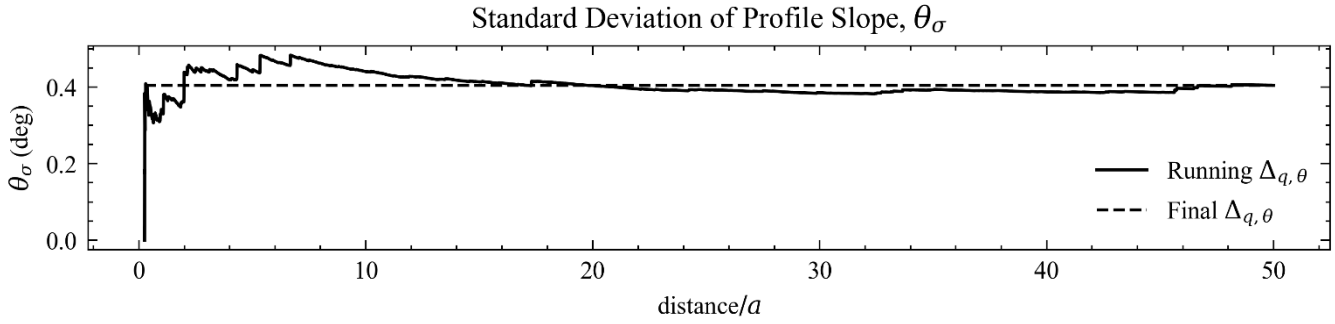


Figure 7.9: Change in the standard deviation of the θ-distribution along the path of the fracture. The standard deviation changes as the distribution of main fracture trajectories accumulates more values. The standard deviation reaches a constant value, indicating attainment of a steady state. It is at this point that full θ-distribution is assumed.

SENSITIVITY ANALYSIS – INPUTS

Weibull Distributions – Base Case

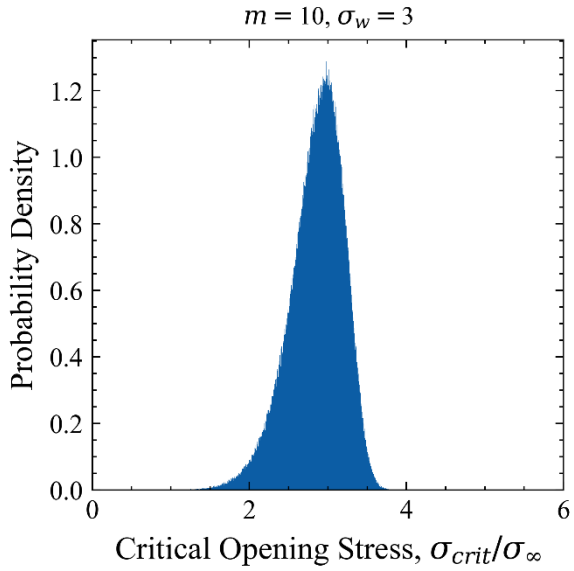


Figure 7.10: Weibull distribution generated with base case parameters for sampling vales of σ_{crit}.

Weibull Distributions – Shape Parameter

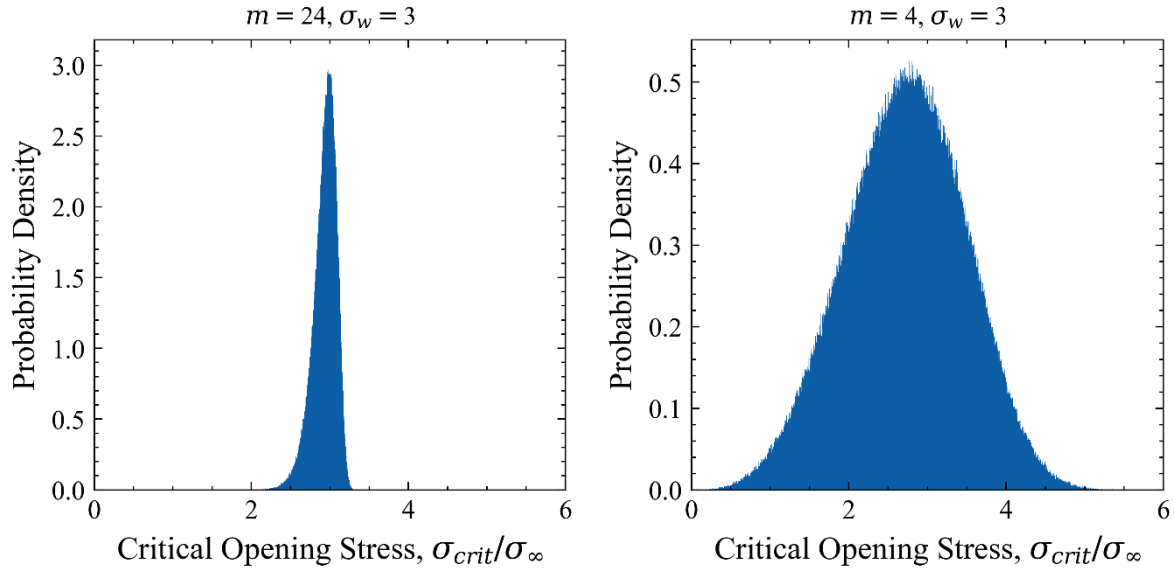


Figure 7.11: Weibull distribution for the (a) maximum and (b) minimum values of shape parameters, m, used in the sensitivity analysis of MF path roughness.

Weibull Distributions – Scale Parameter

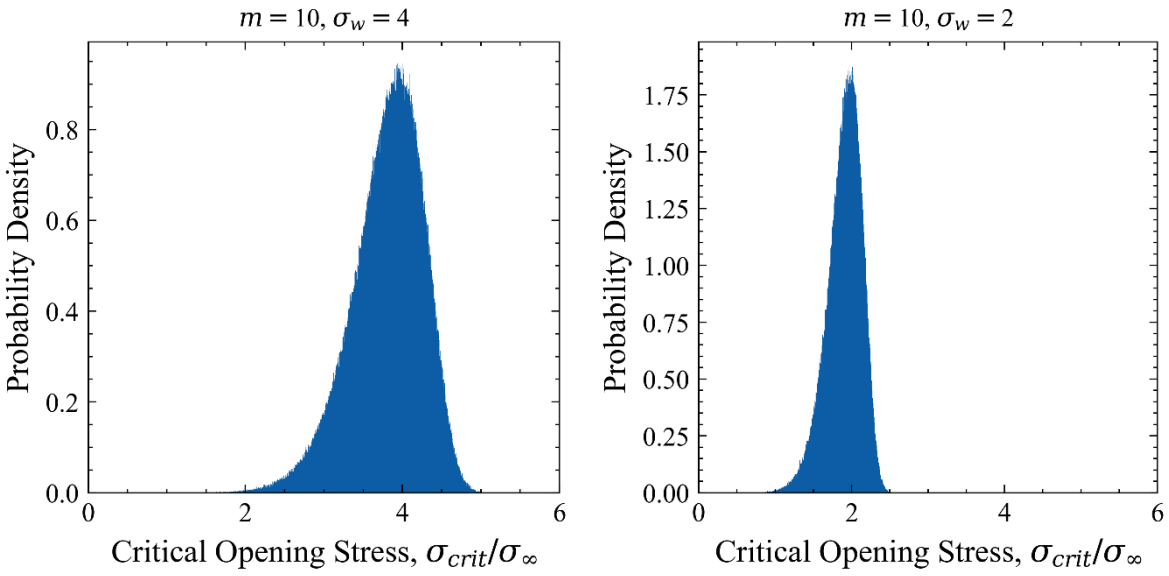


Figure 7.12: Weibull distribution for the (a) maximum and (b) minimum values of scale parameter, σ_w, used in the sensitivity analysis of MF path roughness.

Grid of μC Potential Locations – Base Case and Sensitivity

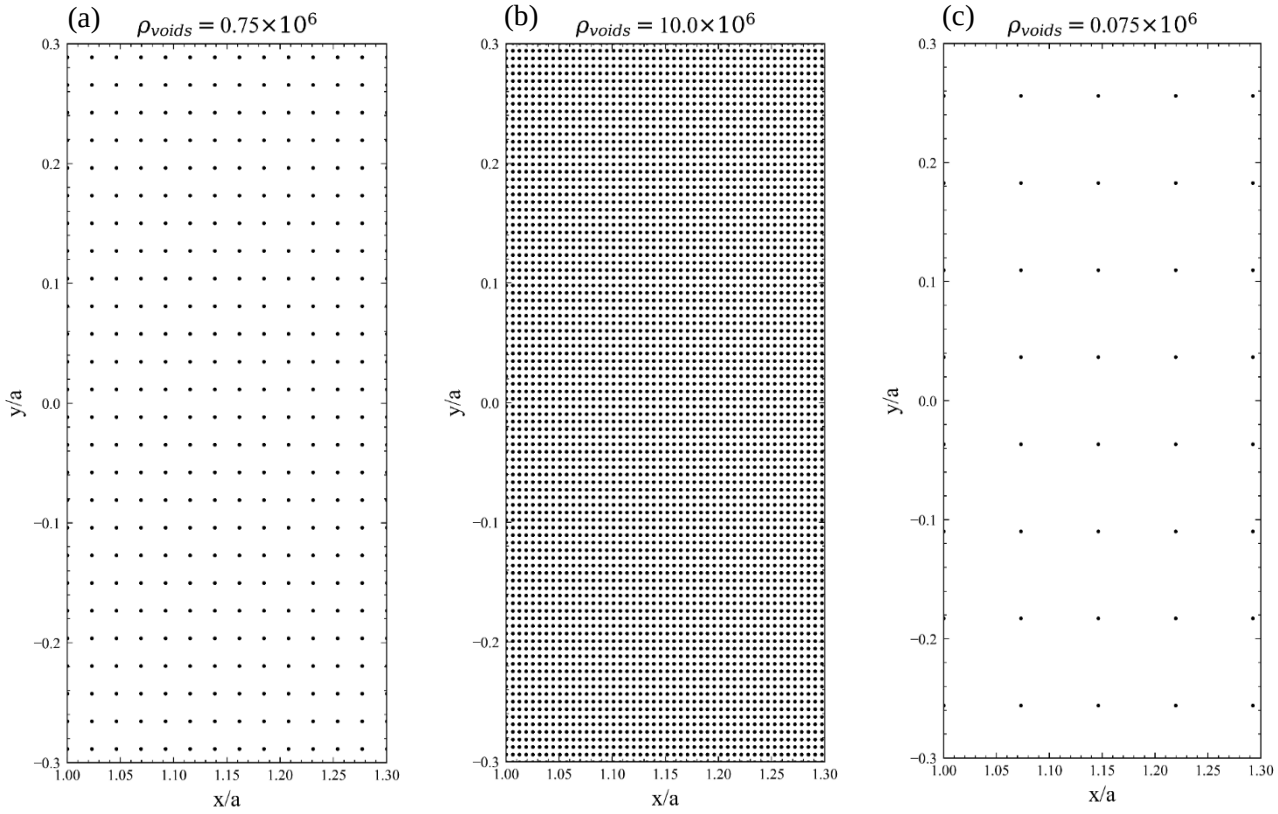


Figure 7.13: Void density for the (a) base case, (b) maximum and (c) minimum values of void density, ρ_{voids}, used in the sensitivity analysis of MF path roughness.

QUALITY OF THE GAUSSIAN FIT

The Gaussian PDF fitted to the θ-distribution demonstrates a varying quality of fit as seen in Figure 7.14. While the Gaussian has been demonstrated to adequately model the fracture path [55] and the instantaneous deviations of the fracture tip [56], in this research the quality of the fit appeared related to the development of the PZ. For example, as the density of μC populations increased, the PZ becomes more well defined, the μC population operates more as an entity. In cases of low μC density, or smaller critical opening stress, σ_{crit}, the PZ becomes less well defined and more sensitive to noise. The Cauchy distribution is also shown in Figure 7.14 for comparison. Where the Gaussian PDF yields a poorer quality fit, the Cauchy PDF demonstrates a reasonable fit. Further investigation is required to explain this.

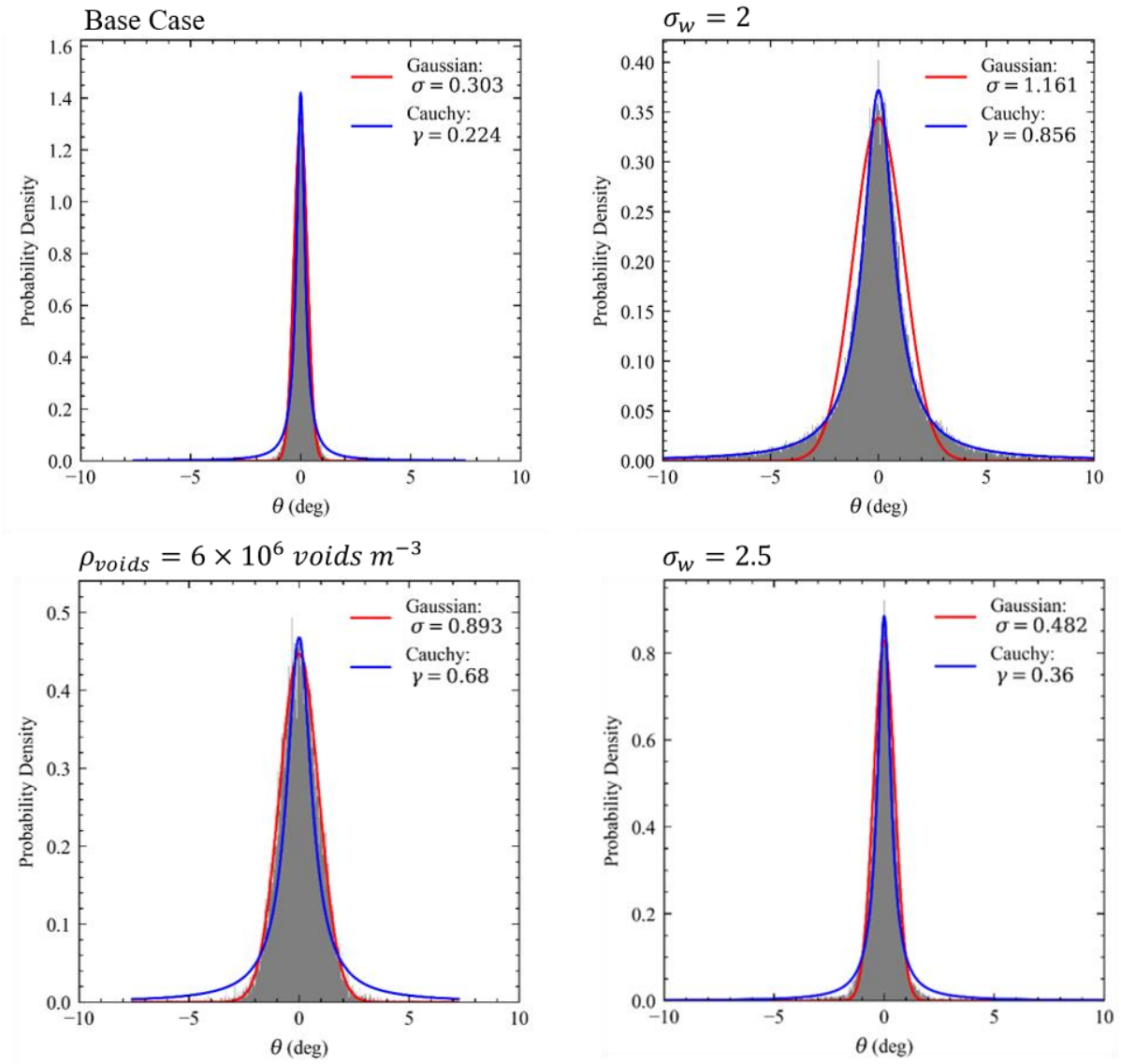


Figure 7.14: Distributions of main fracture orientation, θ, for a sample of sensitivity analyses to compare the quality of the fitted Gaussian PDF. A Cauchy PDF is also fitted to the θ-distribution to compare to the fit of the Gaussian PDF.

MOHR'S CIRCLE SIZE DISTRIBUTIONS

The Mohr's circle size distributions are plotted for extreme values of ρ_{voids} considered in the sensitivity study. Namely, $\rho_{voids} = 0.075 \times 10^6\ voids/m^2$ and $\rho_{voids} = 10 \times 10^6\ voids/m^2$ were considered.

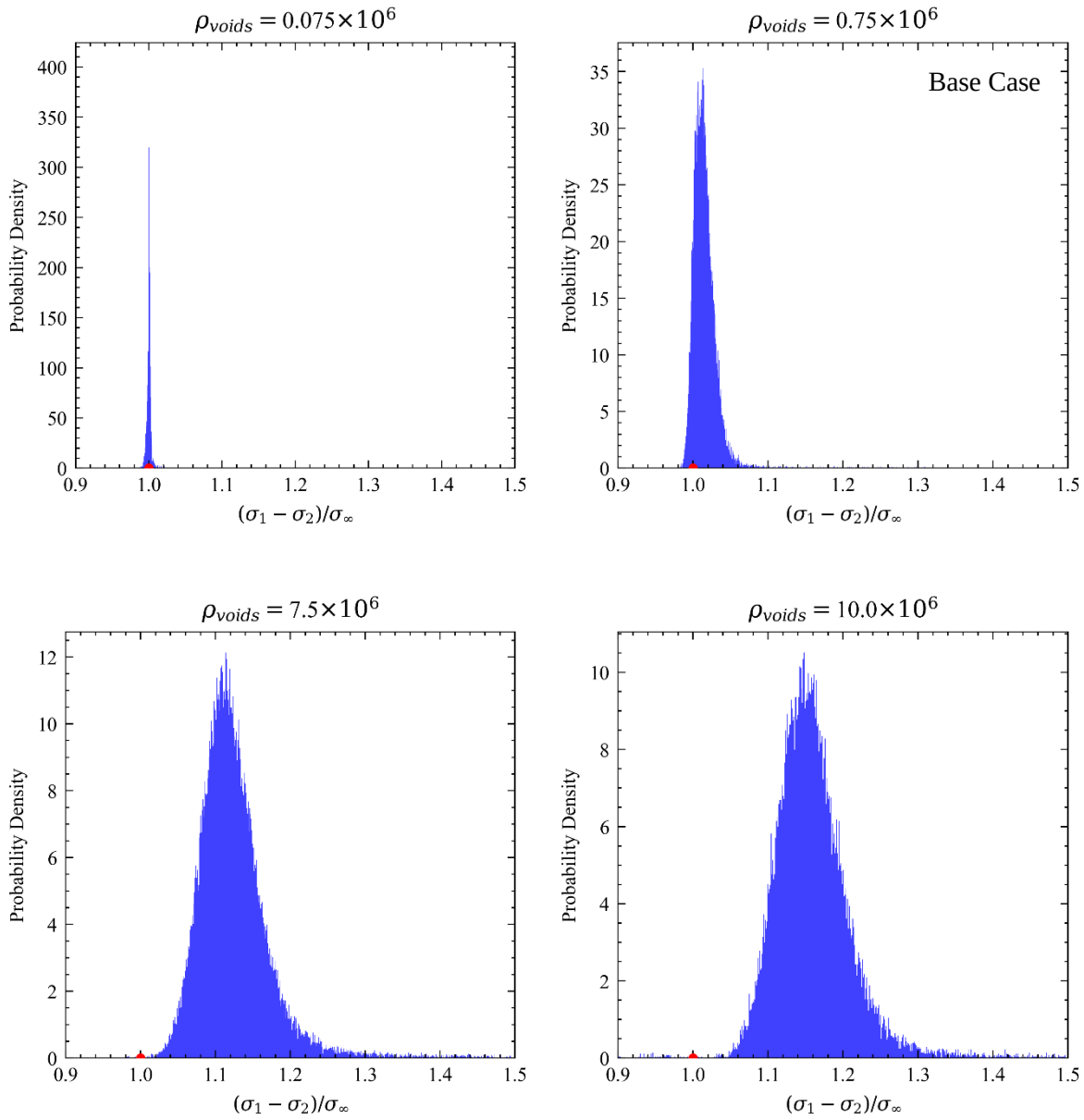


Figure 7.15: Distribution of the size of Mohr's circle measured along the main fracture path. Increasing the void density shifts the distribution in the positive direction, thereby indicating that larger Mohr's circles are typical in denser microcrack populations.

Appendix J PYTHON SCRIPTS

INPUT_PARAMETERS.PY

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 28 16:01:03 2020

Project: File that stores project parameters

Notes on Script:
    This script is for storing the project parameters.

    The use is that, if parameters are to be changed, they only need to be
    changed within this script. In other script, input variables will refer to
    this script to get their values.

    This means that the inputs only need to be changed once (in this script ONLY),
    and every other script can be run with the updated values without changing
    anything.


@author: Dominic Walker, 450239612, dwal9899
"""

# Import the required modules
# import math
import numpy as np


'''''Grid and Stress Parameters'''

    # Define the input parameters required for calculating stresses
rho_s = 2000                            # Density (kg/m**3)
E = 20*10**9                            # Elastic Modulus (Pa)
nu = 0.25                               # Poisson's ratio (-)

G = E*(1/2)*1/(1+nu)                    # Shear modulus (Pa)
Cs = np.sqrt(G/rho_s)                  # Shear wave velocity (m/s)
Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))    # Longitudinal wave velocity (m/s)

sigma_a_xx = 0.
sigma_a_yy = 0.0001*E                      # Applied Stress (Pa)
sigma_a_xy = 0.
sigma_a = np.array([sigma_a_xx, sigma_a_yy, sigma_a_xy])


V = 0.5*Cs;                             # Crack Velocity (m/s)


    # Define the geometry/mgrid being considered
    #   Define crack length 2a
a = 0.05 #0.01                              # Crack length (m)
```

MAIN_SCRIPT.PY

The investigation into the generation of the PZ and the characteristics μC populations and the study of MF-μC interactions were conducted separately. For each, the main script was optimised through targeted data collection and adjustments to the algorithms completed on each iteration.

Main Script – Optimised for PZ Analysis

```
# -*- coding: utf-8 -*-
"""
Project: Yoffe-Griffith crack - Process Zone Analysis


Notes & Assumptions - for STRESS DISTRIBUTION
    -   Ignore microcracks
    -   Only consider material in front of the crack in the direction that
        it is moving
    -   Plane strain

Parameters - for STRESS DISTRIBUTION
  sigma_a = applied stress = applied traction stress = -1x sigma_T (to
            ensure that the crack faces are traction free)
  V       = crack velocity [m/s]
  Cs      = shear wave velocity [m/s]
  Cl      = longitudinal wave velocity [m/s]
  rho_s   = material density [kg/m^3]
  a       = crack width [m]
  nu      = poisson's ratio []
  E       = elastic modulus [Pa]
  G       = elastic shear modulus = E/(2(1+nu)) [Pa]
  alpha   = 1 - nu
  K_I = sigma_aI*(pi*a)^1/2 = stress intensity factor
      Where,  sigma_aI = applied tensile stress sigma_yy in the y
                         direction
                       = sigma_a


@author: Dominic Walker, 450239612, dwal9899
"""

# Import the required modules
# import math
import numpy as np
from scipy.stats import weibull_min      # For calculating void opening stress
from scipy.stats.kde import gaussian_kde
import scipy.interpolate as interp
from scipy.spatial import distance

# import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D        # for 3d plotting
from matplotlib import cm                             # for colormaps

import seaborn as sns

# Import datetime to keep track of the simulation
import datetime

'exec(%matplotlib qt)' # Display plots in their own window

# Storing data for organised saving
import pandas as pd
```

```
55.
56.  # Saving figs and variables
57.  from pathlib import Path
58.  import dill
59.
60.  # Import the Modules built locally
61.  import stresses
62.  # from micro_VoidCrack_pointwiseStress_interact_V2 import microvoid
63.  from micro_VoidCrack_pointwiseStress_SimStage1 import microvoid
64.  import input_parameters
65.  #import plot_stresses
66.
67.
68.  # Set figure font
69.  # https://stackoverflow.com/questions/33955900/matplotlib-times-new-roman-appears-bold
70.  plt.rcParams["font.family"]="Times New Roman"
71.  # del matplotlib.font_manager.weight_dict['roman']
72.  # matplotlib.font_manager._rebuild()
73.
74.  # Suppress figure.max_opening_warning
75.  plt.rc('figure', max_open_warning = 0)
76.
77.
78.
79.  '''''Grid and Stress Parameters'''
80.
81.      # Define the input parameters required for calculating stresses
82.  rho_s = input_parameters.rho_s          # 7950                  # Density (kg/m**3)
83.  E = input_parameters.E                  # 200000*10**6          # Elastic Modulus (Pa)
84.  nu = input_parameters.nu                #0.25                  # Poisson's ratio (-)
85.
86.  G = 0.5*E/(1+nu)                                          # Shear modulus (Pa)
87.  Cs = np.sqrt(G/rho_s)                                     # Shear wave velocity (m/s)
88.  Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))                       # Longitudinal wave velocity (m/s)
89.
90.  sigma_a = input_parameters.sigma_a     #0.0001*E              # Applied Stress (Pa)
91.  V = input_parameters.V                 #0.5*Cs;               # Crack Velocity (m/s)
92.
93.
94.      # Define the geometry/mgrid being considered
95.      #   Define crack length 2a
96.  a = input_parameters.a                 #0.01                  # Crack length (m)
97.
98.      #   Define x, y limits and spacing
99.  x_lim = 1.3*a #4*a        # Change back to 3*a
100. x_min = a+10**(-4)
101. y_lim = 0.3*a #1*a  1.75*a       # Change back to 1.5*a
102. y_min = 0. #-1*y_lim
103. inc = a*0.01        # Change back to 0.005
104.
105. ##K_I = sigma_a*(np.pi*a)**(1/2)
106.
107.
108.
109. #   Get all x- and y-values
110. #x = np.arange(0,x_lim,inc)
111. #y = np.arange(-1*y_lim,y_lim,inc)
112.
113.
114. '''''Stress Field Grid'''
115. #   Get the meshgrid for x- and y-values.
116. #   These will be all the potential locations for voids.
117. YY, XX = np.mgrid[y_min:y_lim:inc, x_min:x_lim:inc]
118. #      x-values start from 0. This corresponds to the centre of the crack
119. #      y-values range between +-y_lim
120.
121.
122.
123. '''''Simulation Parameters'''
```

```
# Time-step
#   The timestep is related to the crack velocity and how much distance the crack covers between each iteration;
#   time_step = distance/crack_velocity.
#   Option a)   The distance between iterations can be specified in terms of the grid size.
#               A finer grid will allow for more possible locations of microcracks, a more accurate stress field,
#               and a smaller timestep. A smaller timestep is not needed to ensure that the advantages of a finer grid are secured.
#   Option b)   The distance between iteratoins can be specified as a fraction of the crack length.
#               Need to check if this can be justified with stress contours for different 'a'.
#               If stress is more localised for smaller 'a', then we can justify why the timestep should be related to the half crack length 'a'
#
#
#   Option c)   <Any other length in the model that can be used?>
# dl = 0.01*a     # Units: m      #0.01*a    # dl is a small displacement of the crack tip. In general, dl = sqrt(dx**2 + dy**2). In the case that dir_n is always 0, then dl = dx.
# dt = 0.01*0.01/V       # Units: s
dt = 100*10**(-9)    # Units: s
dl = V*dt     # Units: m      #0.01*a    # dl is a small displacement of the crack tip. In general, dl = sqrt(dx**2 + dy**2). In the case that dir_n is always 0, then dl = dx.



# Specify the number of voids per unit area of the grid.
#   As for the time step, this 'unit area' requires some length in the simulation to be defined.
#   As for dt, two internal simulation lengths are the grid size and the crack length.
#   Need to check if there are any other 'lengths' in the simulation that would make more sense to use.
#   For now, the crack length will be used to specify the micro-void density.
#   This is because, if a finer mesh is required, it doesn't make sense that the micro-void density changes as well
###void_density = 20      # That is, 10 voids for every area a^2


# Initial Direction of motion of crack (which is the negative of the direction of motion of the voids grid)
dir_i = 0.
dir_n = 0.      # Initialise variable for storing overall direction of crack wrt +ve x-axis.
dir_net = 0.    # dir_n and dir_net are the same thing. dir_net is used when we want to force the crack to travel straight but just read the direction it wants to go in. dir_n is when the crack is actually permitted to change direction.
'''''
# dir_i is for moving geometry in Lagrangian
# dir_n is the direction in which the crack moves - this is for moving geometry in Eulerian for the next iteration
# dir_net is the direction the crack faces - this is for calculating stresses

Approach 1: dir_n = dir_net = 0
Approach 2: dir_n = 0, dir_net =/= 0
Approach 3: dir_n = dir_net =/= 0

'''

'''''Voids Grid Geometry, Total Voids Count and Distributing Microvoids'''
# Voids Grid
#   Voids Grid Dimensions/Size
x_lim_voids = x_lim     #2*(x_lim-a) + a # The stress field grid is initially within the voids grid. The crack tip lines up with the left edge of the voids grid.
x_min_voids = x_min
y_lim_voids = y_lim
y_min_voids = y_min


# Summary of Voids Grid Coordinates
voids_bbox_x0 = np.array([x_min_voids, x_lim_voids, x_lim_voids, x_min_voids, x_min_voids])
voids_bbox_y0 = np.array([y_min_voids, y_min_voids, y_lim_voids, y_lim_voids, y_min_voids])

# Summary of Stress Field Grid Coordinates
stressField_bbox_x0 = np.array([x_min, x_lim, x_lim, x_min, x_min])
stressField_bbox_y0 = np.array([y_min, y_min, y_lim, y_lim, y_min])



# Notes on the Voids Grid:
#   a) When the permitted to change direction the magnitude of the factor for y_lim will need to be revised.
```

```
#   b) While the crack is moving horizontally (i.e. unable to turn), x_lim_voids along determines the length of the simulation.

'''''Distribute Void Locations Vertically'''
# Calculate all the unique x- and y- values to be considered in the simulation.
#   y-values will be spaced more closely near y=0 and more sparsely at the edges where abs(y) is maximised.
numberOfYvals = 100#75 #200              #80

y_distribution_method = 'Ypower9on7'

if y_distribution_method == 'Ygp':
    # Option 1: using a GP with np.geomspace() [exponential solution]
    yy_possibleVoidCoords = a*np.append(np.append(np.geomspace(-4, -0.001, num=int(numberOfYvals/2)), np.array([0])),np.geomspace(0.001, 4, num=int(numberOfYvals/2)))

elif y_distribution_method == 'Ylinear' :

    # Option 2: Linear solution. Divide the interval evenly (maybe fine grid near crack plane and course grid away from crack plane).
    yy_possibleVoidCoords == np.linspace(y_min,y_lim,numberOfYvals,endpoint=True)

elif y_distribution_method == 'Ypower3':

    # Option 3: Using Cubic Function to distribute the required number of points between -1 and 1 then mapping to the required range.
    x_method3 = np.linspace(0,1,numberOfYvals,endpoint=True)
    y_method3 = x_method3**3
    yy_possibleVoidCoords = y_lim*y_method3

elif y_distribution_method == 'Ypower5on3':

    # Option 4: Using cube root Function with another odd number power to distribute the required number of points between -1 and 1 then mapping to the required range.
    x_method3 = np.linspace(0,1,numberOfYvals,endpoint=True) #np.linspace(-1,1,numberOfYvals,endpoint=True)
    y_method4 = np.power(x_method3, 5./3.)
    yy_possibleVoidCoords = y_lim*y_method4

elif y_distribution_method == 'Ypower7on5':

    # Option 5: Same as method 4.
    x_method3 = np.linspace(0,1,numberOfYvals,endpoint=True)
    y_method5 = np.power(x_method3, 7./5.)
    yy_possibleVoidCoords = y_lim*y_method5

elif y_distribution_method == 'Ypower9on7':

    # METHOD 6: Quintic
    x_method3 = np.linspace(0.,1.,numberOfYvals,endpoint=True)
    y_method6 = np.power(x_method3, 9./7.)
    yy_possibleVoidCoords = y_lim*y_method6


#   x-values must be evenly spaced and equal to 'dl' so that voids always move to a location on the stresses grid (i.e. where the stress state is known).
##xx_possibleVoidCoords = np.arange(x_min_voids, x_lim_voids, dl)
# Only one value for x is needed. All the points will start on the same 'column' of points
xx_possibleVoidCoords_0 = np.ones(len(yy_possibleVoidCoords))*x_lim_voids



'''''Assign Voids To Each Point'''

# Specify the number of voids per m^2 of the grid.
# true_void_density = (0.01)*10**6 #0.01, 0.1, 1 => 25, 250, 2500     # voids/mm^2 x 10**6 m/mm^2 void_density*1/a**2
# void_density = int(true_void_density*a**2)                          # That is, x void/s for every area a^2


voidsPerGridPt = 2500#50 #1800  #1000*3     #1000*8
avg_void_density = voidsPerGridPt*len(yy_possibleVoidCoords)/((x_lim-x_min)*(y_lim-y_min))      # Average voids/m^2
# Determine void locations
```

```
x_void = np.array(xx_possibleVoidCoords_0.tolist()*voidsPerGridPt)
y_void = np.array(yy_possibleVoidCoords.tolist()*voidsPerGridPt)

##x_void = np.array(XX_voids.flatten().tolist()*voidsPerGridPt)
##y_void = np.array(YY_voids.flatten().tolist()*voidsPerGridPt)



'''''Microvoid Critical Opening Stress'''
# Inputs for Weibull Survival Probability Distribution
sigma_w = 3*(0.0001*E)  # !!!This value is completely unjustified and has no basis for selection!!!
m = 10                  # shape
# Calculate the critical opening stress for each microvoid. This only needs to be done once at the start of the simulation.
#   m=> shape parameter,
#   scale=> this is interpreted as some reasonable value at which microvoids would open,
#   size => number of samples
openingStress = weibull_min.rvs(m, loc=0, scale=sigma_w, size=len(x_void))     # This returns a numpy.ndarray of length equal to the to
talVoids_count


'''''Microvoid id numbers'''
mv_ID = np.arange(1,len(x_void)+1)


'''''Initialise Instances of the Microvoid Class'''
# Make a list of class instance objects that will store the information from each microcrack.
defectList = []

# Go throuch each x-value and initalise a class instance object and append it to the defectList
for i,__ in enumerate(x_void):
    defectList.append(microvoid(x_void[i],y_void[i],openingStress[i],mv_ID[i]))


'''''The Simulation'''

# Frame of Reference
# frameOref = 'Eulerian'
frameOref = 'Lagrangian'

# Intialise variables which will be used to track the position of the bounding box for the voids grid.
voids_bbox_x = voids_bbox_x0
voids_bbox_y = voids_bbox_y0

# Intialise variables which will be used to track the position of the bounding box for the stress field grid.
sField_bbox_x = stressField_bbox_x0.copy()
sField_bbox_y = stressField_bbox_y0.copy()

# Initialise a list of tuples (x,y) which is all the points that define the bounding box for the stress field grid.
# This variable will not change if a lagrangian flow field specification is used. If a Eulerian flow field specification is used it wil
l be recalculated on every iteration.
# Note: The last coordinates in sField_bbox lists (i.e. sField_bbox_x[-1] and sField_bbox_y[-1]) can be excluded
##sField_bbox_coords = [(sField_bbox_x[i], sField_bbox_y[i]) for i,_ in enumerate(sField_bbox_x[:-1])]   # <-- this is the slow way
sField_bbox_coords = list(zip(sField_bbox_x, sField_bbox_y))[:-1] # <-- this is the quick way.

# Initial position of Fracture tip
# Note: This assumes that the crack fracture with a horizontal crack plane and at the location (a,0)
main_cr_leadingtip_x = a
main_cr_leadingtip_y = 0.

# Initialise the origin of the MF coordinate axes
MF_origin_x = 0.
MF_origin_y = 0.


# Initialise variable that controls when the simulation stops
continue_sim = True

# Keep track of how long the simulation takes.
time_before = datetime.datetime.now()
```

```
320.
321.
322. # The crack propagation approaches are:
323. # Approach 1: Force the fracture to propagate in a straight line and record θ_III at each iteration.
324. # Approach 2: Force the crack to move in a straight line, but permit the crack to rotate.
325. # Approach 3: Allow the fracture to change direction and follow its own path. Record σ_III.
326. approach = 1 # Note: This simulation is hard-coded to Approach 1
327. # approach = 2 # Change isin_sGrid() if this approach is used. AND change stress_state() and mc_stress_applied to apply for Mode II loa
     ding
328. # approach = 3 # Change isin_sGrid() if this approach is used AND change stress_state() and mc_stress_applied to apply for Mode II load
     ing
329.
330. # Keep track of simulation iteration
331. i=0
332. i_max=int(((np.max(voids_bbox_x) - np.min(voids_bbox_x)))/dl)+1
333.
334.
335. '''''Initialise a dataframe to store simulation data'''
336.
337. # Initialise a variable to store all the data
338. #   column headders
339. # column_headders = ['MV_ID','x','y','MV_open','Microcrack_length_m','MC_direction_rad']
340. column_headders = ['x','y','MV_open','Microcrack_length_m','MC_direction_rad']
341.
342. #   Calculate the number of rows that the dataframe 'sumulation_data' will contain
343. simData_length = int(voidsPerGridPt*len(yy_possibleVoidCoords)*((x_void[0]-a)/dl))
344.
345. # Initialise an empty dataframe
346. simulation_data = pd.DataFrame(columns=column_headders, index = np.arange(simData_length))
347.
348. #   Initialise a list of length 'simData_length' for appending as column data in the simulation_data dataframe.
349. ###initialising_list = [None]*len(column_headders)
350. ###rows = [initialising_list]*simData_length
351. #simulation_data = np.array([['MV_ID','x','y','MV_open','Microcrack_length','Open_Microvoid_Age']])
352. ###simulation_data = pd.DataFrame(rows, columns=column_headders)
353.
354. # Initialise a variable that will keep track of what row to put the data into.
355. step = 0
356.
357.
358.
359. # Keep track of how long the simulation takes.
360. time_before = datetime.datetime.now()
361.
362.
363.
364.
365. while continue_sim == True:     # This condition may need to be updated to the length of the main crack path if direction changing is p
     ermitted.
366.
367.     # Keep track of simulation iteration
368.     i+=1
369.     print('iteration: {} of {}'.format(i,i_max))
370.
371.
372.     # Update defect list to only contain defects that are relevent - NOTE: This can only be used in Approach 1.
373.     ##!!!defectList = [defect for defect in defectList if defect.x_mv > np.min(sField_bbox_x)]        # CONFIRMED: THIS SAVES TIME
374.
375.     # Remove voids from defect list that open on the first iteration
376.     if i == 4:
377.         defectList = [mvoid for mvoid in defectList if mvoid.microvoid_open==False]
378.
379.     # APPLY STRESSES TO DEFECTS, GROW MICROCRACKS, MOVE POINTS
380.     # For each microvoid in the defect list:
381.     #   a) (if it is closed) check if microvoid should open (using MF stress field)
382.     #   b) 1. if a microvoid is open check each crack tip to see if they move.
383.     #   b) 2. if a crack tip has non-
     zero velocity, get the next_XYpoint() of the microcrack crack tip and add this point to the geometry array of that crack tip.
384.     #   C) Lagrangian: move all (x,y) values for the microvoid and microcrack points for the new location relative to the main crack
```

```
    #       in preparation for the next simulation step.
    #   OR
    #      Eulerian: move all (x,y) values for the main crack bbox in preparation for the next simulation step.
    for mvoid in defectList:

        # Check if mvoid is within stress field. If it is not, then ignore it. For a microcrack to be considered 'inside' the stress field, its associated microvoid must be inside the stress field.
        if mvoid.isin_sGrid(sField_bbox_x) == True:         #If approach 2 or 3 is used: mvoid.isin_sGrid(mvoid.x_mv, mvoid.y_mv, sField_bbox_coords) == True
            # Check if closed microvoids should be opened
            if mvoid.microvoid_open == False:
                #mvoid.mv_is_open()
                mvoid.mv_is_open(frameOref, MF_origin_x, MF_origin_y, dir_net)

            # Check if the microcracks of opened microvoids grow. If microcracks grow, get their geoemtry.
            if mvoid.microvoid_open == True:
                #mvoid.next_XYpoint(dt)
                # print(dir_n, type(dir_n))

                mvoid.next_XYpoint(dt, frameOref, MF_origin_x, MF_origin_y, dir_net, sField_bbox_coords)

                # # If the microvoid is open increase the age of the OPEN microvoid
                # mvoid.microVoid_ageSinceOpening += dl

                # Calculate effective geometry for data collection
                mvoid.mc_effectiveGeom()



        '''''Data Collection'''
        # Data to collect
        # mvoid_data = [mvoid.microvoid_ID,mvoid.x_mv, mvoid.y_mv,mvoid.microvoid_open,mvoid.a_eff*2, mvoid.inclAng]
        mvoid_data = [mvoid.x_mv, mvoid.y_mv,mvoid.microvoid_open,mvoid.a_eff*2, mvoid.inclAng]


        # Collect Data
        ###simulation_data = np.append(simulation_data, mvoid_data, axis=0)
        ###simulation_data.append(mvoid_data)
        ###simulation_data.loc[len(simulation_data)] = mvoid_data
        simulation_data.loc[step] = mvoid_data
        step += 1 # Increase the simulation step by 1 for the next microvoid

        # Map Microvoid/Micocrack geometry to new position
        # This needs to act on ALL instances of the 'microvoid' class (regardless of whether the MV is inside the stress field or not).

        if frameOref == 'Lagrangian': # This is where we follow the crack and the microvoids grid moves and rotates about the main crack tip

            # ROTATION:
            #   The centre of rotation is the main crack tip
            #main_cr_leadingtip_x
            #main_cr_leadingtip_y

            #   The angle of rotation will be calculated as some function depending on the nearby microcracks (their quantity, distribution and geometry, etc.)
            #   dir_i is the angle of rotation of the crack tip wrt crack axis and center of rotation at main crack tip.
            #   Therefore, the angle of rotation of the voids grid is in the opposite direction.
            dir_i_geo = dir_i   # For now assume that the direction of motion is a straight line so the angle of rotation is 0.
            # dir_i = -1*dir_i
            # dir_n_geo = -1*(dir_net - dir_n) # If we want to move the MF in a straight line (along it's original +ve x-axis)
            dir_n_geo = -1*dir_n

            # DISPLACEMENT:
            # Since the microvoids are moving towards the main crack tip, the displacemnt is negative.
            displace_r = -1*dl


            # Map Microvoid/Micocrack geometry to new position
```

```
            # This needs to act on ALL instances of the 'microvoid' class (regardless of whether the MV is inside the stress field or not).
            # for mvoid in defectList: # This is where we follow the crack and the microvoids grid moves and rotates about the main crack tip

            # The crack plane has direction dir_0. We want to move all the (x,y) coordinates a distance dt*V along the directed line with angle (-pi) to the x-axis (at least while the crack moves horizontally)
            #   Location of microvoid
            mvoid.x_mv, mvoid.y_mv = (mvoid.x_mv - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (mvoid.y_mv - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n_geo), (mvoid.x_mv - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (mvoid.y_mv - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n_geo)

            #   Numpy array that stores microvoid / microcrack geometry data.
            mvoid.x_vals, mvoid.y_vals = (mvoid.x_vals - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (mvoid.y_vals - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n_geo), (mvoid.x_vals - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (mvoid.y_vals - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n_geo)



    '''''Update Geometry for Next Iteration - This depends on if a Lagrangian or Eulerian Perspective is being used'''
    if frameOref == 'Lagrangian': # This is where we follow the crack and the microvoids grid moves and rotates about the main crack tip

        # ROTATION:
        #   The centre of rotation is the main crack tip
        #main_cr_leadingtip_x
        #main_cr_leadingtip_y

        #   The angle of rotation will be calculated as some function depending on the nearby microcracks (their quantity, distribution and geometry, etc.)
        #   dir_i is the angle of rotation of the crack tip wrt crack axis and center of rotation at main crack tip.
        #   Therefore, the angle of rotation of the voids grid is in the opposite direction.
        dir_i_geo = dir_i   # For now assume that the direction of motion is a straight line so the angle of rotation is 0.
        # dir_i = -1*dir_i
        # dir_n_geo = -1*(dir_net - dir_n) # If we want to move the MF in a straight line (along it's original +ve x-axis)
        dir_n_geo = -1*dir_n

        # DISPLACEMENT:
        # Since the microvoids are moving towards the main crack tip, the displacemnt is negative.
        displace_r = -1*dl

        # The equation for rotation some point (x,y) an angle θ about a point (p,q) AND then displacing hozontally by some distance d along -ve x-direction is given by: (SOURCE: https://math.stackexchange.com/questions/270194/how-to-find-the-vertices-angle-after-rotation)
        # x_new = (x-p)cos(θ)-(y-q)sin(θ)+p - d
        # y_new = (x-p)sin(θ)+(y-q)cos(θ)+q

        # Map Microvoid/Microcrack geoemtry to new coordinates
        # THIS IS DONE IN LOOP FOR MICROVOIDS!

        # Move the box bounding the voids grid
        # The crack plane has direction dir_0. We want to move all the (x,y) coordinates a distance dt*V along the directed line with angle (-pi) to the x-axis (at least while the crack moves horizontally)
        voids_bbox_x, voids_bbox_y = (voids_bbox_x - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (voids_bbox_y - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n_geo), (voids_bbox_x - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (voids_bbox_y - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n_geo)


    # This is where we are stationary with the voids grid and we watch the crack move about (translation + rotate) within the voids grid.
    # Rotation is about the crack tip and translation is along the crack plane, however only the stress field grid moves.
    # Note: The location of the crack tip is not obvious in this case. However, by keeping the crack itp location and the crack plane direction
    #       in their own variables it will be easier to keep track of where the crack is going
    elif frameOref == 'Eulerian':


        # ROTATION:
```

```
        #   The center of rotation and displacement is the main crack tip - the center of rotation is the current crack tip position for (main_cr_leadingtip_x, main_cr_leadingtip_y)

        # The angle of rotation on this iteration measured wrt the previous crack plane (anticlockwise positive)
        #   The angle of rotation will be calculated as some function depending on the nearby microcracks (their quantity, distribution and geometry, etc.)
        #   dir_i is the angle of rotation of the crack plane measured from the previous crack plane direction.
        #   Therefore, the angle of rotation of the stress field grid is in the SAME direction.
        # While the incremental change in direction dir_i should be used here, when the crack is not permitted to rotate, dir_i should be taken as zero - but this is not the case.
        # The following line of code works in the general case, regardless of whether the crack can rotate or not
        dir_i_geo = dir_i

        # dir_i = dir_0 # (rad) # This is important for rotating the current grid through an angle dir_i to get the new grid. dir_i can be though of as the relative angle between the previous crack plane and the new crack plane.

        ##print(dir_i - dir_i_geo, type(dir_i), type(dir_i_geo))

        # DISPLACEMENT:
        # The displacement of the main crack along the main crack axis.
        displace_r = dl

        # The equation for rotation some point (x,y) an angle θ about a point (p,q) AND then displacing hozontally by some distance d along -ve x-direction is given by
        # x_new = (x-p)cos(θ)-(y-q)sin(θ)+p  -  d
        # y_new = (x-p)sin(θ)+(y-q)cos(θ)+q

        # Map stress field grid to new positions
        ##XX, YY = (XX - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (YY - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n), (XX - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (YY - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n)

        # Move the box bounding the stress field grid
        # The crack plane has direction dir_0. We want to move all the (x,y) coordinates a distance dt*V along the directed line with angle (-pi) to the x-axis (at least while the crack moves horizontally)
        sField_bbox_x, sField_bbox_y = (sField_bbox_x - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (sField_bbox_y - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n), (sField_bbox_x - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (sField_bbox_y - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n)

        # Get new points of the stress field bounding box contained in a single list with tuple (x,y) elements.
        ##sField_bbox_coords = [(sField_bbox_x[i], sField_bbox_y[i]) for i,_ in enumerate(sField_bbox_x[:-1])] # <-
- This is the slow way
        sField_bbox_coords = list(zip(sField_bbox_x, sField_bbox_y))[:-1]


        # Get the new position of the main crack tip to achieve the appropriate displacements in the next iteration
        main_cr_leadingtip_x = main_cr_leadingtip_x + displace_r*np.cos(dir_n)
        main_cr_leadingtip_y = main_cr_leadingtip_y + displace_r*np.sin(dir_n)


        MF_origin_x = MF_origin_x + displace_r*np.cos(dir_n)    # = main_cr_leadingtip_x - a*np.cos(dir_n)
        MF_origin_y = MF_origin_y + displace_r*np.sin(dir_n)    # = main_cr_leadingtip_y - a*np.sin(dir_n)


    else:
        print('flow field specification needed')


    # Check if the simulation should be terminated.
    # If the main crack cracktip hasn't reached the end of the microcrack field (going horizontally), then continue the simulation
    if (frameOref == 'Lagrangian') & (main_cr_leadingtip_x > np.max(voids_bbox_x)):
        continue_sim = False

    if (frameOref == 'Eulerian') & (np.min(sField_bbox_x) > np.max(voids_bbox_x)):
        continue_sim = False


'''''END OF SIMULATION'''
```

```
559.
560.
561. '''''Simulation Time Check'''
562. # Time after
563. time_after = datetime.datetime.now()
564.
565. # Time taken
566. timetaken_minutes = (time_after - time_before).seconds/60
567. timetaken_whole_minutes = int(timetaken_minutes)
568. timetaken_leftoverSeconds = (timetaken_minutes - timetaken_whole_minutes)*60
569. print('The simulation required %0.0f minutes, %0.0f seconds to run.' %(timetaken_whole_minutes,timetaken_leftoverSeconds))
570.
571.
572.
573. # results = 'view'
574. results = 'save'
575.
576.
577.
578. quantify_voids_num = 'totalVoids' +str(numberOfYvals*voidsPerGridPt)
579.
580. approach_str = 'Approach_{}'.format(approach)
581.
582.
583.
584.
585. if results == 'save':
586.     # folder_name = approach_str+ '_' + frameOref + '_' + voids_distribution_method + '_' + quantify_voids_type + quantify_voids_num + '_' + 'normStep{}_normLength{}_'.format(norm_step,norm_length) + 'SPparams{}pts{}r_spNorm'.format(sp_num, r_sp/a)
587.     folder_name = approach_str+ '_' + frameOref + '_' + y_distribution_method + str(numberOfYvals) +'_'+ 'VoidPerPoint{}'.format(voidsPerGridPt) +'_'+ quantify_voids_num
588.
589. else: # In this case we want to save the figures
590.     # Note: If the folder exists change the folder name slightly so it is obvious which folder is the newer version.
591.
592.     folder_name='scrap'
593.
594. # Generate file path
595. # file_path = Path('C:\\Users\Kids\Desktop\Thesis - Python\Simulation_Stage_2_Results\Figures\\' + folder_name)
596. file_path = Path('Simulation_Stage_1_Results\\Runs\\' + folder_name)
597.
598. # Create Path if it doesn't exist already - https://stackoverflow.com/questions/273192/how-can-i-safely-create-a-nested-directory
599. file_path.mkdir(parents=True, exist_ok=True)
600.
601.
602.
603. '''''Save Simulation Data'''
604.
605. # Convert the data from a numpy array to a pandas dataframe
606. ###simulation_data = pd.DataFrame(columns = simulation_data[0], data=simulation_data[1:])
607.
608. # Inspect the result
609. #pd.set_option('display.max_columns', 500)
610. #simulation_data.head(3)
611.
612. # simulation_data = pd.DataFrame(columns = ['MV ID','x','y','MV open/closed','Microcrack length',' Open Microvoid Age'], data=simulation_data[1:])
613. ###simulation_data = simulation_data.copy()
614.
615.
616. file_name_simulation_data = '{}\simulation_data.pkl'
617.
618. # simulation_data.to_csv(file_name_simulation_data.format(file_path), index=False) #
619. simulation_data.to_pickle(file_name_simulation_data.format(file_path))
620.
621.
622.
623. '''''Save Parameters'''
624.
```

```
625.
626. # SIMULATION PARAMS:
627. # Produce the dataframe to save
628. column_headders_2 = ['approach', 'a_m', 'voidsPerGridPt', 'x_lim', 'x_min', 'y_lim', 'y_min', 'inc', 'yy_possibleVoidCoords', 'V_mPERs'
     , 'dl_m', 'dt_s', 'sigma_a_Pa','sField_bbox_x', 'sField_bbox_y', 'voids_bbox_x', 'voids_bbox_y', 'main_cr_leadingtip_x', 'main_cr_leadi
     ngtip_y']
629.
630.
631. # Create a zip object from two lists
632. simulation_Parameters_zip = zip(column_headders_2, [[approach], [a], [voidsPerGridPt], [x_lim], [x_min], [y_lim], [y_min], [inc], [yy_p
     ossibleVoidCoords], [V], [dl], [dt], [sigma_a],[sField_bbox_x], [sField_bbox_y], [voids_bbox_x], [voids_bbox_y], [main_cr_leadingtip_x]
     , [main_cr_leadingtip_y]])
633.
634.
635.
636. # Create a dictionary from zip object
637. simulation_Parameters_dict = dict(simulation_Parameters_zip)
638. # simulation_Parameters_dict
639.
640. simulation_Parameters_df = pd.DataFrame(data=simulation_Parameters_dict, columns=column_headders_2)
641. # simulation_Parameters_df
642.
643. file_name_Parameters = '{}\Parameters.pkl'
644.
645. # Save Data in Dataframes in the same folder (save as .pkl file)
646. simulation_Parameters_df.to_pickle(file_name_Parameters.format(file_path))
647.
648.
649.
650.
651. '''''Save Current Kernel State'''
652.
653.
654. file_name = '{}\Kernel.pkl'
655.
656. # Save Current Session
657. dill.dump_session(file_name.format(file_path))
```

Main Script – Crack Interactions Analysis

```
1.  # -*- coding: utf-8 -*-
2.  """
3.  Project: Yoffe-Griffith crack - Interactions
4.
5.
6.  Notes & Assumptions - for STRESS DISTRIBUTION
7.      -   Ignore microcracks
8.      -   Only consider material in front of the crack in the direction that
9.          it is moving
```

```
    -   Plane strain

Parameters - for STRESS DISTRIBUTION
  sigma_a = applied stress = applied traction stress = -1x sigma_T (to
            ensure that the crack faces are traction free)
  V       = crack velocity [m/s]
  Cs      = shear wave velocity [m/s]
  Cl      = longitudinal wave velocity [m/s]
  rho_s   = material density [kg/m^3]
  a       = crack width [m]
  nu      = poisson's ratio []
  E       = elastic modulus [Pa]
  G       = elastic shear modulus = E/(2(1+nu)) [Pa]
  alpha   = 1 - nu
  K_I = sigma_aI*(pi*a)^1/2 = stress intensity factor
      Where,  sigma_aI = applied tensile stress sigma_yy in the y
                         direction
                       = sigma_a


@author: Dominic Walker, 450239612, dwal9899
"""

# Import the required modules
# import math
import numpy as np
from scipy.stats import weibull_min      # For calculating void opening stress

import matplotlib.pyplot as plt

# Import datetime to keep track of the simulation
import datetime

'exec(%matplotlib qt)' # Display plots in their own window

# Storing data for organised saving
import pandas as pd

# Saving figs and variables
from pathlib import Path
import dill

# Import the Modules built locally
import stresses
# from micro_VoidCrack_pointwiseStress_interact_V2 import microvoid
from micro_VoidCrack_pointwiseStress_interact_V4 import microvoid
import input_parameters
#import plot_stresses

# Set figure font
# https://stackoverflow.com/questions/33955900/matplotlib-times-new-roman-appears-bold
plt.rcParams["font.family"]="Times New Roman"
# del matplotlib.font_manager.weight_dict['roman']
# matplotlib.font_manager._rebuild()

# Suppress figure.max_opening_warning
plt.rc('figure', max_open_warning = 0)

'''''Grid and Stress Parameters'''

    # Define the input parameters required for calculating stresses
rho_s = input_parameters.rho_s          # 7950                # Density (kg/m**3)
E = input_parameters.E                  # 200000*10**6        # Elastic Modulus (Pa)
nu = input_parameters.nu                #0.25                 # Poisson's ratio (-)

G = 0.5*E/(1+nu)                                          # Shear modulus (Pa)
Cs = np.sqrt(G/rho_s)                                       # Shear wave velocity (m/s)
Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))                          # Longitudinal wave velocity (m/s)

```

```
sigma_a = input_parameters.sigma_a     #0.0001*E              # Applied Stress (Pa)
V = input_parameters.V                 #0.5*Cs;               # Crack Velocity (m/s)


    # Define the geometry/mgrid being considered
    #   Define crack length 2a
a = input_parameters.a                 #0.01                  # Crack length (m)

    #   Define x, y limits and spacing
x_lim = 1.25*a #2.5*a        # Change back to 3*a
x_min = a+10**(-4)
y_lim = 0.3*a #1.25*a         # Change back to 1.5*a
y_min = -1*y_lim
inc = a*0.0025        # Change back to 0.005

##K_I = sigma_a*(np.pi*a)**(1/2)



'''''Stress Field Grid'''
#   Get the meshgrid for x- and y-values
YY, XX = np.mgrid[y_min:y_lim:inc, x_min:x_lim:inc]
#      x-values start from 0. This corresponds to the centre of the crack
#      y-values range between +-y_lim


'''''Stresses and Principal Plane Direction'''
#[sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stresses.stress_Griff(XX,YY,a,sigma_a,nu)
[sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stresses.stress_Yoffe(XX=XX, YY=YY, a=a, sigma_a=sigma_a[1], V=V, rho_s=rho_s, G=G
, nu=nu)

# Calculate Principal Stresses and Rotation required to get Principal Stresses
[sigma_1, sigma_2, rot_to_principal_dir] = stresses.transform2d_ToPrincipal(sigma_xx, sigma_yy, sigma_xy)

# Calculate Normalised Parametes
# Normalised Geometry (divide by half crack length, a)
[XX_norm, YY_norm] = [XX/a, YY/a]

#   Normalised Stresses
[sigma_xx_norm, sigma_yy_norm, sigma_xy_norm] = [sigma_xx/sigma_a[1], sigma_yy/sigma_a[1], sigma_xy/sigma_a[1]]
[sigma_1_norm, sigma_2_norm] = [sigma_1/sigma_a[1], sigma_2/sigma_a[1]]


'''''Simulation Parameters'''
# Time-step
#   The timestep is related to the crack velocity and how much distance the crack covers between each iteration;
#   time_step = distance/crack_velocity.
dt = 50*10**(-9)    # Units: s
dl = V*dt     # Units: m      #0.01*a    # dl is a small displacement of the crack tip. In general, dl = sqrt(dx**2 + dy**2). In the ca
se that dir_n is always 0, then dl = dx.


# Initial Direction of motion of crack (which is the negative of the direction of motion of the voids grid)
dir_0 = 0.
dir_n = 0.      # Initialise variable for storing overall direction of crack wrt +ve x-axis.
dir_net = 0.    # dir_n and dir_net are the same thing. dir_net is used when we want to force the crack to travel straight but just rea
d the direction it wants to go in. dir_n is when the crack is actually permitted to change direction.

'''''
# dir_i is for moving geometry in Lagrangian
# dir_n is the direction in which the crack moves - this is for moving geometry in Eulerian for the next iteration
# dir_net is the direction the crack faces - this is for calculating stresses

Approach 1: dir_n = dir_net = 0
Approach 2: dir_n = 0, dir_net =/= 0
Approach 3: dir_n = dir_net =/= 0

'''
```

```
'''''Voids Grid Geometry, Total Voids Count and Distributing Microvoids'''
# Voids Grid
#   Voids Grid Dimensions/Size
x_lim_voids = 200*(x_lim-
a) + a #70*a    # The stress field grid is initially within the voids grid. The crack tip lines up with the left edge of the voids grid
.
x_min_voids = x_min
y_lim_voids = 1*y_lim
y_min_voids = -1*y_lim_voids


# Summary of Voids Grid Coordinates
voids_bbox_x0 = np.array([x_min_voids, x_lim_voids, x_lim_voids, x_min_voids, x_min_voids])
voids_bbox_y0 = np.array([y_min_voids, y_min_voids, y_lim_voids, y_lim_voids, y_min_voids])

# Summary of Stress Field Grid Coordinates
stressField_bbox_x0 = np.array([x_min, x_lim, x_lim, x_min, x_min])
stressField_bbox_y0 = np.array([y_min, y_min, y_lim, y_lim, y_min])


# Specify the number of voids per m^2 of the grid.
true_void_density = (0.75)*10**6 #0.01, 0.1, 1 => 25, 250, 2500     # voids/mm^2 x 10**6 m/mm^2 void_density*1/a**2
void_density = int(true_void_density*a**2)                          # That is, x void/s for every area a^2


# Specify the void distribution method
voids_distribution_method = 'Deterministic_Grid'
# voids_distribution_method = 'Deterministic_Staggered'
# voids_distribution_method = 'Stochastic_Space'

if voids_distribution_method == 'Deterministic_Grid':

    #   Distribute voids in a grid with grid size depending on the void density
    #   The grid should be a square grid
    inc_voids = a/np.sqrt(void_density)


    # Ensure voids are initially spread evenly about the main crack axis
    # AND make sure no voids are put along the crack axis
    #   First Generate half of the voids grid
    YY_voids0, XX_voids0 = np.mgrid[0.5*inc_voids:y_lim_voids:inc_voids, x_min_voids:x_lim_voids:inc_voids]
    # Reflect the voids grid about the x axis and append this to the original 2d arrays
    XX_voids = np.append(XX_voids0, XX_voids0, axis=0)
    YY_voids = np.append(YY_voids0, -1*YY_voids0, axis=0)
    # XX_voids = XX_voids0
    # YY_voids = YY_voids0

    # Flatten 2d arrays so that they can be fed to the loop that generates microvoid class instances.
    x_void = XX_voids.flatten()
    y_void = YY_voids.flatten()

    totalVoids_count = len(x_void)

elif voids_distribution_method == 'Deterministic_Staggered':
    #   Test a single point
    numberOfPts = 15#1#50
    inc_voids = 0.05 #0.1
    start = inc_voids
    stop = round(inc_voids*(numberOfPts+1),4)
    x_void = np.arange(1,numberOfPts+1)*4*a
    y_void = np.arange(start,stop,inc_voids)*a   #np.array([0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 25])*a

    totalVoids_count = len(x_void)

elif voids_distribution_method == 'Stochastic_Space':

    # Calculate the number of voids required
    vGrid_area = (x_lim_voids-x_min_voids)*(y_lim_voids - y_min_voids)
    # Calculate the total number of voids to be inserted into voids grid.
```

```
    totalVoids_count = int(void_density*(vGrid_area/a**2))
    # totalVoids_count = 20

    #   Pseudo-randomly select a 'totalVoids_count' number values from between 0 and 1 to get all the x-coordintates of the microvoids.
    #   Then stretch and shift values in array along the x-axis to fit the range of x-values assigned to the voids grid.
    #   Repeat this for the y-values.
    x_void = (x_lim_voids-x_min_voids)*(np.random.rand(totalVoids_count)) + x_min_voids
    y_void = (y_lim_voids-y_min_voids)*(np.random.rand(totalVoids_count)) + y_min_voids

else:
    print('Select Void Distribution Method')


'''''Microvoid Critical Opening Stress'''
# Inputs for Weibull Survival Probability Distribution
# sigma_w = 1.5*(0.0001*E)  # !!!This value is completely unjustified and has no basis for selection!!!
sigma_w = 3*(0.0001*E)  # !!!This value is completely unjustified and has no basis for selection!!!
m = 10                   # shape

# Calculate the critical opening stress for each microvoid. This only needs to be done once at the start of the simulation.
#   m=> shape parameter,
#   scale=> this is interpreted as some reasonable value at which microvoids would open,
#   size => number of samples
openingStress = weibull_min.rvs(m, loc=0, scale=sigma_w, size=totalVoids_count)     # This returns a numpy.ndarray of length equal to the totalVoids_count
# openingStress = np.array([0.8*0.0001*E]*totalVoids_count)


'''''Microvoid id numbers'''
mv_ID = np.arange(1,len(x_void)+1)


'''''Initialise Instances of the Microvoid Class'''
# Make a list of class instance objects that will store the information from each microcrack.
defectList = []

# Go throuch each x-value and initalise a class instance object and append it to the defectList
for i,__ in enumerate(x_void):
    defectList.append(microvoid(x_void[i],y_void[i],openingStress[i],mv_ID[i]))
    # defectList.append(microvoid(x_void[i],y_void[i],20000000.0,mv_ID[i]))



'''''The Simulation'''

# Frame of Reference
frameOref = 'Eulerian'
# frameOref = 'Lagrangian'

# Intialise variables which will be used to track the position of the bounding box for the voids grid.
voids_bbox_x = voids_bbox_x0
voids_bbox_y = voids_bbox_y0

# Intialise variables which will be used to track the position of the bounding box for the stress field grid.
sField_bbox_x = stressField_bbox_x0.copy()
sField_bbox_y = stressField_bbox_y0.copy()

# Initialise a list of tuples (x,y) which is all the points that define the bounding box for the stress field grid.
# This variable will not change if a lagrangian flow field specification is used. If a Eulerian flow field specification is used it will be recalculated on every iteration.
# Note: The last coordinates in sField_bbox lists (i.e. sField_bbox_x[-1] and sField_bbox_y[-1]) can be excluded
##sField_bbox_coords = [(sField_bbox_x[i], sField_bbox_y[i]) for i,_ in enumerate(sField_bbox_x[:-1])]   # <-- this is the slow way
sField_bbox_coords = list(zip(sField_bbox_x, sField_bbox_y))[:-1] # <-- this is the quick way.

# Initial position of Fracture tip
# Note: This assumes that the crack fracture with a horizontal crack plane and at the location (a,0)
main_cr_leadingtip_x = a
main_cr_leadingtip_y = 0.
```

```
278.
279. # Initialise the origin of the MF coordinate axes
280. MF_origin_x = 0.
281. MF_origin_y = 0.
282.
283.
284. # Initialise variable that controls when the simulation stops
285. continue_sim = True
286.
287. # Keep track of how long the simulation takes.
288. time_before = datetime.datetime.now()
289.
290. '''''Crack interactions and Crack Path Params'''
291. # Intialise a variable to record the distance travelled by the MF
292. distance_travelled = 0.
293.
294. # Maximum allowable distance travelled
295. distance_travelled_max_all = 1.5*(max(voids_bbox_x) - min(voids_bbox_x))
296.
297. # Stress point configuration to use
298. multipleStressPoints = True
299.
300.
301. if multipleStressPoints == False: # single stres point
302.     # Location where stresses will be calculated in front of Fracture Tip
303.     main_cr_stressPt_x = a + dl #0.01*a
304.     main_cr_stressPt_y = 0.
305.
306. # Otherwise use multiple stress points
307. else:
308.
309.     # Number of stress points
310.     sp_num = 1#20 NOTE: If disappearance jump corection used, it will only work for sp_num=1
311.
312.     # Range in which angles are elected
313.     theta_sp = (np.linspace(start=-0.375, stop=0.375, num=sp_num+2, endpoint=True)*np.pi)[1:-1] # Angles will be selected between -
    1*pi/4 and pi/4
314.     # Circle radius
315.     r_sp = dl #0.01*a #0.0001*a
316.
317.     # Calculate stress points
318.     main_cr_stressPt_x = a + r_sp*np.cos(theta_sp)
319.     main_cr_stressPt_y =r_sp*np.sin(theta_sp)
320.
321. # Record the initial position of the stress point. The stresses due to only the effects MF will be calculated using these variables.
322. main_cr_stressPt_x_0 = main_cr_stressPt_x
323. main_cr_stressPt_y_0 = main_cr_stressPt_y
324.
325.
326.
327. # Initialise array to store the theoretical direction of motion of main fracture
328. # The direction of motion is w.r.t. the global x-axis
329. fracture_dir = np.array([[0.],[0.]]) # These values will be removed at the end of the simulation - they are just so it works on the fir
    st iteration.
330.
331.
332.
333. # Store the stress state in front of them main crack in a variable
334. fracture_stressState = np.array([[],[],[],[],[]]) #np.array([[sigma_xx],[sigma_yy],[sigma_xy],[sigma_1],[sigma_2]])
335. fracture_stressState_MF_only = np.array([[],[],[]]) #np.array([[sigma_xx],[sigma_yy],[sigma_xy]])
336.
337.
338. # The crack propagation approaches are:
339. # Approach 1: Force the fracture to propagate in a straight line and record θ_III at each iteration.
340. # Approach 2: Force the crack to move in a straight line, but permit the crack to rotate.
341. # Approach 3: Allow the fracture to change direction and follow its own path. Record σ_III.
342. approach = 1
343. # approach = 2 # Change isin_sGrid() if this approach is used. AND change stress_state() and mc_stress_applied to apply for Mode II loa
    ding.
```

```
# approach = 3 # Change isin_sGrid() if this approach is used AND change stress_state() and mc_stress_applied to apply for Mode II loading


# Set parameter gamma [1/sec] which controls delayed response.
# Gamma controls the sensitivity of the MF to the surrounding MCs.
# gamma = 1 --> immediate response to microcrack interactions.
# 0 < gamma <= 1
# Small gamma --> MF insensitive to MCs
gamma = 1#0.005

# Keep track of simulation iteration
i=0
i_max=int(((np.max(voids_bbox_x) - np.min(voids_bbox_x)))/dl)+1

'''''Plot Initial Geometry'''
#'''
# Define Grid Extents
stressField_gridExtents_x = stressField_bbox_x0/a
stressField_gridExtents_y = stressField_bbox_y0/a
voids_gridExtents_x = voids_bbox_x0/a
voids_gridExtents_y = voids_bbox_y0/a

# # Clear Current Instance of the 'Initial Geometry' figure.
# plt.close(r'Initial Geometry')

# # Make a plot that shows:
# #   Extents of the stress field grid (as a box)
# #   Extents of the voids grid (as a box)
# #   Locations of microvoids (as points)
# fig, ax = plt.subplots(1,1, constrained_layout = True, figsize = (15,5), num = r'Initial Geometry')

# # Plot Data:
# ax.plot(stressField_gridExtents_x,stressField_gridExtents_y, lw=2, label='Stress Field Grid Extents') # Plot NORMALISED Stress Field Grid Extents
# ax.plot(voids_gridExtents_x,voids_gridExtents_y, lw=1,label='Voids Grid Extents') # Plot NORMALISED Voids Grid Extents
# ax.scatter(x_void/a, y_void/a, c='k', s=1, label = 'micro-void') # Plot Microvoids with NORMALISED COORDINATES

# # Plot representation of MF
# # ax.axhline(y=0., xmin=-1, xmax=1, color='k', linewidth=1, label='Main Fracture') # x = 0
# ax.plot([-1,1], [0,0], color='k', linewidth=1, label='Main Fracture') # y = 0

# # Plot Stress Points
# ax.scatter(main_cr_stressPt_x_0/a,main_cr_stressPt_y_0/a, c='k', s=5, label = 'Stress Point')

# # Plot legend
# ax.legend(bbox_to_anchor=(0.9, 0.9, 0.1, 0.1))

# # Set figure title
# ax.set_title(r'Initial Geometry')

# # Set axis labels
# ax.set_xlabel('x/a')
# ax.set_ylabel('y/a')

# #ax.set_position([0.15,0.15, 0.85, 0.85])
# #'''
# plt.show()



# Run the simulation until the leading crack tip reaches the end of the voids grid
while continue_sim == True:     # This condition may need to be updated to the length of the main crack path if direction changing is permitted.

    # Keep track of simulation iteration
    i+=1
    print('iteration: {} of {}'.format(i,i_max))
```

```
410.
411.     # print(microvoid.sigma_a/sigma_a[1])
412.
413.     # Overwrite variables containing stresses with zeroes.
414.     '''''SHOULD INITIALISE WITH BACKGROUND STRESS FIELD + MF STRESS FIELD'''
415.     #   Initialise variables for storing stresses at fracture tip (sigma_MF + sigma_infty)
416.     # The stresses that are applied to the main crack depend on the orientation of the main crack wrt the initial MF axis
417.     # dir_net stores the current orientation of the MF measured from the initial MF +ve x-axis
418.     # To get the loading of the main fracture, a stress element in the initial axes orientation needs to be rotated to the current orie
    ntation.
419.     # sigma_a_yy and sigma_a_xy in the instantaneous MF orientation load the crack.
420.     # sigma_a_yy is the stress perpendicular to the crack, and sigma_a_xy is the shear stress. The normal stress parallel to the crack
    is ignored.
421.
422.     # sigma_a_xx, sigma_a_yy, sigma_a_xy = microvoid.global_MF_to_local_MF_stresses(0., sigma_a, 0., dir_net)
423.     sigma_a_xx, sigma_a_yy, sigma_a_xy = microvoid.global_MF_to_local_MF_stresses(sigma_a[0], sigma_a[1], sigma_a[2], dir_net) # STRESS
    ES ARE IN INSTANTANEOUS MF CRS
424.     #print([sigma_a_xx, sigma_a_yy, sigma_a_xy])
425.     # print('Top dir_net: {}'.format(dir_net))
426.     ###print(dir_net)
427.
428.     # Update the microvoid static variable for the stresses - NOTE: these stresses are in the instantaneous MF CRS
429.     microvoid.sigma_a = np.array([sigma_a_xx, sigma_a_yy, sigma_a_xy])
430.
431.     # Calculate the stresses felt at a point just in front of the Main Fracture
432.     # While this point moves in the grid as the crack moves, the point is stationary wrt the Main Fracture.
433.     # So, for stress calculation, the initial location of the stress point (main_cr_stressPt_x_0,main_cr_stressPt_y_0) will be used.
434.     # The orientation of the MF is not necessarily constant (if rotation is permitted) so the stress in front of the crack needs to be
    calculated on each iteration (for approach 2 and 3) accordingly.
435.     # MF_tipstress_xx, MF_tipstress_yy, MF_tipstress_xy, __, __ = stresses.stress_Griff(XX=main_cr_stressPt_x_0, YY=main_cr_stressPt_y_
    0, a=a, sigma_a=sigma_a_yy, nu=nu)
436.     # MF_tipstress_xx_II, MF_tipstress_yy_II, MF_tipstress_xy_II, __, __ = stresses.stress_Griff_II(XX=main_cr_stressPt_x_0, YY=main_cr
    _stressPt_y_0, a=a, sigma_aII=sigma_a_xy, nu=nu)
437.     MF_tipstress_xx, MF_tipstress_yy, MF_tipstress_xy, __, __ = stresses.stress_Yoffe(XX=main_cr_stressPt_x_0, YY=main_cr_stressPt_y_0,
     a=a, sigma_a=sigma_a_yy, V=V, rho_s=rho_s, G=G, nu=nu)
438.     MF_tipstress_xx_II, MF_tipstress_yy_II, MF_tipstress_xy_II, __, __ = stresses.stress_Yoffe_II(XX=main_cr_stressPt_x_0, YY=main_cr_s
    tressPt_y_0, a=a, sigma_aII=sigma_a_xy, V=V, rho_s=rho_s, G=G, nu=nu)
439.
440.
441.
442.
443.     fracture_stress_xx = sigma_a_xx # This is in instantaneous MF CRS
444.     fracture_stress_yy = sigma_a_yy
445.     fracture_stress_xy = sigma_a_xy
446.
447.
448.     # Record the stress at the MF tip due to the presence of the MF only (no MCs considered)
449.     # Note: appending is slow and only needs to be done if approach 2 or approach 3 are used.
450.     fracture_stressState_MF_only = np.append(fracture_stressState_MF_only, np.array([[np.mean(fracture_stress_xx)],[np.mean(fracture_st
    ress_yy)],[np.mean(fracture_stress_xy)]]), axis=1)
451.     # print(fracture_stress_yy)
452.
453.
454.     #   Initialise empty arrays for storing stresses over grid (for plotting)
455.     ##grid_stress_xx, grid_stress_yy, grid_stress_xy = sigma_xx.copy(), sigma_yy.copy(), sigma_xy.copy()
456.
457.     # Update defect list to only contain defects that are relevent - NOTE: This can only be used in Approach 1.
458.     ##!!!defectList = [defect for defect in defectList if defect.x_mv > np.min(sField_bbox_x)]        # CONFIRMED: THIS SAVES TIME
459.
460.
461.     # APPLY STRESSES TO DEFECTS, GROW MICROCRACKS, MOVE POINTS
462.     # For each microvoid in the defect list:
463.     #   a) (if it is closed) check if microvoid should open (using MF stress field)
464.     #   b) 1. if a microvoid is open check each crack tip to see if they move.
465.     #   b) 2. if a crack tip has non-
    zero velocity, get the next_XYpoint() of the microcrack crack tip and add this point to the geometry array of that crack tip.
466.     #   C) Lagrangian: move all (x,y) values for the microvoid and microcrack points for the new location relative to the main crack
467.     #       in preparation for the next simulation step.
468.     #   OR
```

```
    #      Eulerian: move all (x,y) values for the main crack bbox in preparation for the next simulation step.
    for mvoid in defectList:

        # Check if mvoid is within stress field. If it is not, then ignore it. For a microcrack to be considered 'inside' the stress field, its associated microvoid must be inside the stress field.
        if mvoid.isin_sGrid(sField_bbox_x) == True:         #If approach 2 or 3 is used: mvoid.isin_sGrid(mvoid.x_mv, mvoid.y_mv, sField_bbox_coords) == True
            # Check if closed microvoids should be opened
            if mvoid.microvoid_open == False:
                #mvoid.mv_is_open()
                mvoid.mv_is_open(frameOref, MF_origin_x, MF_origin_y, dir_net)

            # Check if the microcracks of opened microvoids grow. If microcracks grow, get their geoemtry.
            if mvoid.microvoid_open == True:
                #mvoid.next_XYpoint(dt)
                # print(dir_n, type(dir_n))

                mvoid.next_XYpoint(dt, frameOref, MF_origin_x, MF_origin_y, dir_net, sField_bbox_coords)


            # If there is a microcrack calculate stresses applied from microcrack onto:
                # a) Main Fracture
                # b) Grid for plotting
            #
            # if mvoid.microcrack_sprouted == True:         <-- A MICROVOID THAT IS OPEN HAS SPROUTED
                # Get effective microcrack geometry
                mvoid.mc_effectiveGeom()

                # Calculate stresses applied to microcrack using points describing microcrack geometry (this might have some run-time issues, maybe use less points)
                # Note: Global coordinates are being used
                mvoid.mc_stress_applied(frameOref, MF_origin_x, MF_origin_y, dir_net)

                # MAIN FRACTURE:
                #   Calculate stresses in front of MF resulting from the presence of this MC
                sigma_xx_MF, sigma_yy_MF, sigma_xy_MF = mvoid.interaction(main_cr_stressPt_x, main_cr_stressPt_y, dir_net, frameOref)

                #   Add stresses to variables storing the total effect of all the microcracks on the main fracture.
                fracture_stress_xx += sigma_xx_MF
                fracture_stress_yy += sigma_yy_MF
                fracture_stress_xy += sigma_xy_MF
                # Note: Stresses are in the instantaneous MF CRS


    # If multiple stress points are used, calculate the mean stress acting on the MF.
    # The line below works correctly regardless of if a single stress point is used or multiple are used.
    fracture_stress_xx, fracture_stress_yy, fracture_stress_xy = np.mean(fracture_stress_xx), np.mean(fracture_stress_yy), np.mean(fracture_stress_xy)

    # DETERMINE stress state in front of main fracture and hence MAIN FRACTURE DIRECTION OF MOTION
    #   Calculate principal stresses
    fracture_stress_1, fracture_stress_2, fracture_rot_to_principal = stresses.transform2d_ToPrincipal(fracture_stress_xx, fracture_stress_yy, fracture_stress_xy)

    # Append (rectangular & principal) stresses to the array storing all the stress info. The
    # Store the stress state in front of them main crack in a variable
    fracture_stressState = np.append(fracture_stressState, np.array([[fracture_stress_xx],[fracture_stress_yy],[fracture_stress_xy],[fracture_stress_1],[fracture_stress_2]]), axis=1)


    # When recording stresses, we should record them in terms of both
    #   a) instantaneous MF CRS, and    - We use this to calculate dir_i
    #   b) initial MF CRS               - Could go directly to dir_n if we did this


    #   Determine the direction dir_i that the fracture wants to travel wrt its own MF axes.
    if fracture_stress_yy >= fracture_stress_xx:
```

```
        dir_i = np.arctan(np.tan(-1*float(fracture_rot_to_principal))) #-1*float(fracture_rot_to_principal)


    #   This is the case where sigma_yy < sigma_xx
    else:
        dir_i = np.arctan(np.tan(-1*float(fracture_rot_to_principal) + np.pi/2))

    # Set up delayed / dampened response of MF to MCs to get smoother crack path
    dir_i = dir_i*gamma


    #   To get the net direction of motion wrt initial position, dir_n, sum up all dir_i from all iterations - this is irrespective of frame of reference being used (Lagrangian/Eulerian), dir_n should always be the same
    if approach == 1:
        dir_net = 0. # Crack is not permitted to rotate
        dir_n = 0. # Zero since the crack is forced to move in a straight line with zero rotatoin  #dir_i

    elif approach == 2:
        # In this case the crack is forced to move in a straight line but permitted to rotate.
        # Therefore, geometry should be determined using dir_n = 0,
        # and, stresses should be determined using dir_net (from the previous iteration)
        dir_net += dir_i # Crack is permitted to rotate
        dir_n = 0. # Zero since the crack is forced to move in a straight line with zero rotation  #dir_i

    elif approach == 3:
        dir_n += dir_i # Crack is permitted to rotate and travel in the same direction
        dir_net = dir_n # Crack is permitted to rotate and travel in the same direction


    # Append the direction data and the location of the crack tip to an array
    fracture_dir = np.append(fracture_dir, np.array([[float(dir_i)], [float(dir_net)]]), axis=1)




    '''''Update Geometry for Next Iteration - This depends on if a Lagrangian or Eulerian Perspective is being used
    Note: Lagrangian => moving global reference system - moves with crack
          Eulerian   => stationary global reference system - set at initial position of main fracture'''
    if frameOref == 'Lagrangian': # This is where we follow the crack and the microvoids grid moves and rotates about the main crack tip

        # ROTATION:
        #   The centre of rotation is the main crack tip
        #main_cr_leadingtip_x
        #main_cr_leadingtip_y

        #   The angle of rotation will be calculated as some function depending on the nearby microcracks (their quantity, distribution and geometry, etc.)
        #   dir_i is the angle of rotation of the crack tip wrt crack axis and center of rotation at main crack tip.
        #   Therefore, the angle of rotation of the voids grid is in the opposite direction.
        dir_i_geo = -1*(fracture_dir[1,-1]-fracture_dir[1,-2])   # For now assume that the direction of motion is a straight line so the angle of rotation is 0.
        # dir_i = -1*dir_i
        # dir_n_geo = -1*(dir_net - dir_n) # If we want to move the MF in a straight line (along it's original +ve x-axis)
        dir_n_geo = -1*dir_n

        # DISPLACEMENT:
        # Since the microvoids are moving towards the main crack tip, the displacemnt is negative.
        displace_r = -1*dl


        # Move the box bounding the voids grid
        # The crack plane has direction dir_0. We want to move all the (x,y) coordinates a distance dt*V along the directed line with angle (-pi) to the x-axis (at least while the crack moves horizontally)
        voids_bbox_x, voids_bbox_y = (voids_bbox_x - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (voids_bbox_y - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n_geo), (voids_bbox_x - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (voids_bbox_y - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n_geo)

```

```
594.
595.         # Map Microvoid/Micocrack geometry to new position
596.         # This needs to act on ALL instances of the 'microvoid' class (regardless of whether the MV is inside the stress field or not).
597.         for mvoid in defectList: # This is where we follow the crack and the microvoids grid moves and rotates about the main crack tip
598.
599.             # The crack plane has direction dir_0. We want to move all the (x,y) coordinates a distance dt*V along the directed line with angle (-pi) to the x-axis (at least while the crack moves horizontally)
600.             #   Location of microvoid
601.             mvoid.x_mv, mvoid.y_mv = (mvoid.x_mv - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (mvoid.y_mv - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n_geo), (mvoid.x_mv - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (mvoid.y_mv - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n_geo)
602.
603.             #   Numpy array that stores microvoid / microcrack geometry data.
604.             mvoid.x_vals, mvoid.y_vals = (mvoid.x_vals - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (mvoid.y_vals - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n_geo), (mvoid.x_vals - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (mvoid.y_vals - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n_geo)
605.
606.
607.     # This is where we are stationary with the voids grid and we watch the crack move about (translation + rotate) within the voids grid.
608.     # Rotation is about the crack tip and translation is along the crack plane, however only the stress field grid moves.
609.     # Note: The location of the crack tip is not obvious in this case. However, by keeping the crack itp location and the crack plane direction
610.     #       in their own variables it will be easier to keep track of wher ethe crack is going
611.     elif frameOref == 'Eulerian':
612.
613.
614.         # ROTATION:
615.         #   The center of rotation and displacement is the main crack tip - the center of rotation is the current crack tip position for (main_cr_leadingtip_x, main_cr_leadingtip_y)
616.
617.         # The angle of rotation on this iteration measured wrt the previous crack plane (anticlockwise positive)
618.         #   The angle of rotation will be calculated as some function depending on the nearby microcracks (their quantity, distribution and geometry, etc.)
619.         #   dir_i is the angle of rotation of the crack plane measured from the previous crack plane direction.
620.         #   Therefore, the angle of rotation of the stress field grid is in the SAME direction.
621.         # While the incremental change in direction dir_i should be used here, when the crack is not permitted to rotate, dir_i should be taken as zero - but this is not the case.
622.         # The following line of code works in the general case, regardless of whether the crack can rotate or not
623.         dir_i_geo = fracture_dir[1,-1]-fracture_dir[1,-2]
624.
625.         # dir_i = dir_0 # (rad) # This is important for rotating the current grid through an angle dir_i to get the new grid. dir_i can be though of as the relative angle between the previous crack plane and the new crack plane.
626.
627.         ##print(dir_i - dir_i_geo, type(dir_i), type(dir_i_geo))
628.
629.         # DISPLACEMENT:
630.         # The displacement of the main crack along the main crack axis.
631.         displace_r = dl
632.
633.         # The equation for rotation some point (x,y) an angle θ about a point (p,q) AND then displacing hozontally by some distance d along -ve x-direction is given by
634.         # x_new = (x-p)cos(θ)-(y-q)sin(θ)+p  -  d
635.         # y_new = (x-p)sin(θ)+(y-q)cos(θ)+q
636.
637.         # Map stress field grid to new positions
638.         ##XX, YY = (XX - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (YY - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n), (XX - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (YY - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n)
639.
640.         # Move the box bounding the stress field grid
641.         # The crack plane has direction dir_0. We want to move all the (x,y) coordinates a distance dt*V along the directed line with angle (-pi) to the x-axis (at least while the crack moves horizontally)
642.         sField_bbox_x, sField_bbox_y = (sField_bbox_x - main_cr_leadingtip_x)*np.cos(dir_i_geo) - (sField_bbox_y - main_cr_leadingtip_y)*np.sin(dir_i_geo) + main_cr_leadingtip_x + displace_r*np.cos(dir_n), (sField_bbox_x - main_cr_leadingtip_x)*np.sin(dir_i_geo) + (sField_bbox_y - main_cr_leadingtip_y)*np.cos(dir_i_geo) + main_cr_leadingtip_y + displace_r*np.sin(dir_n)
643.
```

```
        # Get new points of the stress field bounding box contained in a single list with tuple (x,y) elements.
        ##sField_bbox_coords = [(sField_bbox_x[i], sField_bbox_y[i]) for i,_ in enumerate(sField_bbox_x[:-1])] # <-
    - This is the slow way
        sField_bbox_coords = list(zip(sField_bbox_x, sField_bbox_y))[:-1]


        # Get the new position of the main crack tip to achieve the appropriate displacements in the next iteration
        main_cr_leadingtip_x = main_cr_leadingtip_x + displace_r*np.cos(dir_n)
        main_cr_leadingtip_y = main_cr_leadingtip_y + displace_r*np.sin(dir_n)


        MF_origin_x = MF_origin_x + displace_r*np.cos(dir_n)    # = main_cr_leadingtip_x - a*np.cos(dir_n)
        MF_origin_y = MF_origin_y + displace_r*np.sin(dir_n)    # = main_cr_leadingtip_y - a*np.sin(dir_n)

        main_cr_stressPt_x = main_cr_stressPt_x + displace_r*np.cos(dir_n)
        main_cr_stressPt_y = main_cr_stressPt_y + displace_r*np.sin(dir_n)


    else:
        print('flow field specification needed')



    # Check if the simulation should be terminated.
    # If the main crack cracktip hasn't reached the end of the microcrack field (going horizontally), then continue the simulation
    if (frameOref == 'Lagrangian') & (main_cr_leadingtip_x > np.max(voids_bbox_x)):
        continue_sim = False

    if (frameOref == 'Eulerian') & (np.min(sField_bbox_x) > np.max(voids_bbox_x)):
        continue_sim = False


    # Only allow the MF to travel a distance equal to 4 times its length
    distance_travelled += dl
    if distance_travelled >= distance_travelled_max_all: #16*2*a:
        continue_sim = False
        print('MF Max Distance Travelled Reached')



# Remove the first column
fracture_dir = fracture_dir[:,1:]


'''''END OF SIMULATION'''



'''''Simulation Time Check'''
# Time after
time_after = datetime.datetime.now()

# Time taken
timetaken_minutes = (time_after - time_before).seconds/60
timetaken_whole_minutes = int(timetaken_minutes)
timetaken_leftoverSeconds = (timetaken_minutes - timetaken_whole_minutes)*60
print('The simulation required %0.0f minutes, %0.0f seconds to run.' %(timetaken_whole_minutes,timetaken_leftoverSeconds))


'''''Variables for Statical Analysis and Plotting'''
# Calculate distance travelled
# At each step the main crack travels dl
distance = np.arange(0,fracture_dir.shape[1])*dl

# Need to find the index in distance where the values is >= (x_lim-x_min) - all data after this point will be considered in plots.
startData_index = np.searchsorted(distance, (x_lim-x_min), side='left')

# # Get distance from the start of where the collected data will be used
```

```
# distance_data = np.arange(0,fracture_dir[:,startData_index:].shape[1])*dl

# Make the correction to the fracture_dir array if Approach 1 was used.
# The first row needs to be θ (theta_dot)
# The second row needs to be θ(t)
if approach == 1:
    theta = fracture_dir[0]
    theta_inc = np.append(np.array([fracture_dir[0,i+1] - fracture_dir[0,i] for i in np.arange(0,fracture_dir.shape[1]-1)]), [0.], axis=0)

elif (approach==2) | (approach==3):
    theta_inc = fracture_dir[0]
    theta = fracture_dir[1]
else:
    print('Approach not specified.')


# If the crack was forced to move in a straight line, the pseudo path needs to be calculated
# We know the distance that the crack each time, and we know the angles
# this means we can calculate the displacements at each time step,
# then accumulate all the previous displacements at each point to get the position of the crack tip wrt the initial position of the MF axes

# This gives the incremental changes in x any y wrt the instantaneous MF axes. The increments
##x_inc_instantaneous = dl*np.cos(theta_inc)
##y_inc_instantaneous = dl*np.sin(theta_inc)

# A note on units:
#   The units of theta_inc are rad/(distance dl)
#   The units of theta are rad
#   The units of theta_dot are rad/s or deg/sec
theta_dot = theta_inc/dt


# Overall changes in x and y wrt initial MF axes
x_inc = dl*np.cos(theta)
y_inc = dl*np.sin(theta)

# Overall changes in Vx and Vy wrt initial MF axes
Vx = V*np.cos(theta)
Vy = V*np.sin(theta)

# The pseudo crack path or actual crack path is determined wrt the initial MF axes.
# Therefore, the crack path is determined my summing all the small displacements Δx and Δy wrt the initial MF axes
fracture_path_x = a + np.array([sum(x_inc[:i+1]) for i,__ in enumerate(x_inc)]) # Path represents crack tip
fracture_path_y = np.array([sum(y_inc[:i+1]) for i,__ in enumerate(y_inc)])


# Plot all angles in degrees (need to convert from radians)
rad_to_deg = 180./np.pi


'''''Create Folder to save all Simulation State, Figures and Data'''

# The name of the folder needs to uniquely identify the simulation that was run based on the main key parameters.
# Parameters that are important for identifying the simulation:
#

# results = 'view'
results = 'save'


# norm_step = str(dl/a)
norm_length = str(int(x_lim_voids/a)-1)


if voids_distribution_method != 'Deterministic_Staggered':
    quantify_voids_type = 'trueVoidDensityPerMM2'
    quantify_voids_num = str(true_void_density/10**6)
```

```
else:
    quantify_voids_type = 'voidCount'
    quantify_voids_num = str(numberOfPts)


approach_str = 'Approach_{}'.format(approach)


if results == 'save':
    # folder_name = approach_str+ '_' + frameOref + '_' + voids_distribution_method + '_' + quantify_voids_type + quantify_voids_num + '_' + 'normStep{}_normLength{}_'.format(norm_step,norm_length) + 'SPparams{}pts{}r_spNorm'.format(sp_num, r_sp/a)
    folder_name = approach_str+ '_' + frameOref + '_' + voids_distribution_method + '_' + quantify_voids_type + quantify_voids_num + '_' + 'normLength{}_'.format(norm_length) + 'SPparams{}pts{}r_spNorm'.format(sp_num, r_sp/a) + '_V_{}Cs'.format(V/Cs)

else: # In this case we want to save the figures
    # Note: If the folder exists change the folder name slightly so it is obvious which folder is the newer version.

    folder_name='scrap'

# Generate file path
# file_path = Path('C:\\Users\Kids\Desktop\Thesis - Python\Simulation_Stage_2_Results\Figures\\' + folder_name)
file_path = Path('Simulation_Stage_2_Results\Runs\\' + folder_name)

# Create Path if it doesn't exist already - https://stackoverflow.com/questions/273192/how-can-i-safely-create-a-nested-directory
file_path.mkdir(parents=True, exist_ok=True)



'''''Save Current Kernel State'''


file_name = '{}\Kernel.pkl'

# Save Current Session
dill.dump_session(file_name.format(file_path))



'''''Save Current Data'''
# Produce the dataframe to save
column_headders_1 = ['distance_travelled_m','theta_inc_radPERdl','theta_dot_radPERs','theta_rad', 'X_inc_global_mPERdl', 'Y_inc_global_mPERdl', 'X_m', 'Y_m','Vx_mPERs', 'Vy_mPERs', 'fracture_stress_state_MF_only_Pa', 'fracture_stress_state_Pa']
column_headders_2 = ['approach', 'a_m', 'V_mPERs', 'dl_m', 'dt_s', 'sigma_a_Pa', 'startData_index', 'sField_bbox_x', 'sField_bbox_y', 'voids_bbox_x', 'voids_bbox_y', 'main_cr_leadingtip_x', 'main_cr_leadingtip_y', 'true_void_density', 'sigma_w', 'm']



# SIMULATION DATA:
# Create a zip object from two lists
simulation_data_Distance_Direction_Stresses_zip = zip(column_headders_1, [distance, theta_inc, theta_dot, theta,
                                                                          x_inc, y_inc,fracture_path_x,fracture_path_y,
                                                                          Vx, Vy,
                                                                          tuple(map(tuple, fracture_stressState_MF_only.T)), tuple(map(tuple, fracture_stressState.T))
                                                                          ]
                                                      )
# Create a dictionary from zip object
simulation_data_Distance_Direction_Stresses_dict = dict(simulation_data_Distance_Direction_Stresses_zip)
# simulation_data_Distance_Direction_Stresses_dict

#Create dataframe
simulation_data_Distance_Direction_Stresses_df = pd.DataFrame(data=simulation_data_Distance_Direction_Stresses_dict, columns=column_headders_1)
# simulation_data_Distance_Direction_Stresses_df

file_name_Distance_Direction_Stresses = '{}\Distance_Direction_Stresses.pkl'

# SIMULATION PARAMS:
# Create a zip object from two lists
```

```
simulation_Parameters_zip = zip(column_headders_2, [[approach], [a], [V], [dl], [dt], [sigma_a], [startData_index], [sField_bbox_x], [sField_bbox_y], [voids_bbox_x], [voids_bbox_y], [main_cr_leadingtip_x], [main_cr_leadingtip_y],[true_void_density], [sigma_w], [m]])



# Create a dictionary from zip object
simulation_Parameters_dict = dict(simulation_Parameters_zip)
# simulation_Parameters_dict

simulation_Parameters_df = pd.DataFrame(data=simulation_Parameters_dict, columns=column_headders_2)
# simulation_Parameters_df

file_name_Parameters = '{}\Parameters.pkl'
# Save Data in Dataframes in the same folder (save as .pkl file)


simulation_data_Distance_Direction_Stresses_df.to_pickle(file_name_Distance_Direction_Stresses.format(file_path))

simulation_Parameters_df.to_pickle(file_name_Parameters.format(file_path))
```

MICROVOIDS_CLASS.PY

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jul  2 14:28:43 2020

Project: Microvoid and Microcrack Information


@author: Dominic Walker, 450239612, dwal9899

"""


# Import the required modules
#import math
import numpy as np
#from scipy.interpolate import Rbf

from shapely.geometry import Point, Polygon

# Import local module
import stresses
# import stresses_sigma_aII_adjusted_type2 as stresses
import input_parameters



class microvoid:


    '''''Initialise Variables containing Input Parameters'''
    a = input_parameters.a
    rho_s = input_parameters.rho_s
    G = input_parameters.G
    nu = input_parameters.nu
```

```
    V = input_parameters.V
    Cs = input_parameters.Cs

    # There are three components of applied load,
    # sigma_a = [sigma_t, sigma_n, tau_a]
    sigma_a = input_parameters.sigma_a      # Note: This is always in instantaneous MF CRS


    '''''Establish velocity-stress relationship'''
    # Parameters to be used for calculating Crack Velocity from the stress level
    #   Point of jump to upper branch
    ##sigma_k = 1.3*sigma_a
    V_k = 0.01*Cs            #0.2*Cs



    def __init__(self, x_mv, y_mv, microvoid_OpeningStress, microvoid_ID):

        # Initialise location of microvoid
        self.x_mv = x_mv    #np.array(x_mv)
        self.y_mv = y_mv    #np.array(y_mv)
        # self.x_mv = np.array(x_mv)
        # self.y_mv = np.array(y_mv)
        self.microvoid_OpeningStress = microvoid_OpeningStress      # Initialise a variable that will store the microvoid 'critical opening stress'.

        # Initialise minimum stress required for microcrack crack propagation
        self.sigma_k = microvoid_OpeningStress

        # Initilise a numpy array that stores microvoid / microcrack geometry data.
        # The list has the following set out: self.x_vals = np.array([crack_tip_x1],[crack_tip_x2])
        self.x_vals = np.array([[x_mv],
                                [x_mv]])
        self.y_vals = np.array([[y_mv],
                                [y_mv]])


        # Give this instance of a microvoid/microcrack a unique ID number
        self.microvoid_ID = microvoid_ID    #class_name.ClassVariable so that the microvoid_id is independent of the instance


        # Keep track of the state of the microvoid
        self.microvoid_open = False
        # self.microcrack_sprouted = False
        ##self.microVoid_ageSinceOpening = 0
        ##self.microCrack_age = 0             # Age is quantified as the total distance travelled by microcrack since initial opening. On each iteration the main crack displaces a distance, dl, horizontally. This is the same distance that the microcrack moves (but with opposite direction)
        ##self.microCrack_length = 0

        # Keep track of the microcrack crack velocity history (for each crack tip) of the previous timestep only
        self.V_previous = np.array([0,0])          # Assume that the initial state of a microcrack is stationary. This is a numpy array with two elements to account for the two ends at which a microcrack can grow.

        # Keep track of the microcrack crack direction history (for each crack tip) of the previous timestep only
        self.dir_previous = np.array([0., np.pi])   # Assume an initial direction of growth for each possible microcrack crack tip. Late it will be enforced that a microcrack cannot change direction by more than 90 deg between iterations.
        #This will allow each microcrack to travel in different directions initially as well as choose the correct direction vector of the relevant principal plane line in each iteration.



        # Initialise variables that will store the effective microcrack geometry
        self.a_eff = np.nan                         #   Effective microcrack length
        self.inclAng = np.nan                       #   Effective microcrack angle of inclination   - This defines the local x-direction        # NOTE: This is measured relative to the global coordinate system
        self.mid_pt = np.array([np.nan,np.nan])     #   Midpoint of effective microcrack            - This will be the local origin


```

```
98.          # Initialise variables that will record the average stress applied to the microcrack - GLOBAL COORDINATES
99.          self.sigma_xx_av = np.nan
100.         self.sigma_yy_av = np.nan
101.         self.sigma_xy_av = np.nan
102.
103.         # Keep track of when MC starts to repel the MF
104.         ##self.repelling = False
105.
106.
107.
108.
109.     # The function transfers points in one coordinate reference system (CRS1) to points in another CRS (CRS2)
110.     #
111.     # This function will be used for moving between CRSs
112.     # e.g. to get the appropriate (x,y) wrt a microcrack's reference system so that stresses can be calculated.
113.     # e.g.
114.     #
115.     # This function works irrespective of the frame of reference being used (Eulerian/Lagrangian).
116.     # But also, it is useful when going between frames of reference.
117.     #
118.     # NOTE: NEED TO CHECK IF THIS WORKS WITH MATRICES X AND Y
119.     #
120.     # Reference: http://www.inf.ed.ac.uk/teaching/courses/cg/lectures/cg3_2013.pdf - Slide 18
121.     #
122.     # Function arguments:
123.     #   (x,y) is the point (or list of points) that are being converted from one CRS to another
124.     #   (x_0,y_0) is the origin of CRS2 wrt origin of CRS1
125.     #   theta_0 is the angle of inclination of CRS2 wrt CRS1 - this is measured from CRS1's +ve x-axis to CRS2's +ve x-axis
126.     #   i.e. Angle of inclination of CRS2 x-axis wrt CRS1 X-axis. -pi/2 < theta_0 <= pi/2
127.     #
128.     @staticmethod
129.     def change_CRS_geom(x, y, x_0, y_0, theta_0):
130.
131.
132.         # Make sure that the datatype is correct
133.         # if str(type(x)) not in ['numpy.ndarray', "<class 'numpy.ndarray'>"]:
134.         #     x = np.array([x])
135.         #     y = np.array([y])
136.
137.
138.         # Reverse sign of theta_0 (all points must rotate in the opposite way to the axes)
139.         theta = -1*theta_0
140.
141.         # Matrix for Translation and rotation - Note: The rotation matrix is anticlockwise positive (follows normal sign convention)
142.         M = np.array([[np.cos(theta), -1*np.sin(theta)],
143.                       [np.sin(theta), np.cos(theta)],
144.                       ])
145.
146.
147.         # Convert points from global to local
148.         ##x_CRS2, y_CRS2 = np.matmul(M, np.array([x-x_0, y-y_0]))
149.
150.         # Convert points from global to local - writing like this allows for changing CRS of 2d arrays of points
151.         x_CRS2 = M[0,0]*(x-x_0) + M[0,1]*(y-y_0)
152.         y_CRS2 = M[1,0]*(x-x_0) + M[1,1]*(y-y_0)
153.
154.         return x_CRS2, y_CRS2
155.
156.
157.
158.
159.
160.
161.     # The function calculates the major principal stress and rotation to the principal direction at any point (x,y) that is within the
    specified grid of x- and y-values.
162.     #
163.     # Function arguments:
164.     #   -   some point (x,y)
165.     #   -   the simulation frame of refrence
```

```
    #   -   location (main_cr_leadingtip_x, main_cr_leadingtip_y) & direction, dir_n, of motion of the main crack.
    #
    def stress_state(self, frameOref, MF_origin_x, MF_origin_y, dir_n):

        # Points for calculating stresses along microcrack
        x_mc = self.x_mv    #self.x_vals
        y_mc = self.y_mv    #self.y_vals


        '''''Ensure geometry is in instantaneous MF CRS'''
        # If a lagrangian frame of reference is used, the point (x,y) will give the correct stress directly
        # If the eulerian frame of reference is selected, the point (x,y) needs to be moved to an equivalent point (x',y') in the equiv
    alent Lagrangian system in order to calculate the stresses.

        # These points need to be brought into the MF coordinate system from the global coordinate system
        if frameOref == 'Eulerian':
            # Change the CRS from the Global CRS to the MF CRS
            x_mc, y_mc = microvoid.change_CRS_geom(self.x_mv, self.y_mv, MF_origin_x, MF_origin_y, dir_n)




        '''''Calculate Stresses and determine principal direction'''
        # Calculate the stresses at the point/s (x,y)
        ##sigma_xx, sigma_yy, sigma_xy, __, __ = stresses.stress_Yoffe(x_mc, y_mc, microvoid.a, microvoid.sigma_a, microvoid.V, microvo
    id.rho_s, microvoid.G, microvoid.nu)
        ##sigma_xx, sigma_yy, sigma_xy, __, __ = stresses.stress_Griff(x_mc, y_mc, microvoid.a, microvoid.sigma_a, microvoid.nu)

        # Calculate the stresses at the point/s (x,y) - Account for Mixed Mode Loading
        # Note: Here we calculate the stresses from the MF using geometry and stresses in terms of the instantaneous MF CRS
        # Yoffe
        sigma_xx_I, sigma_yy_I, sigma_xy_I, __, __ = stresses.stress_Yoffe(x_mc, y_mc, microvoid.a, microvoid.sigma_a[1], microvoid.V,
    microvoid.rho_s, microvoid.G, microvoid.nu)          #   Stresses from Mode I loading
        # sigma_xx_II, sigma_yy_II, sigma_xy_II, __, __ = stresses.stress_Yoffe_II(x_mc, y_mc, microvoid.a, microvoid.sigma_a[2], micro
    void.V, microvoid.rho_s, microvoid.G, microvoid.nu)    #   Stresses from Mode II loading

        # # Griffith
        # sigma_xx_I, sigma_yy_I, sigma_xy_I, __, __ = stresses.stress_Griff(x_mc, y_mc, microvoid.a, microvoid.sigma_a[1], microvoid.n
    u)          #   Stresses from Mode I loading
        # sigma_xx_II, sigma_yy_II, sigma_xy_II, __, __ = stresses.stress_Griff_II(x_mc, y_mc, microvoid.a, microvoid.sigma_a[2], micro
    void.nu)    #   Stresses from Mode II loading

        # Mean stress acting on MC
        # sigma_xx = np.mean(sigma_xx_I + sigma_xx_II)
        # sigma_yy = np.mean(sigma_yy_I + sigma_yy_II)
        # sigma_xy = np.mean(sigma_xy_I + sigma_xy_II)
        sigma_xx = sigma_xx_I# + sigma_xx_II
        sigma_yy = sigma_yy_I# + sigma_yy_II
        sigma_xy = sigma_xy_I# + sigma_xy_II

        sigma_1, sigma_2, rot_to_principal_dir = stresses.transform2d_ToPrincipal(sigma_xx, sigma_yy, sigma_xy)




        # Calculate the acute angle between the +ve x-axis and the major principal plane.
        #   'rot_to_principal_dir' is the angle through which an element must be rotated to get the principal planes.
        #   An anticlockwise rotation through an angle 'theta' of an element corresponds to a rotation through a negative angle 2*theta
     in mohrs circle.
        #   A clockwise rotation through an angle 'theta' corresponds to a rotation through a positive angle 2*theta in mohrs circle.
        #   A rotation of an element in an anti-clockwise direction through an angle 'theta' results in the plane that corresponded.
        #   When considering the rotation of a horizontal line through some angle theta, sign convention is opposite.
        #       Therefore, an anticlockwise rotation of a horizontal line through an angle theta results in a line with gradient tan(-
    1*theta) (since theta<0)
        #             and, a clockwise rotation of a horizontal line through an angle theta results in a line with gradient tan(-
    1*theta) (since theta>0)


        # Now, the initial orientation of an element has the plane on which sigma_yy acts being parallel with the +ve x-axis.
        # If sigma_yy >= sigma_xx, then theta = rot_to_principal_dir (that is theta as described above)... and theta_I = -1*theta.
```

```
226.         # If sigma_yy < sigma_xx, then theta = rot_to_principal_dir (that is theta as described above)... and theta_I = -
     1*theta - pi/2 OR EQUIVALENTLY, theta_I = -1*theta + pi/2.
227.
228.         # If sigma_xx > sigma_yy, then the angle between the +ve x-axis and principal plane is calculated as:
229.         #   a) if rot_to_principal_dir > 0, theta_I = 90 - rot_to_principal_dir
230.         #   b) if rot_to_principal_dir < 0, theta_I = -1*(90 + rot_to_principal_dir)
231.
232.         if sigma_yy >= sigma_xx:
233.             theta_I = -1*rot_to_principal_dir
234.
235.         # This is the case where sigma_yy < sigma_xx
236.         else:
237.             theta_I = np.arctan(np.tan(-1*rot_to_principal_dir + np.pi/2))
238.
239.
240.         '''''Convert principal direction into Global coordinates - if required'''
241.         # !!! Note: The rotation to a principal direction is measured from the instantaneous MF +ve x axis.
242.         # The actual angle of inclination of the major principal plane should be measured wrt GLOBAL axes so that MCs grow in the corre
     ct direction. (since geometry is always in terms of the GLOBAL axes)
243.         # In a Lagrangian frame of reference, the instantaneous MF axes coincided with the GLOBAL axes.
244.         # In a Eulerian frame of reference, the GLOBAL axes are related to the instantaneous MF axes via dir_n.
245.         if frameOref == 'Eulerian':
246.             # Put theta_I in terms of the Global coordinate +ve x-axis.
247.             theta_I = theta_I + dir_n
248.
249.
250.         return [sigma_1, theta_I]
251.
252.
253.
254.     # Check if the microvoid is open
255.     # This function will only be run on microvoids that are not yet open. It will check if microvoids are now open.
256.     def mv_is_open(self, frameOref, MF_origin_x, MF_origin_y, dir_n):
257.         # Get the stress level at the location of the microcrack
258.         stress_now, __ = self.stress_state(frameOref, MF_origin_x, MF_origin_y, dir_n)
259.
260.         # If the stress level exceeds the 'microvoid_OpeningStress' then set microvoid_open to 'True'. If this is run on a microvoid th
     at is already open, no problems will occur.
261.         if (stress_now >= self.microvoid_OpeningStress):
262.             self.microvoid_open = True
263.
264.
265.
266.     # This function takes in the grid of x-values and y-
     values, major principal stress and direction of major principal plane and timestep.
267.     # The velocity is calculated knowing the stress state and the veocity of the crack tip at the previous timestep.
268.     #
269.     # Function arguments are:
270.     #   -   stress field geometry, XX and YY
271.     #   -   stress field values, sigma_1
272.     #   -   The current location of the crack tips (x1,y1) and (x2,y2)      <-
     - these should be available within the class instance already in the list that records the geometry of the microcrack.
273.     #   -   Time Step
274.     #   -   Crack velocity at previous time step, V_previous (this is because of the way that the crack velocity is related to the stre
     ss level)
275.     #
276.     # The function calculates:
277.     #   -   uses stress_state() to calculate the level of major principal stress and the direcion of the principal plane at the microcr
     ack crack tips
278.     #   -   The crack velocity (speed) at the microcrack crack tips
279.     # The function output:
280.     #   -   New (x1,y1) and (x2,y2) values given to list/s containing microcrack geometry
281.     def crack_velocity(self, frameOref, MF_origin_x, MF_origin_y, dir_n, sField_bbox_coords):
282.
283.
284.         # Initialise lists to store the velocity and direction of motion of microcrack crack tips.
285.         # mc_V = np.array([np.nan, np.nan])
286.         mc_dir = np.array([np.nan, np.nan])
287.
```

```
        # Calculate average stress on the MC and the principal direction.
        mc_AppliedStress, mc_MajPrincDir = self.stress_state(frameOref, MF_origin_x, MF_origin_y, dir_n)

        # Calculate the velocity of the MC tips
        #   CASE 1: Stress isn't high enough to move crack tip. i.e. mc_tipStress<sigma_k.
        #   CASE 2: Stress IS high enough to move crack tip. i.e. mc_tipStress>=sigma_k.
        if mc_AppliedStress < self.sigma_k:
            mc_V = np.array([0., 0.])

        elif mc_AppliedStress >= self.sigma_k:
            mc_V = np.array([microvoid.V_k, microvoid.V_k])

        else:
            raise Exception('Check crack_velocity() function. Microcrack crack tip velocity assignment not working properely.')



        # Each open void is associated with two microcrack crack tips.
        # Get the velocity (speed + direction) for each crack tip.

        for i, __ in enumerate(mc_dir):


            # Calculate the direction of motion of the crack.
            # NOTE: THE DIRECTION OF MOTION IS MEASURED FROM THE GLOBAL +VE X-AXIS. This is done because all the geometry is in terms of the GLOBAL axes.
            #  Even if the crack doesn't move we still want to know what direction it would move in case it would move in the next iteration
            #  Note: mc_tip_MajPrincDir is the angle of inclination to the +ve x axis for the major(?) principal plane.
            # There are two cases:
            #  CASE 1: The angle of the principal plane is within +/- 90 degrees of the last direction of motion of the microcrack - new direction of motion is the same as that given by mc_tip_MajPrincDir
            #  CASE 2: The angle of the principal plane is within +/- 90 degrees of the last direction of motion of the microcrack - new direction of motion is the same as that given by mc_tip_MajPrincDir -/+ 180 deg
            # Note: We need to ensure that the angles always remain within +/- 180 degrees
            # mc_tip_MajPrincDir will always be given as an angle between +/- 90 degrees

            # Within each case above there are two more cases:
            # Case a: mc_tip_MajPrincDir and previous_direction have the same sign and are within 90 deg of each other
            # Case b: mc_tip_MajPrincDir and previous_direction have opposite sign and are within 90 deg of each other
            # Case c: mc_tip_MajPrincDir and previous_direction have the same sign and are NOT within 90 deg of each other
            # Case d: mc_tip_MajPrincDir and previous_direction have opposite sign and are NOT within 90 deg of each other

            # SIMPLE SOLUTION: Use the angle of the major principal plane and the previous angle of the major principal plane to produce unit vectors.
            #                  Calculate the angle between the unit vectors.
            #                  If the angle between the vectors is <= 90 deg, assign the new direction equal to mc_tip_MajPrincDir,
            #                  Elif the angle between the vectors is > 90 deg, multiply the vector by -1 and get the new angle

            # Make unit vectors using direction of:
            # a) unit vector pointing in previous direction of motion
            m = np.array([np.cos(self.dir_previous[i]), np.sin(self.dir_previous[i])])

            # b) current major principal plane - unit vector points along the plane in the direction that is within quadrant 1 or quadrant 4.
            n = np.array([np.cos(mc_MajPrincDir),np.sin(mc_MajPrincDir)])

            # Define a unit vector that points along the +ve x-axis
            x_unitVec = np.array([1,0])

            # Calculate angle between the unit vectors (in radians)
            ang_bw = np.arccos(np.dot(m,n)/(np.linalg.norm(m)*np.linalg.norm(n)))

            # If the angle between the vectors is <= 90 deg, assign the new direction equal to mc_tip_MajPrincDir,
            if ang_bw <= np.pi/2:
                mc_dir[i] = mc_MajPrincDir

```

```
            # If the angle between the vectors is greater than 90 deg, multiply n by -1 and get the direction of the vector wrt the +ve x-axis.
            else:
                n = -1*n

                # Calculate the direction of motion. If the y-component of vector n is negative, then multiply the angle by -1.
                if n[1] >= 0:
                    mc_dir[i] = np.arccos(np.dot(x_unitVec,n)/(np.linalg.norm(x_unitVec)*np.linalg.norm(n)))
                else:
                    mc_dir[i] = -1*np.arccos(np.dot(x_unitVec,n)/(np.linalg.norm(x_unitVec)*np.linalg.norm(n)))


        return [mc_V, mc_dir]



    # This function checks if new points need to be added to the arrays containing the geomentry of the each side of the microcrack.
    # This is done by running the next_XYpoint() function twice; once for each side of the microcrack.
    # If non-None-type objects are returned then the point is added to the geometry arrays

    # Note. If both points to add is the same as the location of the microcrack (i.e. microcrack is open, but crack doesn't grow), then don't add the points.


    # This function:
    #           1. Calculates the new (x,y) values for the microcrack crack tips
    #               a) If microvoid is not open or if the microvoid is open but a microcrack hasn't sprouted yet and doesn't sprout this iteration, RETURN NONE
    #               b) If a microcrack has sprouted at the microvoid then add the geometry points to the geometry arrays - do this even if the velocity is 0 for one or both of the microcrack crack tips.
    #           2. Updates the V_previous and dir_previous arrays with the new V and dir of each microcrack crack tip, respectively.
    #               a) If microvoid is not open or if the microvoid is open but a microcrack hasn't sprouted yet and doesn't sprout this iteration, RETURN NONE
    #               b) If a microcrack has sprouted at the microvoid then add the new V and dir values to the V_previous and dir_previous arrays - do this even if the velocity is 0 for one or both of the microcrack crack tips.
    #
    # Note: Geometry values are appropriate to the current coordinate system only!
    #
    # Function arguments:
    #   -   The current location of the crack tips (x1,y1) and (x2,y2)
    #   -   The level of major principal stress and the direcion of the principal plane at the microcrack crack tips
    #   -   Time Step
    #
    # This function calculates:
    #   -   The crack velocity (speed) at the microcrack crack tips
    #     !!If crack velocity is 0 (ZERO), then update the class instance of V_previous to be 0 ONLY and nothing else.    (What if one side of crack tip has V=0, but the other crack tip does not?)
    #   If V NOT 0, then calculate
    #   -   The displacement of the crack tip/s in terms of (r',theta') => (distance, direction) with origin at location of microcrack crack tip at the previous timestep.
    #   -   Converts (r',theta') to new (x1,y1) and (x2,y2) values
    #   -   Store

    # Function Output:
    #   -   New (x1,y1) and (x2,y2) values given to list/s containing microcrack geometry
    def next_XYpoint(self, dt, frameOref, MF_origin_x, MF_origin_y, dir_n, sField_bbox_coords):#XX,YY,sigma_1, rot_to_principal_dir,


        # Calculate the crack velocity and direction of motion for current geometry and stress state.
        mc_V, mc_dir = self.crack_velocity(frameOref, MF_origin_x, MF_origin_y, dir_n, sField_bbox_coords)

        # If a microvoid is not opened, then exit function.
        if self.microvoid_open == False:
            return None


        # If we get to 'else' that means,
        #       a) the microvoid is open, AND
```

```
        #       b) a microcrack has sprouted before or is sprouting now (though, it is not necessarily going to grow on this iteration)

        # In this situation,
        #       a) obtain new geometry values for each crack tip
        #       b) append new geometry values to the x_vals and y_vals arrays
        #       c) update the V_previous and dir_previous lists
        # Note: Because of the way that the information is stored, we need to return something if the microvoid is opened but the micro crack doesnt grow.
        # So just return the previous location the microcrack crack tip. This will be returned automatically if velocity is 0.
        else:
            # Make a note that a microcrack is now growing.
            ###self.microcrack_sprouted = True

            '''''SLOW WAY'''
            # # Initialise matrices that will temporarily store the new values of x and y before appending them to the overall x_vals, y_vals geometry arrays.
            # x_new = np.array([[np.nan],[np.nan]])
            # y_new = np.array([[np.nan],[np.nan]])


            # # For each microcrack crack tip do steps (a)-(c) above
            # for i, __ in enumerate(mc_V):

            #     # Obtain new point (x,y)
            #     #   Distance travelled, r = V_current*dt = mc_V[i]*dt
            #     #   direction of motion, theta = mc_dir[i] (rad)
            #     #   (x_new, y_new) = (x_old +r*cos(theta),y_old +r*sin(theta))
            #     x_new[i] = self.x_vals[i,-1] + (mc_V[i]*dt)*np.cos(mc_dir[i])
            #     y_new[i] = self.y_vals[i,-1] + (mc_V[i]*dt)*np.sin(mc_dir[i])



            #     # update the V_previous and dir_previous lists
            #     self.V_previous[i] = mc_V[i]
            #     self.dir_previous[i] = mc_dir[i]

            # self.x_vals = np.append(self.x_vals, x_new, axis=1)
            # self.y_vals = np.append(self.y_vals, y_new, axis=1)


            'FAST WAY'
            # Obtain new point (x,y)
            #   Distance travelled, r = V_current*dt = mc_V[i]*dt
            #   direction of motion, theta = mc_dir[i] (rad)
            #   (x_new, y_new) = (x_old +r*cos(theta),y_old +r*sin(theta))
            x_new = self.x_vals[:,-1] + (mc_V*dt)*np.cos(mc_dir)
            y_new = self.y_vals[:,-1] + (mc_V*dt)*np.sin(mc_dir)

            # update the V_previous and dir_previous lists
            ###self.V_previous = mc_V
            self.dir_previous = mc_dir


            # Append new geometry values to the x_vals and y_vals arrays
            self.x_vals = np.append(self.x_vals, np.array([[x_new[0]],[x_new[1]]]), axis=1)
            self.y_vals = np.append(self.y_vals, np.array([[y_new[0]],[y_new[1]]]), axis=1)

            # self.x_vals = np.append(self.x_vals, x_new, axis=1)
            # self.y_vals = np.append(self.y_vals, x_new, axis=1)



    # This function checks if a microcrack is within the stress field grid or not.
    # The stress field grid is defined by the corner points.
    # It is important to note that the rectangular bounding box can have any orientation in an Eulerian flow field Specification,
```

```
    # while, in a Lagrangian flow field specification the orientation of the box is fixed.
    # The code required for the Eulerian case is essentially a generalised version of the Lagrangian flow field where by the rectangle can take any orientation.
    #
    #
    # Function arguments:
    #   -   Location of the microvoid
    #   -   Arrays defining the perimeter points of the rectangular stress field bounding box
    #   -   the flow field specification being used.
    #
    # Function Output:
    #   -   Boolean indicating if the microvoid is within the stress field grid.
    # @staticmethod
    # def isin_sGrid(x, y, sField_bbox_coords):

    #     # Location of microvoid - as a Point object (Geo-coordinates)
    #     mv_geo = Point(x, y)

    #     # Polygon defined by the stress field bounding box.
    #     sField_bbox_geo = Polygon(sField_bbox_coords)

    #     return mv_geo.within(sField_bbox_geo)

    def isin_sGrid(self,sField_bbox_x):

        if (self.x_mv > np.min(sField_bbox_x)) & (self.x_mv < np.max(sField_bbox_x)):
            return True
        else:
            return False




    # The function calculates effective geometry of a microcrack for calculating stresses applied back onto the main fracture.

    # The effective microcrack is the line connecting the end points of the microcrack
    # The origin of the local coordinate is the midpoint of the microcrack crack tips
    # The angle of inclination is determined usin gthe microcrack crack tips.

    # Function arguments:
    #
    #
    def mc_effectiveGeom(self):

        # The half of effective microcrack length
        self.a_eff = 0.5*np.sqrt((self.x_vals[0,-1] - self.x_vals[1,-1])**2 + (self.y_vals[0,-1] - self.y_vals[1,-1])**2)
        # self.a_eff = 0.5*abs(self.x_vals[0,-1] - self.x_vals[1,-1]) # This approximation is reasonable when approach 1 is being used1

        # Get the angle of inclination of the local MC x-axis wrt global X-axis (i.e. the geometry in which all geometry is defined)
        #   In Eulerian frame, inclAng is the angle bw the initial MF +ve x-axis and the MC +ve x-axis (because all geometry is in terms of initial MF axis position)
        #   In Lagrangian frame, inclAng is the angle bw the instantaneous MF +ve x-axis and the MC +ve x-axis
        if self.x_vals[0,-1] != self.x_vals[1,-1]:
            self.inclAng = np.arctan((self.y_vals[0,-1] - self.y_vals[1,-1])/(self.x_vals[0,-1] - self.x_vals[1,-1])) + np.pi # Increase inclAng angle by pi so that the MC local +ve x-axis is point back towards MF.

            # x_fit = np.array([self.x_vals[0,::-1], self.x_vals[1]]).flatten()
            # y_fit = np.array([self.y_vals[0,::-1], self.y_vals[1]]).flatten()
            # m, b = np.polyfit(x_fit,y_fit,1)
            # self.inclAng = np.arctan(m) + np.pi

        else:
            self.inclAng = np.pi/2 # if the x_vals are the same, then the MC is vertical

        # Determine the location of  the origin of the local coordinate system in terms of the global coordinate system. The origin is set at the midpoint of the effective microcrack.
        self.mid_pt = 0.5*np.array([self.x_vals[0,-1] + self.x_vals[1,-1], self.y_vals[0,-1] + self.y_vals[1,-1]])

```

```
540.
541.
542.
543.
544.     # # The function transfers points in the global coordinates to points in local coordinates belonging to a MICROCRACK
545.     # #
546.     # # This function will be used to get the appropriate (x,y) wrt a microcrack's reference system so that
547.     # # stresses can be calculated.
548.     # #
549.     # # This function works irrespective of the frame of reference being used (Eulerian/Lagrangian).
550.     # #
551.     # # NOTE: NEED TO CHECK IF THIS WORKS WITH MATRICES X AND Y
552.     # #
553.     # # Reference: http://www.inf.ed.ac.uk/teaching/courses/cg/lectures/cg3_2013.pdf - Slide 18
554.     # #
555.     # # Function arguments:
556.     # #
557.     # #
558.     # def global_to_local_geom(self, x, y):
559.
560.     #     # Angle of inclination of local x-axis wrt global X-axis. -pi/2 < theta_local <= pi/2
561.     #     theta_local = self.inclAng
562.
563.     #     # Angle of inclination of line from global origin to local origin (wrt global X-axis)
564.     #     beta = np.arctan(self.mid_pt[1]/self.mid_pt[0])
565.
566.     #     # Angle required for translating the *rotated* global coordinates to the local coordinates
567.     #     alpha = beta - theta_local
568.
569.     #     # Displacements
570.     #     R = np.sqrt(self.mid_pt[0]**2 + self.mid_pt[1]**2)
571.     #     dx = -1*R*np.cos(alpha)
572.     #     dy = -1*R*np.sin(alpha)
573.
574.     #     # Matrix for Translation and rotation
575.     #     M = np.array([[np.cos(theta_local), -1*np.sin(theta_local), dx],
576.     #                   [np.sin(theta_local), np.cos(theta_local), dy],
577.     #                   [0, 0, 1]
578.     #                   ])
579.
580.
581.     #     # Convert points from global to local
582.     #     x_local, y_local, __ = np.matmul(M, np.array([[x],[y],[1]]))
583.
584.
585.     #     return x_local, y_local
586.
587.
588.
589.
590.
591.
592.     # # The function transfers points in the local (MC) coordinates to global coordinates.
593.
594.     # # Function arguments:
595.     # #
596.     # #
597.     # def local_to_global_geom(self, x, y):
598.
599.     #     # Angle of inclination of local x-axis wrt global X-axis. -pi/2 < theta_local <= pi/2
600.     #     theta_local = self.inclAng
601.
602.     #     # Angle of inclination of line from global origin to local origin (wrt global X-axis)
603.     #     beta = np.arctan(self.mid_pt[1]/self.mid_pt[0])
604.
605.     #     # Angle required for translating the *rotated* global coordinates to the local coordinates
606.     #     alpha = beta - theta_local
607.
608.     #     # Displacements
```

```
609.     #     R = np.sqrt(self.mid_pt[0]**2 + self.mid_pt[1]**2)
610.     #     dx = -1*R*np.cos(alpha)
611.     #     dy = -1*R*np.sin(alpha)
612.
613.     #     # Matrix for Translation and rotation
614.     #     M = np.array([[np.cos(theta_local), -1*np.sin(theta_local), dx],
615.     #                   [np.sin(theta_local), np.cos(theta_local), dy],
616.     #                   [0, 0, 1]
617.     #                   ])
618.
619.     #     M_inv = np.linalg.inv(M)
620.
621.     #     # Convert points from global to local
622.     #     x_global, y_global, __ = np.matmul(M_inv, np.array([[x],[y],[1]]))
623.
624.
625.     #     return x_global, y_global
626.
627.
628.
629.
630.
631.     # The function transfers stresses in the LOCAL coordinates to stresses in MF coordinates
632.     # To transfer stresses from local coordinates to global coordinates, rotate the stress element through an angle theta_local.
633.     # Note: The sign of theta_local is EXTREMELY important since the cartesian plane and mohr's circle have different sign conventions
    for rotation.
634.     #
635.     # IMPORTANT: Global Main Fracture (MF) Stresses ==> Stresses in terms of the MF coordinate system.
636.     #               In the Lagrangian frame of reference, the Global coordinates system and the MF coordinate system coincide.
637.     #               In the Eulerian frame of reference, the Global coordinate system and the MF coordinate system are different.
638.     # When we talk about stresses, the global stresses specificially refer to the MF coordinate system.
639.     # This DOES NOT affect the magnitude of the principal stresses, BUT it DOES affect the direction of the major principal plane (as t
    he direction is measured from the MF axis)
640.     #
641.     #
642.     #
643.     #
644.     # Function arguments:
645.     #    sigma_xx, sigma_yy, sigma_xy are in local MC CRS -- need to convert to instantaneous MF axes
646.     #
647.     #
648.     def local_to_global_MF_stresses(self,sigma_xx, sigma_yy, sigma_xy, dir_n, frameOref):
649.
650.
651.         if frameOref == 'Eulerian':
652.             # The difference, self.inclAng - dir_n, gives the angle of inclination of the local MC axes measured from the instantaneous
     MF axes.
653.             # Note: The angles are relative to the initial orientation of the MF axes.
654.             theta_local = self.inclAng - dir_n  # Eulerian
655.
656.             # In the Lagrangian frame, all geometries in the MV & MC field are in terms of the orientation of the instantaneous MF fiel
    d.
657.         elif frameOref == 'Lagrangian':
658.             theta_local = self.inclAng          # Lagrangian
659.
660.         else:
661.             print('flow field specification needed')
662.
663.
664.
665.         # Rotate the stress element in the opposite direction to the angle of inclination of the local x-axis wrt global X-axis
666.         theta_rotElement = -1*theta_local
667.
668.         '''''SIGN CONVENTION IS anti-CLOCKWISE POSITIVE - BUT THIS IS ACCOUNTED FOR ALREADY IN DERIVATION (double check this)'''
669.         # Adjustment for converting from anticlockwise positive to clockwise positive
670.         #   For anti-clockwise rotation of stress element theta_rotElement < 0
671.         #   For clockwise rotation of stress element theta_rotElement > 0         (Mohr's Circle is clockwise positive)
672.         ###theta_rotElement = theta_rotElement #-1*theta_rotElement
673.
```

```
674.         transformation_array = np.array([[np.cos(theta_rotElement)**2, np.sin(theta_rotElement)**2, 2*np.sin(theta_rotElement)*np.cos(t
     heta_rotElement)],
675.                                          [np.sin(theta_rotElement)**2, np.cos(theta_rotElement)**2, -
     2*np.sin(theta_rotElement)*np.cos(theta_rotElement)],
676.                                          [-
     1*np.sin(theta_rotElement)*np.cos(theta_rotElement), np.sin(theta_rotElement)*np.cos(theta_rotElement), (np.cos(theta_rotElement)**2 -
     np.sin(theta_rotElement)**2)]
677.                                          ])
678.
679.         # Option 1 - this is only for individual points
680.         #sigma_xx_global, sigma_yy_global, sigma_xy_global = np.matmul(transformation_array,np.array(sigma_xx, sigma_yy, sigma_xy))
681.
682.         # Option 2 -
683.         #sigma_xx_global = sigma_xx*np.cos(theta_rotElement)**2 + sigma_yy*np.sin(theta_rotElement)**2 + 2*sigma_xy*np.sin(theta_rotEle
     ment)*np.cos(theta_rotElement)
684.         #sigma_yy_global = sigma_xx*np.sin(theta_rotElement)**2 + sigma_yy*np.cos(theta_rotElement)**2 -
     2*sigma_xy*np.sin(theta_rotElement)*np.cos(theta_rotElement)
685.         #sigma_xy_global = -
     1*sigma_xx*np.sin(theta_rotElement)*np.cos(theta_rotElement) + sigma_yy*np.sin(theta_rotElement)*np.cos(theta_rotElement) + sigma_xy*(n
     p.cos(theta_rotElement)**2 - np.sin(theta_rotElement)**2)
686.
687.         # Option 3 - option 2 but neater (maybe not as fast as Option 2?)
688.         sigma_xx_global = sigma_xx*transformation_array[0,0] + sigma_yy*transformation_array[0,1] + sigma_xy*transformation_array[0,2]
689.         sigma_yy_global = sigma_xx*transformation_array[1,0] + sigma_yy*transformation_array[1,1] + sigma_xy*transformation_array[1,2]
690.         sigma_xy_global = sigma_xx*transformation_array[2,0] + sigma_yy*transformation_array[2,1] + sigma_xy*transformation_array[2,2]
691.
692.
693.
694.         return sigma_xx_global, sigma_yy_global, sigma_xy_global
695.
696.
697.
698.
699.
700.
701.     # The function transfers stresses in the GLOBAL coordinates to stresses in LOCAL coordinates
702.     # To transfer stresses from local coordinates to global coordinates, rotate the stress element through an angle theta_local.
703.     # Note: The sign of theta_local is EXTREMELY important since the cartesian plane and mohr's circle have different sign conventions
     for rotation.
704.     #
705.     # Note: inclAng is measured relative to the global axes, but dir_n is measured relative to the initial position of the MF +ve x-
     axis.
706.     #       In a Eulerian frame, the Global axes and the initial MF axes coincide.
707.     #       In a Lagrangian frame, the Global axes follow the instantaneous position of the MF axes.
708.     #           Therefore, inclAng is itself the angle between the axes
709.     #       When a Lagrangian frame of reference is used
710.
711.     # Function arguments:
712.     #
713.     #
714.     def global_MF_to_local_stresses(self, sigma_xx, sigma_yy, sigma_xy, dir_n, frameOref):
715.
716.         # Angle of inclination of local x-axis wrt Main Fracture X-axis. -pi/2 < theta_local <= pi/2
717.
718.         if frameOref == 'Eulerian':
719.             theta_local = self.inclAng - dir_n  # Eulerian
720.         elif frameOref == 'Lagrangian':
721.             theta_local = self.inclAng          # Lagrangian
722.
723.
724.         # Rotate the stress element in the SAME direction as the angle of inclination of the LOCAL x-axis measured from the GLOBAL X-
     axis
725.         theta_rotElement = theta_local
726.
727.
728.         '''''SIGN CONVENTION IS anti-CLOCKWISE POSITIVE'''
```

```
        # Adjustment for converting from anticlockwise positive to clockwise positive
        #   For anti-clockwise rotation of stress element theta_rotElement < 0
        #   For clockwise rotation of stress element theta_rotElement > 0         (Mohr's Circle is clockwise positive)
        ###theta_rotElement = theta_rotElement#-1*theta_rotElement

        transformation_array = np.array([[np.cos(theta_rotElement)**2, np.sin(theta_rotElement)**2, 2*np.sin(theta_rotElement)*np.cos(theta_rotElement)],
                                         [np.sin(theta_rotElement)**2, np.cos(theta_rotElement)**2, -2*np.sin(theta_rotElement)*np.cos(theta_rotElement)],
                                         [-1*np.sin(theta_rotElement)*np.cos(theta_rotElement), np.sin(theta_rotElement)*np.cos(theta_rotElement), np.cos(theta_rotElement)**2 - np.sin(theta_rotElement)**2]
                                         ])

        #sigma_xx_local, sigma_yy_local, sigma_xy_local = np.matmul(transformation_array,np.array([sigma_xx, sigma_yy, sigma_xy]))


        sigma_xx_local = sigma_xx*transformation_array[0,0] + sigma_yy*transformation_array[0,1] + sigma_xy*transformation_array[0,2]
        sigma_yy_local = sigma_xx*transformation_array[1,0] + sigma_yy*transformation_array[1,1] + sigma_xy*transformation_array[1,2]
        sigma_xy_local = sigma_xx*transformation_array[2,0] + sigma_yy*transformation_array[2,1] + sigma_xy*transformation_array[2,2]


        return sigma_xx_local, sigma_yy_local, sigma_xy_local









    # The function transfers stresses in the GLOBAL coordinates to stresses in LOCAL coordinates
    # To transfer stresses from local coordinates to global coordinates, rotate the stress element through an angle theta_local.
    # Note: The sign of theta_local is EXTREMELY important since the cartesian plane and mohr's circle have different sign conventions for rotation.
    #
    # Note: inclAng is measured relative to the global axes, but dir_n is measured relative to the initial position of the MF +ve x-axis.
    #       In a Eulerian frame, the Global axes and the initial MF axes coincide.
    #       In a Lagrangian frame, the Global axes follow the instantaneous position of the MF axes.
    #           Therefore, inclAng is itself the angle between the axes
    #       When a Lagrangian frame of reference is used

    # Function arguments:
    #
    @staticmethod
    def global_MF_to_local_MF_stresses(sigma_xx, sigma_yy, sigma_xy, dir_net):

        # Angle of inclination of local x-axis wrt Main Fracture X-axis. -pi/2 < theta_local <= pi/2
        # Rotate the stress element in the SAME direction as the angle of inclination of the LOCAL x-axis measured from the GLOBAL X-axis
        ###theta_rotElement = dir_net


        '''''SIGN CONVENTION IS anti-CLOCKWISE POSITIVE - BUT DERIVATION ACCOUNTS FOR THIS - USING AN ANGLE THETA IN THE FORMULA ==> ROTATING ANTICLOCKWISE IN THAT DIRECTION'''
        # # Adjustment for converting from anticlockwise positive to clockwise positive
        # #   For anti-clockwise rotation of stress element theta_rotElement < 0
        # #   For clockwise rotation of stress element theta_rotElement > 0         (Mohr's Circle is clockwise positive)
        # theta_rotElement = theta_rotElement #-1*theta_rotElement

        # transformation_array = np.array([[np.cos(theta_rotElement)**2, np.sin(theta_rotElement)**2, 2*np.sin(theta_rotElement)*np.cos(theta_rotElement)],
        #                                  [np.sin(theta_rotElement)**2, np.cos(theta_rotElement)**2, -2*np.sin(theta_rotElement)*np.cos(theta_rotElement)],
        #                                  [-1*np.sin(theta_rotElement)*np.cos(theta_rotElement), np.sin(theta_rotElement)*np.cos(theta_rotElement), np.cos(theta_rotElement)**2 - np.sin(theta_rotElement)**2]
        #                                  ])
```

```
785.
786.         # #sigma_xx_local, sigma_yy_local, sigma_xy_local = np.matmul(transformation_array,np.array([sigma_xx, sigma_yy, sigma_xy]))
787.
788.
789.         # sigma_xx_local = sigma_xx*transformation_array[0,0] + sigma_yy*transformation_array[0,1] + sigma_xy*transformation_array[0,2]
790.         # sigma_yy_local = sigma_xx*transformation_array[1,0] + sigma_yy*transformation_array[1,1] + sigma_xy*transformation_array[1,2]
791.         # sigma_xy_local = sigma_xx*transformation_array[2,0] + sigma_yy*transformation_array[2,1] + sigma_xy*transformation_array[2,2]
792.
793.         sigma_xx_local = 0.5*(sigma_xx + sigma_yy) + 0.5*(sigma_xx - sigma_yy)*np.cos(2*dir_net) + sigma_xy*np.sin(2*dir_net)
794.         sigma_yy_local = 0.5*(sigma_xx + sigma_yy) - 0.5*(sigma_xx - sigma_yy)*np.cos(2*dir_net) - sigma_xy*np.sin(2*dir_net)
795.         sigma_xy_local = -0.5*(sigma_xx - sigma_yy)*np.sin(2*dir_net) + sigma_xy*np.cos(2*dir_net)
796.
797.
798.
799.
800.         return sigma_xx_local, sigma_yy_local, sigma_xy_local
801.
802.
803.
804.
805.
806.
807.     # The function calculates the average stresses that should be applied to the microcrack that experiences the main fracture ONLY.
808.     # All stresses and geometries are in GLOBAL COORDINATES
809.     #
810.     # Note: When an Eulerian Frame Of Reference is used the microcrack geometry needs to be transposed into the MF coordinate system (f
    rom the global coordinate system)
811.     #
812.     #
813.     # Function arguments:
814.     # (x,y) points over which stresses will be calculated
815.     # info required for calcuating stresses at that point (using Yoffe)
816.     #
817.     def mc_stress_applied(self, frameOref, MF_origin_x, MF_origin_y, dir_n):
818.
819.         # Points for calculating stresses along microcrack
820.         x_mc = self.x_vals
821.         y_mc = self.y_vals
822.
823.         # These points need to be brought into the MF coordinate system from the global coordinate system
824.         if frameOref == 'Eulerian':
825.             # Change the CRS from the Global CRS to the MF CRS
826.             x_mc, y_mc = microvoid.change_CRS_geom(self.x_vals, self.y_vals, MF_origin_x, MF_origin_y, dir_n)
827.
828.         # Calculate the stresses at the point/s (x,y)
829.         ##sigma_xx, sigma_yy, sigma_xy, __, __ = stresses.stress_Yoffe(x_mc, y_mc, microvoid.a, microvoid.sigma_a, microvoid.V, microvo
    id.rho_s, microvoid.G, microvoid.nu)
830.         ##sigma_xx, sigma_yy, sigma_xy, __, __ = stresses.stress_Griff(x_mc, y_mc, microvoid.a, microvoid.sigma_a, microvoid.nu)
831.
832.
833.         # Calculate the stresses at the point/s (x,y) - Account for Mixed Mode Loading
834.         # Note: Here we calculate the stresses from the MF using geometry and stresses in terms of the instantaneous MF CRS
835.         # Yoffe
836.         sigma_xx_I, sigma_yy_I, sigma_xy_I, __, __ = stresses.stress_Yoffe(x_mc, y_mc, microvoid.a, microvoid.sigma_a[1], microvoid.V,
    microvoid.rho_s, microvoid.G, microvoid.nu)          #   Stresses from Mode I loading
837.         # sigma_xx_II, sigma_yy_II, sigma_xy_II, __, __ = stresses.stress_Yoffe_II(x_mc, y_mc, microvoid.a, microvoid.sigma_a[2], micro
    void.V, microvoid.rho_s, microvoid.G, microvoid.nu)    #   Stresses from Mode II loading
838.
839.         # # Griffith
840.         # sigma_xx_I, sigma_yy_I, sigma_xy_I, __, __ = stresses.stress_Griff(x_mc, y_mc, microvoid.a, microvoid.sigma_a[1], microvoid.n
    u)          #   Stresses from Mode I loading
841.         # sigma_xx_II, sigma_yy_II, sigma_xy_II, __, __ = stresses.stress_Griff_II(x_mc, y_mc, microvoid.a, microvoid.sigma_a[2], micro
    void.nu)    #   Stresses from Mode II loading
842.
843.         # Overall Stresses on the point (x,y)
844.         sigma_xx = sigma_xx_I# + sigma_xx_II
```

```
        sigma_yy = sigma_yy_I# + sigma_yy_II
        sigma_xy = sigma_xy_I #+ sigma_xy_II


        # Calculate the mean stress for each component
        # It is assumed that the stress is applied uniformly on the microcrack
        self.sigma_xx_av = np.mean(sigma_xx)
        self.sigma_yy_av = np.mean(sigma_yy)
        self.sigma_xy_av = np.mean(sigma_xy)

        # Note: The above stresses are in terms of the instantaneous MF axes



    # This function
    #

    # Function arguments:
    #
    #
    #
    def interaction(self, x, y, dir_n, frameOref):

        # Convert points where stresses need to be calculated into LOCAL COORDINATES
        ##x_local, y_local = self.global_to_local_geom(x, y)
        # OR ALTERNATIVELY,
        # Change coordinate reference system of stress point (x,y) from global CRS to local MC CRS
        # Note: self.inclAng is the
        x_local, y_local = microvoid.change_CRS_geom(x,y, self.mid_pt[0], self.mid_pt[1], self.inclAng)         # WHAT ABOUT FRAME OF R
EFERNECE HERE?



        # Convert applied stresses from Global to LOCAL COORDINATES - i.e. Convert from Instantaneous MF to local MC CRS
        sigma_xx_av_local, sigma_yy_av_local, sigma_xy_av_local = self.global_MF_to_local_stresses(self.sigma_xx_av, self.sigma_yy_av,
self.sigma_xy_av, dir_n, frameOref)


        # Calculate (rectangular) stresses in LOCAL COORDINATES at relevant points due to loads applied to microcrack

        # Note if sigma_yy_av_local <= 0, then do not include the contribution of this microcrack (at least for mode I cracking)
        if sigma_yy_av_local >= 0.: #(sigma_yy_av_local > 0.) & (sigma_xy_av_local != 0.): # Consider crack mode I and II

            # Calculate stresses resulting from Mode I loading - Griffith (Williams) Stress Soln
            sigma_xx_interaction_I, sigma_yy_interaction_I, sigma_xy_interaction_I, __, __ = stresses.stress_Griff_MC(x_local, y_local,
 self.a_eff, sigma_yy_av_local, microvoid.nu)
            sigma_xx_interaction_II, sigma_yy_interaction_II, sigma_xy_interaction_II, __, __ = stresses.stress_Griff_II_MC(x_local, y_
local, self.a_eff, sigma_xy_av_local, microvoid.nu)

            # sigma_xx_interaction_I, sigma_yy_interaction_I, sigma_xy_interaction_I, __, __ = stresses.stress_Yoffe_MC(x_local, y_loca
l, self.a_eff, sigma_yy_av_local, self.V_k, microvoid.rho_s, microvoid.G, microvoid.nu)
            # sigma_xx_interaction_II, sigma_yy_interaction_II, sigma_xy_interaction_II, __, __ = stresses.stress_Yoffe_II_MC(x_local,
y_local, self.a_eff, sigma_xy_av_local, self.V_k, microvoid.rho_s, microvoid.G, microvoid.nu)


            # Use superposition for stresses generated by two different crack propagation modes
            sigma_xx_interaction = sigma_xx_interaction_I + sigma_xx_interaction_II
            sigma_yy_interaction = sigma_yy_interaction_I + sigma_yy_interaction_II - sigma_yy_av_local # Correct the stresses so that
we only get the stress increment from the crack (need to remove sigma_infty)
            sigma_xy_interaction = sigma_xy_interaction_I + sigma_xy_interaction_II - sigma_xy_av_local # Correct the stresses so that
we only get the stress increment from the crack (need to remove sigma_infty)


            # Correct the stresses so that we only get the stress increment from the crack (need to remove sigma_infty)
            # sigma_yy_interaction = sigma_yy_interaction - sigma_yy_av_local
            # sigma_xy_interaction = sigma_xy_interaction - sigma_xy_av_local


```

```
906.         else: #elif (sigma_yy_av_local < 0.) & (sigma_xy_av_local != 0.):  # Consider crack mode II (only)
907.
908.             sigma_xx_interaction_I, sigma_yy_interaction_I, sigma_xy_interaction_I = 0., 0., 0.
909.             sigma_xx_interaction_II, sigma_yy_interaction_II, sigma_xy_interaction_II, __, __ = stresses.stress_Griff_II_MC(x_local, y_
    local, self.a_eff, sigma_xy_av_local, microvoid.nu)
910.             # sigma_xx_interaction_II, sigma_yy_interaction_II, sigma_xy_interaction_II, __, __ = stresses.stress_Yoffe_II_MC(x_local,
    y_local, self.a_eff, sigma_xy_av_local, self.V_k, microvoid.rho_s, microvoid.G, microvoid.nu)
911.
912.
913.             sigma_xx_interaction = sigma_xx_interaction_I + sigma_xx_interaction_II
914.             sigma_yy_interaction = sigma_yy_interaction_I + sigma_yy_interaction_II# - sigma_yy_av_local # Correct the stresses so that
     we only get the stress increment from the crack (need to remove sigma_infty)
915.             sigma_xy_interaction = sigma_xy_interaction_I + sigma_xy_interaction_II - sigma_xy_av_local # Correct the stresses so that
    we only get the stress increment from the crack (need to remove sigma_infty)
916.
917.             # Correct the stresses so that we only get the stress increment from the crack (need to remove sigma_infty)
918.             # sigma_xy_interaction = sigma_xy_interaction - sigma_xy_av_local
919.
920.
921.         # Apply transmission factors
922.         # <transmission factors>
923.
924.
925.         # Convert stresses from local coordinates to GLOBAL coordinates?? DOESNT THIS TRANSFER STRESSES TO INSTANTANEOUS MF STRESSES
926.         sigma_xx_interaction_global, sigma_yy_interaction_global, sigma_xy_interaction_global = self.local_to_global_MF_stresses(sigma_
    xx_interaction, sigma_yy_interaction, sigma_xy_interaction, dir_n, frameOref)
927.
928.         # If negative normal stress applied to stress point set the stress increment to 0.
929.         # if sigma_yy_interaction_global < 0.:
930.         #     sigma_xx_interaction_global, sigma_yy_interaction_global, sigma_xy_interaction_global = 0., 0., 0.
931.
932.         # if sigma_xy_interaction_global < 0.:
933.         #     sigma_xx_interaction_global, sigma_yy_interaction_global, sigma_xy_interaction_global = 0., 0., 0.
934.             # sigma_xy_interaction_global = 0.
935.
936.
937.         return sigma_xx_interaction_global, sigma_yy_interaction_global, sigma_xy_interaction_global#, sigma_xx_interaction, sigma_yy_i
    nteraction, sigma_xy_interaction
938.
939.
```

STRESSES.PY

```
1.  # -*- coding: utf-8 -*-
2.  """
3.  Project: Stress Field ahead of Crack
4.
5.  This module is used to calculate the stresses in front of a crack.
6.  Both Yoffe moving crack and Griffith-Williams stationary crack stresses are considered.
7.
8.  The principal stresses and azimuthal (principal) directions can also be
9.  calculated using functions in this script.
10.
11. @author: Dominic Walker, 450239612, dwal9899
12. """
13.
14. import numpy as np
15.
16.
```

```
17.
18.  """ MAIN FUNCTION 1: Yoffe far-field stresses
19.  This function calculates the far-field stresses in a moving Yoffe Crack.
20.  It takes in material, geometric and loading parameters and returns the
21.  stress field in the following order
22.  [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
23.  The other stress components are zero.
24.
25.  Inputs: (Units are all SI units)
26.        xx   matrix containing all the x-values
27.        yy   matrix contianing all corresponding y-values
28.        a       = crack width [m]
29.        sigma_a = applied stress = applied traction stress = -1 x sigma_T
30.                  (to ensure that the crack faces are traction free)
31.        V       = crack velocity [m/s]
32.        rho_s   = material density [kg/m^3]
33.        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
34.        nu      = poisson's ratio []
35.
36.
37.  Notes & Assumptions
38.      -   Assume linear elastic?
39.      -   Ignore microcracks
40.      -   Only consider material in front of the crack in the direction that
41.          it is moving
42.
43.  """
44.  def stress_FarYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu):
45.
46.
47.
48.      alpha   = 1 - nu
49.      Cs = np.sqrt(G/rho_s)
50.      Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))
51.
52.      # Calculate some constant parameters to calculate stresses
53.      b_s = np.sqrt(1-(V/Cs)**2)
54.      b_2s = np.sqrt(1-(1/2)*(V/Cs)**2)
55.
56.      b_l = np.sqrt(1-(V/Cl)**2)
57.      b_2l = np.sqrt(1-(1/2)*(V/Cl)**2)
58.
59.      # Calculate some (non-constant) parameters
60.      psi = np.arctan(b_s*YY/XX)
61.      psi_1 = np.arctan(b_s*YY/(XX+a))
62.      psi_2 = np.arctan(b_s*YY/(XX-a))
63.      psi_av = (1/2)*(psi_1+psi_2)
64.
65.      phi = np.arctan(b_l*YY/XX)
66.      phi_1 = np.arctan(b_l*YY/(XX+a))
67.      phi_2 = np.arctan(b_l*YY/(XX-a))
68.      phi_av = (1/2)*(phi_1+phi_2)
69.
70.      rho = np.sqrt(XX**2 + (b_s**2)*YY**2)
71.      rho_1 = np.sqrt((XX+a)**2+(b_s**2)*YY**2)
72.      rho_2 = np.sqrt((XX-a)**2+(b_s**2)*YY**2)
73.      rho_gm = np.sqrt(rho_1*rho_2)
74.
75.      r = np.sqrt(XX**2 + (b_l**2)*YY**2)
76.      r_1 = np.sqrt((XX+a)**2+(b_l**2)*YY**2)
77.      r_2 = np.sqrt((XX-a)**2+(b_l**2)*YY**2)
78.      r_gm = np.sqrt(r_1*r_2)
79.
80.
81.      # Calculate the AWAY FROM TIP stresses:
82.      # Calculate sigma_xx for each coordinate (xx,yy)
83.      sigma_xx = sigma_a/(b_s*b_l-b_2s**4)*(b_2s**2*(2*b_2l**2-b_2s**2)*(r/r_gm)*np.cos(phi_av-phi) - b_s*b_l*(rho/rho_gm)*np.cos(psi_av-
     psi) - b_2s**2*(2*b_2l**2 - b_2s**2) + b_s*b_l)
84.
```

```
    # Calculate sigma_yy for each coordinate (xx,yy)
    sigma_yy = sigma_a+sigma_a/(b_s*b_l-b_2s**4)*(-1*b_2s**4*(r/r_gm)*np.cos(phi_av-phi) + b_s*b_l*(rho/rho_gm)*np.cos(psi_av-psi)+b_2s**4 - b_s*b_l)


    # Calculate sigma_xy for each coordinate (xx,yy)
    sigma_xy = (sigma_a*b_l*b_2s**2)/(b_s*b_l-b_2s**4)*((r/r_gm)*np.sin(phi_av-phi)-(rho/rho_gm)*np.sin(psi_av-psi))

    # Calculate sigma_zz for each coordinate (xx,yy)
    sigma_zz = nu*(sigma_xx + sigma_yy)

    # Calculate sigma_zw for each coordinate (xx,yy)
    sigma_zw = 0. #IGNORE THIS! -1*(sigma_a*b_l*V**2)/((b_s*b_l-b_2s**4)*4*alpha*Cs**2)*((rho/rho_gm)*np.sin(psi_av-psi))


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)




""" MAIN FUNCTION 2: Yoffe near-field stresses

Assumption: As for Griffith crack, near-field Yoffe is Applicable where r_R << a       (r_R = radial distance from right crack tip)

This function calculates the far-field stresses in a moving Yoffe Crack.
It takes in material, geometric and loading parameters and returns the
stress field in the following order
[sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
The other stress components are zero.

Inputs: (Units are all SI units)
      xx   matrix containing all the x-values
      yy   matrix contianing all corresponding y-values
      a       = crack width [m]
      sigma_a = applied stress = applied traction stress = -1 x sigma_T
                (to ensure that the crack faces are traction free)
      V       = crack velocity [m/s]
      rho_s   = material density [kg/m^3]
      G       = elastic shear modulus = E/(2(1+nu)) [Pa]
      nu      = poisson's ratio []

Procedure:

Notes & Assumptions:
    -   plane strain assumption
    -   Ignore microcracks
    -   Only consider material in front of the crack in the direction that
          it is moving
    -

"""
def stress_NearYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu):

    alpha   = 1 - nu
    Cs = np.sqrt(G/rho_s)
    Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))

    # Calculate some constant parameters to calculate stresses
    b_s = np.sqrt(1-(V/Cs)**2)
    b_2s = np.sqrt(1-(1/2)*(V/Cs)**2)

    b_l = np.sqrt(1-(V/Cl)**2)
    b_2l = np.sqrt(1-(1/2)*(V/Cl)**2)

    K_I = sigma_a*(np.pi*a)**(1/2)

    # Calculate some (non-constant) parameters
    psi_2 = np.arctan(b_s*YY/(XX-a))
    phi_2 = np.arctan(b_l*YY/(XX-a))
```

```
    r_R = np.sqrt((XX-a)**2+YY**2)
    theta_r = np.arctan(YY/(XX-a))


    # Calculate sigma_xx for each coordinate (xx,yy)
    sigma_xx = K_I*(2*np.pi*r_R)**(-1/2)*(b_s*b_l-b_2s**4)**(-1)*(b_2s**2*(2*b_2l**2-
b_2s**2)*np.cos(0.5*phi_2)*(np.cos(theta_r)**2 + (b_l**2)*np.sin(theta_r)**2)**(-
1/4) - b_s*b_l*np.cos(0.5*psi_2)*(np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(-1/4))

    # Calculate sigma_yy for each coordinate (xx,yy)
    sigma_yy = -1*K_I*(2*np.pi*r_R)**(-1/2)*(b_s*b_l-b_2s**4)**(-
1)*((b_2s**4)*np.cos(0.5*phi_2)*((np.cos(theta_r)**2 + (b_l**2)*np.sin(theta_r)**2)**(-
1/4)) - b_s*b_l*np.cos(0.5*psi_2)*((np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(-1/4)))

    # Calculate sigma_xy for each coordinate (xx,yy)
    sigma_xy = K_I*b_l*b_2s**2*(2*np.pi*r_R)**(-1/2)*(b_s*b_l-b_2s**4)**(-
1)*(np.sin(0.5*phi_2)*((np.cos(theta_r)**2 + (b_l**2)*np.sin(theta_r)**2)**(-
1/4)) - np.sin(0.5*psi_2)*((np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(-1/4)))

    # Calculate sigma_zz for each coordinate (xx,yy)
    sigma_zz = nu*(sigma_xx + sigma_yy)

    # Calculate sigma_zw for each coordinate (xx,yy)
    sigma_zw = 0. #IGNORE THIS! -1*K_I*b_l*(V**2)/((2*np.pi*r_R)**(1/2)*(b_s*b_l-
b_2s**4)*4*alpha*Cs**2)*(np.sin(1/2*psi_2)/((np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(1/4)))


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)









""" MAIN FUNCTION 1b: Yoffe far-field stresses - Mode II Cracking
This function calculates the far-field stresses in a moving Yoffe Crack for Mode II Cracking.
It takes in material, geometric and loading parameters and returns the
stress field in the following order
[sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
The other stress components are zero.

Inputs: (Units are all SI units)
      xx   matrix containing all the x-values
      yy   matrix contianing all corresponding y-values
      a       = crack width [m]
      sigma_a = applied stress = applied traction stress = -1 x sigma_T
                (to ensure that the crack faces are traction free)
      V       = crack velocity [m/s]
      rho_s   = material density [kg/m^3]
      G       = elastic shear modulus = E/(2(1+nu)) [Pa]
      nu      = poisson's ratio []


Notes & Assumptions
    -   Assume linear elastic?
    -   Ignore microcracks
    -   Only consider material in front of the crack in the direction that
        it is moving

"""
def stress_FarYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu):

    alpha   = 1 - nu
    Cs = np.sqrt(G/rho_s)
    Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))
```

```
215.
216.     # Calculate some constant parameters to calculate stresses
217.     b_s = np.sqrt(1-(V/Cs)**2)
218.     b_2s = np.sqrt(1-(1/2)*(V/Cs)**2)
219.
220.     b_l = np.sqrt(1-(V/Cl)**2)
221.     b_2l = np.sqrt(1-(1/2)*(V/Cl)**2)
222.
223.     # Calculate some (non-constant) parameters
224.     psi = np.arctan(b_s*YY/XX)
225.     psi_1 = np.arctan(b_s*YY/(XX+a))
226.     psi_2 = np.arctan(b_s*YY/(XX-a))
227.     psi_av = (1/2)*(psi_1+psi_2)
228.
229.     phi = np.arctan(b_l*YY/XX)
230.     phi_1 = np.arctan(b_l*YY/(XX+a))
231.     phi_2 = np.arctan(b_l*YY/(XX-a))
232.     phi_av = (1/2)*(phi_1+phi_2)
233.
234.     rho = np.sqrt(XX**2 + (b_s**2)*YY**2)
235.     rho_1 = np.sqrt((XX+a)**2+(b_s**2)*YY**2)
236.     rho_2 = np.sqrt((XX-a)**2+(b_s**2)*YY**2)
237.     rho_gm = np.sqrt(rho_1*rho_2)
238.
239.     r = np.sqrt(XX**2 + (b_l**2)*YY**2)
240.     r_1 = np.sqrt((XX+a)**2+(b_l**2)*YY**2)
241.     r_2 = np.sqrt((XX-a)**2+(b_l**2)*YY**2)
242.     r_gm = np.sqrt(r_1*r_2)
243.
244.
245.     # Calculate the AWAY FROM TIP stresses:
246.     # Calculate sigma_xx for each coordinate (xx,yy)
247.     sigma_xx = -1*sigma_aII*b_s/(b_s*b_l-b_2s**4)*(b_l*(2*b_2l**2-b_2s**2)*(r/r_gm)*np.sin(phi_av-
    phi) - b_s*(b_2s**2)*(rho/rho_gm)*np.sin(psi_av-psi))
248.
249.     # Calculate sigma_yy for each coordinate (xx,yy)
250.     sigma_yy = sigma_aII*b_s*(b_2s**2)/(b_s*b_l-b_2s**4)*(b_l*(r/r_gm)*np.sin(phi_av-phi) - b_s*(rho/rho_gm)*np.sin(psi_av-psi))
251.
252.
253.     # Calculate sigma_xy for each coordinate (xx,yy)
254.     sigma_xy = sigma_aII + (sigma_aII)/(b_s*b_l-b_2s**4)*(b_s*b_l*(r/r_gm)*np.cos(phi_av-phi) - b_2s**4*(rho/rho_gm)*np.cos(psi_av-
    psi) - b_s*b_l + b_2s**2)
255.
256.     # Calculate sigma_zz for each coordinate (xx,yy)
257.     sigma_zz = nu*(sigma_xx + sigma_yy)
258.
259.     # Calculate sigma_zw for each coordinate (xx,yy)
260.     sigma_zw = 0. #IGNORE THIS! -1*(sigma_aII*b_2s**2**V**2)/((b_s*b_l-b_2s**4)*4*alpha*Cs**2)*((rho/rho_gm)*np.cos(psi_av-psi) - 1.)
261.
262.
263.     return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)
264.
265.
266.
267.
268. """ MAIN FUNCTION 2b: Yoffe near-field stresses - Mode II Cracking
269.
270. Assumption: As for Griffith crack, near-field Yoffe is Applicable where r_R << a       (r_R = radial distance from right crack tip)
271.
272. This function calculates the far-field stresses in a moving Yoffe Crack for Mode II Cracking.
273. It takes in material, geometric and loading parameters and returns the
274. stress field in the following order
275. [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
276. The other stress components are zero.
277.
278. Inputs: (Units are all SI units)
279.       xx   matrix containing all the x-values
280.       yy   matrix contianing all corresponding y-values
281.       a       = crack width [m]
```

```
282.       sigma_a = applied stress = applied traction stress = -1 x sigma_T
283.                 (to ensure that the crack faces are traction free)
284.       V       = crack velocity [m/s]
285.       rho_s   = material density [kg/m^3]
286.       G       = elastic shear modulus = E/(2(1+nu)) [Pa]
287.       nu      = poisson's ratio []
288.
289. Procedure:
290.
291. Notes & Assumptions:
292.     -   plane strain assumption
293.     -   Ignore microcracks
294.     -   Only consider material in front of the crack in the direction that
295.           it is moving
296.     -
297.
298. """
299. def stress_NearYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu):
300.
301.     alpha   = 1. - nu
302.     Cs = np.sqrt(G/rho_s)
303.     Cl = Cs*np.sqrt(2*(1-nu)/(1-2*nu))
304.
305.     # Calculate some constant parameters to calculate stresses
306.     b_s = np.sqrt(1-(V/Cs)**2)
307.     b_2s = np.sqrt(1-(1/2)*(V/Cs)**2)
308.
309.     b_l = np.sqrt(1-(V/Cl)**2)
310.     b_2l = np.sqrt(1-(1/2)*(V/Cl)**2)
311.
312.     K_II = sigma_aII*(np.pi*a)**(1/2)
313.
314.     # Calculate some (non-constant) parameters
315.     psi_2 = np.arctan(b_s*YY/(XX-a))
316.     phi_2 = np.arctan(b_l*YY/(XX-a))
317.     r_R = np.sqrt((XX-a)**2+YY**2)
318.     theta_r = np.arctan(YY/(XX-a))
319.
320.
321.     # Calculate sigma_xx for each coordinate (xx,yy)
322.     sigma_xx = -1*K_II*b_s*(2*np.pi*r_R)**(-1/2)*(b_s*b_l-b_2s**4)**(-1)*(b_l*(2*(b_2l**2)-
    b_2s**2)*np.sin(0.5*phi_2)*(np.cos(theta_r)**2 + (b_l**2)*np.sin(theta_r)**2)**(-
    1/4) - b_s*(b_2s**2)*np.sin(0.5*psi_2)*(np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(-1/4))
323.
324.     # Calculate sigma_yy for each coordinate (xx,yy)
325.     sigma_yy = K_II*b_s*(b_2s**2)*(2*np.pi*r_R)**(-1/2)*(b_s*b_l-b_2s**4)**(-
    1)*(b_l*np.sin(0.5*phi_2)*((np.cos(theta_r)**2 + (b_l**2)*np.sin(theta_r)**2)**(-
    1/4)) - b_s*np.sin(0.5*psi_2)*((np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(-1/4)))
326.
327.     # Calculate sigma_xy for each coordinate (xx,yy)
328.     sigma_xy = K_II*(2*np.pi*r_R)**(-1/2)*(b_s*b_l-b_2s**4)**(-
    1)*(b_s*b_l*np.cos(0.5*phi_2)*((np.cos(theta_r)**2 + (b_l**2)*np.sin(theta_r)**2)**(-
    1/4)) - (b_2s**4)*np.cos(0.5*psi_2)*((np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(-1/4)))
329.
330.     # Calculate sigma_zz for each coordinate (xx,yy)
331.     sigma_zz = nu*(sigma_xx + sigma_yy)
332.
333.     # Calculate sigma_zw for each coordinate (xx,yy)
334.     sigma_zw = 0. #IGNORE THIS! -1*K_II*(b_2s**2)*(V**2)/((2*np.pi*r_R)**(1/2)*(b_s*b_l-
    b_2s**4)*4*alpha*Cs**2)*(np.cos(0.5*psi_2)/((np.cos(theta_r)**2 + (b_s**2)*np.sin(theta_r)**2)**(1/4)))
335.
336.
337.     return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)
338.
339.
340.
341.
342.
343.
```

```
""" MAIN FUNCTION 3: Griffith far-field stresses

This function calculates the far-field stresses in a Griffith-Inglis Crack.
It takes in material, geometric and loading parameters and returns the
stress at each point in (x,y) in the following order:
[sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
The other stress components are zero.

Inputs: (Units are all SI units)
      xx   matrix containing all the x-values
      yy   matrix contianing all corresponding y-values
      a       = crack width [m]
      sigma_a = applied stress = applied traction stress = -1 x sigma_T
                (to ensure that the crack faces are traction free)
      nu      = poisson's ratio []

Notes & Assumptions:
  -   Ignore microcracks
  -   material is linear elastic, isotropic solid
  -   no plastic deformation occurs around the crack tip
  -   infinitely long(z-direction), ellyptical hole
  -   sharp crack
  -   crack faces must be traction free
  -   when the applied load is removed the displacement across the crack
      faces is resuced until it vanishes
  -   plane strain conditions

"""
def stress_FarGriff(XX, YY, a, sigma_a, nu):


    r_L = np.sqrt((XX+a)**2+YY**2)
    r_R = np.sqrt((XX-a)**2+YY**2)
    r_C = np.sqrt(XX**2 + YY**2)
    r_gm = np.sqrt(r_L*r_R)

    theta_L = np.arctan(YY/(XX+a))
    theta_R = np.arctan(YY/(XX-a))
    theta_C = np.arctan(YY/XX)
    theta_av = (1/2)*(theta_L+theta_R)

    # Calculate the AWAY FROM TIP stresses:
    # Calculate sigma_xx for each coordinate (XX,YY)
    sigma_xx = -1*sigma_a + sigma_a*(r_C/(2*r_gm))*(2*np.cos(theta_av-
theta_C) + 2*np.sin(theta_C)*np.sin(theta_av) - np.sin(theta_R)*np.sin(theta_av+theta_R-
theta_C) - np.sin(theta_L)*np.sin(theta_av+theta_L-theta_C))

    # Calculate sigma_yy for each coordinate (XX,YY)
    sigma_yy = sigma_a*(r_C/(2*r_gm))*(2*np.cos(theta_av-
theta_C) - 2*np.sin(theta_C)*np.sin(theta_av) + np.sin(theta_R)*np.sin(theta_av+theta_R-
theta_C) + np.sin(theta_L)*np.sin(theta_av+theta_L-theta_C))


    # Calculate sigma_xy for each coordinate (XX,YY)
    sigma_xy = sigma_a*(r_C/(2*r_gm))*(np.sin(theta_R)*np.cos(theta_av+theta_R-theta_C) + np.sin(theta_L)*np.cos(theta_av+theta_L-
theta_C) - 2*np.sin(theta_C)*np.cos(theta_av))

    # Calculate sigma_zz for each coordinate (XX,YY)
    sigma_zz = nu*(sigma_xx + sigma_yy)

    # Calculate sigma_zw for each coordinate (XX,YY)
    sigma_zw = 0. #IGNORE THIS! -1*sigma_a*(r_C/r_gm)*np.sin(theta_av-theta_C)

    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)


""" MAIN FUNCTION 4: Griffith near-field stresses

Applicable where r_R << a       (r_R = radial distance from right crack tip)
```

```
408.
409. This function calculates the far-field stresses in a Griffith-Inglis Crack.
410. It takes in material, geometric and loading parameters and returns the
411. stress at each point in (x,y) in the following order:
412. [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
413. The other stress components are zero.
414.
415. Inputs: (Units are all SI units)
416.       xx   matrix containing all the x-values
417.       yy   matrix contianing all corresponding y-values
418.       a       = crack width [m]
419.       sigma_a = applied stress = applied traction stress = -1 x sigma_T
420.                 (to ensure that the crack faces are traction free)
421.       nu      = poisson's ratio []
422.
423.
424. Notes: Assumptions
425.   -   Ignore microcracks
426.   -   material is linear elastic, isotropic solid
427.   -   no plasti deformatio noccurs around the crack tip
428.   -   infinitely long(z-direction), ellyptical hole
429.   -   sharp crack
430.   -   crack faces must be traction free
431.   -   when the applied load is removed the displacement across the crack
432.       faces is resuced until it vanishes
433.   -   plane strain conditions
434.
435. """
436. def stress_NearGriff(XX,YY,a,sigma_a,nu):
437.
438.     r = np.sqrt((XX-a)**2+YY**2)
439.     theta = np.arctan(YY/(XX-a))
440.     K_I = sigma_a*(np.pi*a)**(1/2)
441.
442.     # Calculate the AWAY FROM TIP stresses:
443.     # Calculate sigma_xx for each coordinate (XX,YY)
444.     sigma_xx = (K_I/(2*np.pi*r)**(1/2))*np.cos((1/2)*theta)*(1 - np.sin((1/2)*theta)*np.sin((3/2)*theta))
445.
446.     # Calculate sigma_yy for each coordinate (XX,YY)
447.     sigma_yy = (K_I/(2*np.pi*r)**(1/2))*np.cos((1/2)*theta)*(1 + np.sin((1/2)*theta)*np.sin((3/2)*theta))
448.
449.
450.     # Calculate sigma_xy for each coordinate (XX,YY)
451.     sigma_xy = (K_I/(2*np.pi*r)**(1/2))*np.sin((1/2)*theta)*np.cos((1/2)*theta)*np.cos((3/2)*theta)
452.
453.     # Calculate sigma_zz for each coordinate (XX,YY)
454.     sigma_zz = nu*(sigma_xx + sigma_yy)
455.
456.     # Calculate sigma_zw for each coordinate (XX,YY)
457.     sigma_zw = 0. #IGNORE THIS! -1*(K_I/(2*np.pi*r)**(1/2))*np.sin((1/2)*theta)
458.
459.
460.     return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)
461.
462.
463.
464.
465.
466.
467.
468. """ MAIN FUNCTION __: Griffith far-field stresses - MODE II CRACKING
469.
470. This function calculates the far-field stresses in a Griffith-Inglis Crack for MODE II CRACKING.
471. It takes in material, geometric and loading parameters and returns the
472. stress at each point in (x,y) in the following order:
473. [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
474. The other stress components are zero.
475.
476. Inputs: (Units are all SI units)
```

```
       xx   matrix containing all the x-values
       yy   matrix contianing all corresponding y-values
       a       = crack width [m]
       sigma_a = applied stress = applied traction stress = -1 x sigma_T
                 (to ensure that the crack faces are traction free)
       nu      = poisson's ratio []

Notes & Assumptions:
   -   Ignore microcracks
   -   material is linear elastic, isotropic solid
   -   no plastic deformation occurs around the crack tip
   -   infinitely long(z-direction), ellyptical hole
   -   sharp crack
   -   crack faces must be traction free
   -   when the applied load is removed the displacement across the crack
       faces is resuced until it vanishes
   -   plane strain conditions

"""
def stress_FarGriff_II(XX, YY, a, sigma_aII, nu):


    r_L = np.sqrt((XX+a)**2+YY**2)
    r_R = np.sqrt((XX-a)**2+YY**2)
    r_C = np.sqrt(XX**2 + YY**2)
    r_gm = np.sqrt(r_L*r_R)

    theta_L = np.arctan(YY/(XX+a))
    theta_R = np.arctan(YY/(XX-a))
    theta_C = np.arctan(YY/XX)
    theta_av = 0.5*(theta_L+theta_R)

    # Calculate the AWAY FROM TIP stresses:

    # Calculate sigma_yy for each coordinate (XX,YY)
    sigma_yy = sigma_aII*(r_C/(2*r_gm))*(np.sin(theta_R)*np.cos(theta_av+theta_R-theta_C) + np.sin(theta_L)*np.cos(theta_av+theta_L-
theta_C) -2*np.sin(theta_C)*np.cos(theta_av))

    # Calculate sigma_xx for each coordinate (XX,YY)
    sigma_xx = -1*sigma_yy - 2*sigma_aII*(r_C/r_gm)*np.sin(theta_av-theta_C)

    # Calculate sigma_xy for each coordinate (XX,YY)
    sigma_xy = sigma_aII*(r_C/(2*r_gm))*(2*np.cos(theta_av-
theta_C) + 2*np.sin(theta_C)*np.sin(theta_av) - np.sin(theta_R)*np.sin(theta_av+theta_R-
theta_C) - np.sin(theta_L)*np.sin(theta_av+theta_L-theta_C))

    # Calculate sigma_zz for each coordinate (XX,YY)
    sigma_zz = nu*(sigma_xx + sigma_yy)

    # Calculate sigma_zw for each coordinate (XX,YY)
    sigma_zw = 0. #IGNORE THIS! -1*sigma_aII*(r_C/r_gm)*np.cos(theta_av-theta_C)

    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)



""" MAIN FUNCTION __: Griffith near-field stresses - MODE II CRACKING

Applicable where r_R << a       (r_R = radial distance from right crack tip)

This function calculates the far-field stresses in a Griffith-Inglis Crack for MODE II CRACKING.
It takes in material, geometric and loading parameters and returns the
stress at each point in (x,y) in the following order:
[sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] (Pa)
The other stress components are zero.

Inputs: (Units are all SI units)
       xx   matrix containing all the x-values
       yy   matrix contianing all corresponding y-values
```

```
       a       = crack width [m]
       sigma_a = applied stress = applied traction stress = -1 x sigma_T
                 (to ensure that the crack faces are traction free)
       nu      = poisson's ratio []


Notes: Assumptions
  -   Ignore microcracks
  -   material is linear elastic, isotropic solid
  -   no plasti deformatio noccurs around the crack tip
  -   infinitely long(z-direction), ellyptical hole
  -   sharp crack
  -   crack faces must be traction free
  -   when the applied load is removed the displacement across the crack
      faces is resuced until it vanishes
  -   plane strain conditions

"""
def stress_NearGriff_II(XX,YY,a,sigma_aII,nu):

    r = np.sqrt((XX-a)**2+YY**2)
    theta = np.arctan(YY/(XX-a))
    K_II = sigma_aII*(np.pi*a)**(1/2)

    # Calculate the NEAR TIP stresses:
    # Calculate sigma_xx for each coordinate (XX,YY)
    sigma_xx = -1*(K_II/(2*np.pi*r)**(1/2))*np.sin(0.5*theta)*(2 + np.cos(0.5*theta)*np.cos(1.5*theta))

    # Calculate sigma_yy for each coordinate (XX,YY)
    sigma_yy = (K_II/(2*np.pi*r)**(1/2))*np.sin(0.5*theta)*np.cos(0.5*theta)**np.cos(1.5*theta)


    # Calculate sigma_xy for each coordinate (XX,YY)
    sigma_xy = (K_II/(2*np.pi*r)**(1/2))*np.cos(0.5*theta)*(1 - np.sin(0.5*theta)*np.sin(1.5*theta))

    # Calculate sigma_zz for each coordinate (XX,YY)
    sigma_zz = nu*(sigma_xx + sigma_yy)

    # Calculate sigma_zw for each coordinate (XX,YY)
    sigma_zw = 0. #IGNORE THIS! -1*(K_II/(2*np.pi*r)**(1/2))*np.cos((1/2)*theta)


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)









"""Weights Function - Near Field Stresses"""
# Exponential Weights Function
def weights(r_R,a):
    return np.exp(-1*3*r_R/a)


""" MAIN FUNCTION 5a: Yoffe Stresses Inferred (Near field & far field)


This function merges the near field and far field stress distributions for each
stress component produced by stress_FarYoffe() and stress_NearYoffe().

Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
```

```
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                        the limits of the region of the transition zone in terms
                        of a radius length from r_R

"""
# kwarg** is to indicate if the datatype is either an integer or numpy array.
#transition_zone = [0.1*a, a]
def stress_Yoffe(XX, YY, a, sigma_a, V, rho_s, G, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_NearYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)

    # Calculate far-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_FarYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)

    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate near-field stresses
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)


    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)

    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)









""" MAIN FUNCTION 5b: Yoffe Stresses Inferred (Near field & far field) - Mode II cracking


This function merges the near field and far field stress distributions for each
stress component produced by stress_FarYoffe_II() and stress_NearYoffe_II().

```

```
Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                        the limits of the region of the transition zone in terms
                        of a radius length from r_R

"""
# kwarg** is to indicate if the datatype is either an integer or numpy array.
#transition_zone = [0.1*a, a]
def stress_Yoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_NearYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu)

    # Calculate far-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_FarYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu)

    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate near-field stresses
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G
, nu)

    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu)


    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)



""" MAIN FUNCTION 5a: Yoffe Stresses Inferred (Near field & far field)


This function merges the near field and far field stress distributions for each
```

```
stress component produced by stress_FarYoffe() and stress_NearYoffe().

Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                        the limits of the region of the transition zone in terms
                        of a radius length from r_R

"""
# kwarg** is to indicate if the datatype is either an integer or numpy array.
#transition_zone = [0.1*a, a]
def stress_Yoffe_MC(XX, YY, a, sigma_a, V, rho_s, G, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_NearYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)

    # Calculate far-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_FarYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)

    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate near-field stresses
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)


    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarYoffe(XX, YY, a, sigma_a, V, rho_s, G, nu)

    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far



    # If we are along the middle of the MC, then set stresses to background stresses - these will be subtracted out in the simulation.

    if str(type(XX)) == "<class 'numpy.ndarray'>":
        sigma_yy[XX <=a] = sigma_a
        sigma_xy[XX <=a] = 0.
        sigma_xx[XX <=a] = 0.
    elif (str(type(XX)) in ["<class 'numpy.float64'>", "float"]) & (XX < a):
```

```
        sigma_yy = sigma_a
        sigma_xy = 0.
        sigma_xx = 0.
        # print('In middle region')

    elif (str(type(XX)) not in ["<class 'numpy.float64'>", "float"]) & (XX < a):
        print('type not accounted for',type(XX), XX)



    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)









""" MAIN FUNCTION 5b: Yoffe Stresses Inferred (Near field & far field) - Mode II cracking


This function merges the near field and far field stress distributions for each
stress component produced by stress_FarYoffe_II() and stress_NearYoffe_II().

Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                          the limits of the region of the transition zone in terms
                          of a radius length from r_R

"""
# kwarg** is to indicate if the datatype is either an integer or numpy array.
#transition_zone = [0.1*a, a]
def stress_Yoffe_II_MC(XX, YY, a, sigma_aII, V, rho_s, G, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_NearYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu)

    # Calculate far-field stresses
    [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw] = stress_FarYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu)

    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)
```

```
883.
884.     # Calculate near-field stresses
885.     [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G
    , nu)
886.
887.     # Calculate far-field stresses
888.     [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarYoffe_II(XX, YY, a, sigma_aII, V, rho_s, G, nu)

889.
890.     # Calculate the weighted average of the near and far field stresses
891.     sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
892.     sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
893.     sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
894.     sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
895.     sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far
896.
897.
898.
899.     # If we are along the middle of the MC, then set stresses to background stresses - these will be subtracted out in the simulation.

900.     if str(type(XX)) == "<class 'numpy.ndarray'>":
901.         sigma_yy[XX <=a] = 0.
902.         sigma_xy[XX <=a] = sigma_aII
903.         sigma_xx[XX <=a] = 0.
904.     elif (str(type(XX)) in ["<class 'numpy.float64'>", "float"]) & (XX < a):
905.         sigma_yy = 0.
906.         sigma_xy = sigma_aII
907.         sigma_xx = 0.
908.         # print('In middle region')
909.
910.     elif (str(type(XX)) not in ["<class 'numpy.float64'>", "float"]) & (XX < a):
911.         print('type not accounted for',type(XX), XX)
912.
913.
914.
915.
916.     return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)
917.
918.
919.
920.
921.
922.
923.
924.
925.
926. """ MAIN FUNCTION 6a: Griffith Stresses Inferred (Near field & far field)
927.
928. This function merges the near field and far field stress distributions for each
929. stress component produced by stress_FarGriff() and stress_NearGriff().
930.
931. Within the transitional zone the method of transition is linear such that the
932. transitional stresses can be calculated as:
933.         transitional_stress = alpha * stress_near + beta * stress_far,
934.
935.         where, alpha + beta = 1
936.         and alpha and beta are linearly varying between 0 and 1 over the
937.         transitional zone.
938.
939. Assumptions:
940.     -   The transitional zone is between r_R = 0.1*a and r_R = a
941.
942. Inputs: (Units are all SI units)
943.         xx   matrix containing all the x-values
944.         yy   matrix contianing all corresponding y-values
945.         a       = crack width [m]
946.         sigma_a = applied stress = applied traction stress = -1 x sigma_T
947.                     (to ensure that the crack faces are traction free)
948.         V       = crack velocity [m/s]
```

```
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                        the limits of the region of the transition zone in terms
                        of a radius length from r_R
"""
def stress_Griff(XX, YY, a, sigma_a, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearGriff(XX,YY,a,sigma_a,nu)
    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarGriff(XX,YY,a,sigma_a,nu)


    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far

    # if str(type(XX)) != "<class 'numpy.float64'>":
    #     print(XX/a, str(type(XX)))


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)




""" MAIN FUNCTION 6b: Griffith Stresses Inferred (Near field & far field) - MODE II CRACKING

This function merges the near field and far field stress distributions for each
stress component produced by stress_FarGriff_II() and stress_NearGriff_II() for MODE II CRACKING.

Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
```

```
                        the limits of the region of the transition zone in terms
                        of a radius length from r_R
"""
def stress_Griff_II(XX, YY, a, sigma_aII, nu):


    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearGriff_II(XX,YY,a,sigma_aII,nu)
    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarGriff_II(XX,YY,a,sigma_aII,nu)


    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far

    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)




""" MAIN FUNCTION __: Griffith Stresses Inferred (Near field & far field) - FOR MICROCRACKS

This function merges the near field and far field stress distributions for each
stress component produced by stress_FarGriff() and stress_NearGriff().

Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                        the limits of the region of the transition zone in terms
                        of a radius length from r_R
"""
def stress_Griff_MC(XX, YY, a, sigma_a, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
```

```
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearGriff(XX,YY,a,sigma_a,nu)
    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarGriff(XX,YY,a,sigma_a,nu)


    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far

    # If we are along the middle of the MC, then set stresses to background stresses - these will be subtracted out in the simulation.

    if str(type(XX)) == "<class 'numpy.ndarray'>":
        sigma_yy[XX <=a] = sigma_a
        sigma_xy[XX <=a] = 0.
        sigma_xx[XX <=a] = 0.
    elif (str(type(XX)) in ["<class 'numpy.float64'>", "float"]) & (XX < a):
        sigma_yy = sigma_a
        sigma_xy = 0.
        sigma_xx = 0.
        # print('In middle region')

    elif (str(type(XX)) not in ["<class 'numpy.float64'>", "float"]) & (XX < a):
        print('type not accounted for',type(XX), XX)

    # else:
        # If we're in here then we are considering a single point, and that single point is not in the middle region.


    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)


""" MAIN FUNCTION __: Griffith Stresses Inferred (Near field & far field) - MODE II CRACKING

This function merges the near field and far field stress distributions for each
stress component produced by stress_FarGriff_II() and stress_NearGriff_II() for MODE II CRACKING.

Within the transitional zone the method of transition is linear such that the
transitional stresses can be calculated as:
        transitional_stress = alpha * stress_near + beta * stress_far,

        where, alpha + beta = 1
        and alpha and beta are linearly varying between 0 and 1 over the
        transitional zone.

Assumptions:
    -   The transitional zone is between r_R = 0.1*a and r_R = a

Inputs: (Units are all SI units)
        xx   matrix containing all the x-values
        yy   matrix contianing all corresponding y-values
        a       = crack width [m]
        sigma_a = applied stress = applied traction stress = -1 x sigma_T
                    (to ensure that the crack faces are traction free)
        V       = crack velocity [m/s]
        rho_s   = material density [kg/m^3]
        G       = elastic shear modulus = E/(2(1+nu)) [Pa]
        nu      = poisson's ratio []

        r_R radial distance from the right crack tip (i.e. r_R = r-a)
        transition_zone = a list with two elements. The elements in order define
                          the limits of the region of the transition zone in terms
                          of a radius length from r_R
"""
```

```
def stress_Griff_II_MC(XX, YY, a, sigma_aII, nu):

    # Get radial distance of the point (x,y) from the point where x = a, y = 0
    r_R = np.sqrt((XX-a)**2 + YY**2)

    # Calculate near-field stresses
    [sigma_xx_near, sigma_yy_near, sigma_xy_near, sigma_zz_near, sigma_zw_near] = stress_NearGriff_II(XX,YY,a,sigma_aII,nu)
    # Calculate far-field stresses
    [sigma_xx_far, sigma_yy_far, sigma_xy_far, sigma_zz_far, sigma_zw_far] = stress_FarGriff_II(XX,YY,a,sigma_aII,nu)


    # Calculate the weights using the weights funtion
    weights_array = weights(r_R,a)

    # Calculate the weighted average of the near and far field stresses
    sigma_xx = (weights_array)*sigma_xx_near + (1. - weights_array)*sigma_xx_far
    sigma_yy = (weights_array)*sigma_yy_near + (1. - weights_array)*sigma_yy_far
    sigma_xy = (weights_array)*sigma_xy_near + (1. - weights_array)*sigma_xy_far
    sigma_zz = (weights_array)*sigma_zz_near + (1. - weights_array)*sigma_zz_far
    sigma_zw = (weights_array)*sigma_zw_near + (1. - weights_array)*sigma_zw_far



    # If we are along the middle of the MC, then set stresses to background stresses - these will be subtracted out in the simulation.

    if str(type(XX)) == "<class 'numpy.ndarray'>":
        sigma_yy[XX <=a] = 0.
        sigma_xy[XX <=a] = sigma_aII
        sigma_xx[XX <=a] = 0.
    elif (str(type(XX)) in ["<class 'numpy.float64'>", "float"]) & (XX < a):
        sigma_yy = 0.
        sigma_xy = sigma_aII
        sigma_xx = 0.
        # print('In middle region')

    elif (str(type(XX)) not in ["<class 'numpy.float64'>", "float"]) & (XX < a):
        print('type not accounted for',type(XX), XX)

    # else:
        # If we're in here then we are considering a single point, and that single point is not in the middle region.



    return [sigma_xx, sigma_yy, sigma_xy, sigma_zz, sigma_zw]   # (Unit: Pa)









""" MAIN FUNCTION 7: Principal Stresses and Directions
This function takes in the cauchy stress tensor components and calculates the
principal stresses and principal direction of the (major/minor???) principal stress.


Inputs: (Units need only be consistent for the stresses)
     sigma_xx   matrix containing the normal stresses on a plane
                that is perpendicular to the x-axis, at each point (x,y)
     sigma_yy   matrix containing all the normal stresses on a plane
                that is perpendicular to the y-axis, at each point (x,y)
     sigma_xy   matrix containing all the shear stresses on a plane
                that is perpendicular to the y-axis or x-axis, at each point (x,y)

Outputs:
    sigma_1                 matrix containing the major principal stress on a plane
    sigma_2                 matrix containing the major principal stress on a plane
    rot_to_principal_dir    rotation required of an element to obtain element with
                            principal stesses.
```

```
                            The actual direction (angle of inclination in 2D) of
                            the major principal plane to the horizontal is at this
                            stage thought to be the same as the rotation angle of each element.
                            UNITS: radians


Notes: Assumptions
    -   sigma_zz can simply be ignored and so the plane stress situation can be considered

"""
def transform2d_ToPrincipal(sigma_xx, sigma_yy, sigma_xy):

    # Calculate the major principal stress at each point (x,y)
    sigma_1 = 0.5*(sigma_xx + sigma_yy) + np.sqrt((0.5*(sigma_xx-sigma_yy))**2 + sigma_xy**2)

    # Calculate the minor principal stresses at each point (x,y)
    sigma_2 = 0.5*(sigma_xx + sigma_yy) - np.sqrt((0.5*(sigma_xx-sigma_yy))**2 + sigma_xy**2)

    # Calculate the angle of rotation of an element at each (x,y) in order
    # to obtain the principal stresses (UNITS: radians)
    rot_to_principal_dir = 0.5*np.arctan(2*sigma_xy/(sigma_xx-sigma_yy))*(-1)   # NOTE: the *-
1 is compensating for rotating everything the wrong way everywhere else. Multiplying by -1 is just a quick patch to the issue
    # Note: Rotation is anticlockwise positive and corresonts to the STRESS ELEMENT rotation (NOT rotation in Mohrs Circle)


    return [sigma_1, sigma_2, rot_to_principal_dir]
```