## *Numerical Simulation of Fluid Overpressure Driven Faulting and Seismicity In Low Porosity Rocks*

SNELL, THOMAS,ALAN

# NUMERICAL SIMULATION OF FLUID OVERPRESSURE DRIVEN FAULTING AND SEISMICITY IN LOW POROSITY ROCKS

**Thomas Snell**

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Durham University

Department of Earth Sciences, Durham University, UK

2019

***Abstract***

Pore fluid overpressures in active fault systems can drive fluid flow and cause fault weakening and seismicity. In return, deformation accommodated by different mode of failure (e.g. brittle vs. ductile) also affects fault zone permeability and, hence, fluid flow and pore fluid pressure distribution. The resulting non-linear, complex feedback between fluid flow, fluid pressure and fault deformation controls the length of the nucleation phase of an earthquake and the duration of the interseismic period.

In this thesis we: 1) model overpressured, supercritical $CO_2$ fluid flow in natural, exhumed faults in evaporite sequences, which represent an analogue of the seismic sources at hypocentre depth of recent seismic events in the Northern Apennines of Italy (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia earthquakes). 2) perform parameter studies on pore pressure diffusion and earthquake nucleation, with realistic models of ductile failure, varying the dimension of components of fault zone architecture and neighbouring lithology, outer fault core width and the height of pressurised layers abutting the fault core.

Our results show that: 1) the duration of the nucleation phase is significantly reduced, from a few years to a few months, when realistic models of fault zone architecture and pore pressure- and deformation-dependent permeability are considered. We implement a four-component model of fault zone architecture in simulations (damage zone, outer fault core, inner fault core and primary slip zone) in contrast to the one- or two-component models of fault zone architecture previously considered. 2) For a given tectonic loading rate, a thinner fault core results in a more effective fault weakening. The impact of fluid flow on the fault

being more significant for faults with a thinner rather than thicker outer fault core. In the absence of fluids, the base mechanical strength of the slipping portion of the fault did not vary with thickness. Similarly, an increasing the thickness of an overpressured aquifer intersecting a fault in the damage zone produces a higher magnitude of pore pressure in the fault core, which weakens the principal slip zone. Understanding the controls exerted on the duration of the nucleation phase of earthquakes has important implications for premonitory signal detection, as identifying extended nucleation phases of active faults would increase the likelihood of detection of early seismicity warnings.

# Contents

***Declaration***

I declare that this thesis, which I submit for the degree of Doctor of Philosophy at Durham University, is my own work and not substantially the same as any which has previously been submitted at this or any other university.

Thomas A. Snell

Durham University

October 2019

***Acknowledgements***

Firstly, I would like to thank Nic, for being a supportive supervisor and guiding me through the transition from Physics and Biophysics to Geophysics. His high standards have helped me develop my communication and writing skills to higher levels and made me a more rounded person.

Second, thanks to Jeroen for both technical and emotional support throughout the project. His clear thinking really sharpened the critical processes I use for analysing ideas the project, particularly the numerical simulation code would not have built without his input.

Third, I would like to thank Stefan for his superb theoretical input, really helping me to understand the earthquake nucleation phase and the nature of the approximations involved.

I would like to thank Anna my wife and Arthur my son, for putting up with the disruption on family life that the PhD caused, for believing in my and giving me the time and emotional support, I needed to finish. Thanks to Arthur for reminding me continuously that there are far more important things to worry about in life.

# CHAPTER 1

*Introduction*

# 1 Introduction

## 1.1 **Rationale and broad context**

The potential for earthquakes exists throughout the crust, both along intracontinental and plate boundaries faults, where deformation is high and stress level may be near the strength of faults (Townend and Zoback, 2000). Under these conditions, small perturbations such as increasing pore fluid pressure can affect the stress state of the faulted crust and cause fault reactivation, frictional instability and trigger seismicity.

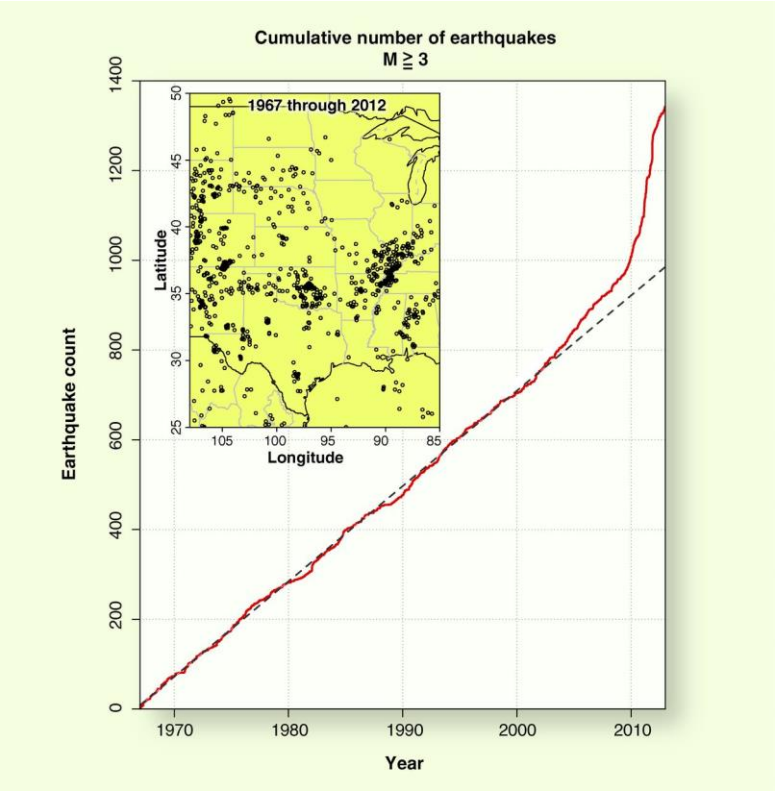



**Figure 1.1: "Cumulative count of earthquakes with M≥ 3 in the central and eastern United States, 1967–2012.The dashed line corresponds to the long-term rate of 21.2 earthquakes/year. (Inset ) Distribution of epicenters in the region considered here." (Ellsworth, 2013).Subsurface pore fluid pressure gradients (Cox, 2010; Sibson, 1990, 1992) and fluid migration (Collettini et al., 2009; Cox, 1995; Cox et al., 1987; De Paola et al., 2008; Hickman et al., 1995; Miller, 1996;**

**Rice, 1992; Sibson, 2000) in the subsurface can significantly alter the frictional strength of faults and induce seismicity. Natural fluid flows (Di Luccio et al., 2010; Mahesh et al., 2012; Miller et al., 2004; Mizoguchi et al., 2008; Parotidis et al., 2003; Terakawa et al., 2013; Yoshida et al., 2016, 2003) and a number of human subsurface injection activities (Davies et al., 2013) can induce these pore pressure gradients in the faulted crust.**

A number of large earthquakes have been driven by natural subsurface fluid flow (up to $M_W$ 9.0, Terakawa et al., 2013). Mantle degassing through the release of supercritical carbon dioxide ($CO_2$) is thought to have driven many instances of natural seismicity. This $CO_2$ is released from structural or lithological traps in the subsurface (Noir et al., 1997; Nur and Booker, 1972; Parotidis et al., 2003; Terakawa et al., 2013; Yoshida et al., 2016), fed by the degassing processes (Di Luccio et al., 2010; Mahesh et al., 2012; Miller et al., 2004). Further, this $CO_2$ release itself can depend on seismicity and be released coseismically triggering subsequent further seismicity (Keranen et al., 2013; Sumy et al., 2014).

A number of subsurface injection activities contribute to modern energy production and can induce seismicity, particularly: 1) carbon capture and storage (Zoback and Gorelick, 2012), by injection of supercritical $CO_2$ into deep formations for permanent capture and storage; 2) hydraulic fracturing (Atkinson et al., 2015, 2016; Bao and Eaton, 2016; Clarke et al., 2014; Davies et al., 2013; De Pater and Baisch, 2011; Elsworth et al., 2016; Farahbod et al., 2015a, 2015b; Friberg et al., 2014; Holland, 2013; Keranen et al., 2013; Lei et al., 2017; Maxwell et al., 2002; McGarr, 2014; Rutledge et al., 2004; Rutledge and Phillips, 2003; Schultz et al., 2015; Skoumal et al., 2015; Sumy et al., 2014; Vermylen and Zoback, 2011), by injection of water into low porosity, tight reservoirs to stimulate hydro-fracturing,

and enable oil and gas production; and 3) wastewater disposal (Ake et al., 2005; Frohlich, 2012; Frohlich and Brunt, 2013; Hornbach et al., 2016; Keranen et al., 2014, 2013; Kim, 2013), by injection into deep saline aquifers.

There is still considerable uncertainty concerning the relationship between the timing of seismicity after human subsurface fluid injection (Folger and Tiemann, 2015), sometimes occurring immediately and, in other cases, long after the fluid injection has begun or even ceased. This variability in timing indicates a complicated relationship between low porosity faults, fluid flow and earthquake nucleation. However, continent-scale seismic monitoring of the USA provides evidence that areas considered geologically stable have now experienced increased rates of seismicity due to fluid-injection activities (Fig 1.1) (Ellsworth, 2013; McGarr, 2014; McGarr et al., 2015; Weingarten et al., 2015).

Faulting and rock failure in the seismogenic layer of the brittle crust (about 15 km depth) can be accommodated by two main failure modes, brittle deformation in rocks displaying elastic-frictional behaviour (localised deformation by discrete faulting; R. H. Sibson, 1977) and ductile failure (fracturing distributed at the mesoscopic scale; Rutter et al., 1986). The specific mode of failure is known to control the development of fracture patterns (Caine et al., 1996; Cox, 1995; Mitchell and Faulkner, 2008; Peach and Spiers, 1996; Wong et al., 1997; Zoback and Byerlee, 1975), which affect the transport properties of rocks and, hence, control fluid circulation in the upper crust (De Paola et al., 2009; Fischer, 1992; Morrow and Lockner, 1997, 1994; Paterson and Wong, 2005; Zhu et al., 1997).

Brittle faults are zones of finite thickness, which are comprised of distinct domains, each with a characteristic suite of fault rocks and different transport

properties (Caine et al., 1996); taken together these domains are referred to as the fault zone architecture (Caine et al., 1996). A simplistic, but still useful schematic model of this fault zone architecture considers a fault core, accommodating most of the strain, surrounded by a damage zone of distributed fracturing which is itself surrounded by relatively intact protolith rock (Chester et al., 2004; Faulkner et al., 2010). More complex models of fault zone architecture refine the fault core itself in to an outer fault core of highly fractured rocks, an inner fault core of cohesive cataclasite and primary slip zones composed of incohesive gouges (De Paola et al., 2008).

The size and distribution of the different fault zone domains control fluid flow within a fault zone (Caine et al., 1996) and, hence, fault reactivation. The following factors, parameters and conditions control the mode of failure and architecture of fault zones:

- Environmental conditions: confining pressure controls the transition from brittle to ductile failure in rocks (Byerlee, 1968) and pore fluid pressure, mediates a similar effect by controlling the effective confining pressure.
- Lithological variations: as, for any given environmental conditions, they may accommodate deformation by different mode of failure (e.g. brittle vs. ductile) and produce differing suites of fault rocks in the fault zone domains (e.g. Bullock et al., 2014; Collettini et al., 2009; De Paola et al., 2008; D. R. Faulkner et al., 2010), each with differing transport properties.

The above parameters control fault zone architecture and reactivation processes in the brittle crust and are typically resolved with varying degrees of certainty. Several fault zone parameters are essentially unknowns, such as the initial

stress state and the evolution of rock permeability with pore fluid pressure and deformation. In fact, pore pressure perturbation and fluid flow in the subsurface is dependent on the level of connectivity of the fault/fracture patterns. Such limitations in the predictions of fault reactivation do impact on our ability to estimate earthquake nucleation and earthquake recurrence intervals, which are affected by the same unknowns, as well as further uncertainties due to the lack of information about fault zone dimensions, internal structure and large-scale connectivity.

Fault reactivation usually begins as stable, non-oscillatory frictional sliding on a fault asperity, which is usually a relatively small fault patch with either low frictional strength (e.g. due to high pore fluid pressure) or high shear stress (e.g. stress concentration at fault bends). An earthquake can then nucleate when such rupture patch reaches a critical size, the nucleation length, at which fast and unstable, oscillatory sliding and rupture propagation begin (Marone, 1998; Scholz, 1998).

Natural subsurface fluid flow has been implicated in both the deadly $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia seismic sequences regions with irregular seismic recurrence intervals. (De Paola et al., 2008; Porreca et al., 2018). Constraining the long-term controls of fault reactivation and the short-term controls of the duration of the earthquake nucleation phase, leading to seismic faulting, can help mitigate the seismic hazard such fluid induced seismicity. As such understanding the dependence of earthquake nucleation and the nucleation phase on the transport and geometric properties of complex, natural fault zones, therefore, become the focus of this thesis.

## 1.2 **Aims and objectives**

Fault zone transport properties and geometric parameters nonlinearly mediate the hydraulic connectivity between naturally pressurized reservoirs or injection sites and the actively slipping portion of faults. This nonlinearity arises both from the pressure dependence of permeability and the hysteretic permeability changes associated with discontinuous fracturing. Put directly the ratio of permeability and pressure is not a constant tensor across time. Therefore, the solutions to the pressure field for different faults independently exhibiting both above behaviours in response to the same stimulus cannot be superimposed to solve for a fault simultaneously exhibiting both behaviours.

The research presented in this thesis aims to constrain better how such fault zone properties and controlling parameters influence pore pressure diffusion in fault zones, when a simplified but still realistic fault zone architecture is accounted for. Previous simulation studies have approximated using one or two component models of fault zone architecture and considered only continuous failure behaviors (Cappa et al., 2009; Cappa and Rutqvist, 2011; Hsiung et al., 2005; Mazzoldi et al., 2012; Rinaldi et al., 2014; Rutqvist et al., 2013, 2009; Leclère et al., 2015). Such simple models represent gross fluid flow behaviours in the vicinity of the fault, but do not resolve how the finer components composing the fault zone influence fluid flow behavior and earthquake nucleation. Further, the absence of discontinuous brittle or ductile mode of failure in the outer fault core in particular, may have a primary impact as it can be responsible for permeability changes within the fault core over several orders of magnitude (De Paola et al., 2009).

A more refined four-component model of fault zone architecture with more comprehensive models of continuous and discontinuous failure will allow us to simulate fluid-flow and failure in natural faults more closely, capture these primary contributions to fluid flow evolution and therefore resolve more precisely how these processes control earthquake nucleation.

Numerical experiments are performed to model fluid flow in natural fault zones with complex architecture, as taken from field studies of exhumed seismic fault analogues, and dynamic evolution of fault rock transport properties, as taken from rock mechanics experiments. Modelling results are then used to investigate how pore fluid variations may affect the strength of seismic faults during the interseismic period and control the earthquake nucleation phase. More specifically, the over-arching aims of this thesis are:

- To model pore pressure diffusion during the interseismic period in natural fault zones, accounting for their complex architecture and deformation features, due to the operation of realistic brittle and ductile modes of failure. We decompose fault zone architecture into a four-component model (damage zone, outer and inner fault core and primary slip zone), as opposed to a one- (Cappa et al., 2009; Cappa and Rutqvist, 2011; Hsiung et al., 2005; Mazzoldi et al., 2012; Rinaldi et al., 2014; Rutqvist et al., 2013, 2009) or two-component model in previous studies (Leclère et al., 2015).
- To simulate the earthquake nucleation phase and the evolution of pore pressure during this period.
- To constrain the dependence of earthquake nucleation on: transport and fault properties (e.g. particularly modes of failure, pore pressure and stress-sensitive permeability), multilayer scale lithological properties (e.g. varying

thickness of overpressured reservoirs) and fault zone dimensions (e.g. relative ratio of fault core/damage zone thickness).

### 1.3 **Methodology: Numerical modelling**

The above aims have been achieved by:

- Building a multiphysics model of nonlinear diffusion in low permeability fault zones, in turn, incorporating realistic models of:
    1) Complex, natural fault zone architecture, as obtained from field observations of exhumed extensional faults in the Northern Apennines, assumed as analogous to the hypocentral faults of the $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia earthquakes (De Paola et al., 2008; Porreca et al., 2018).
    2) Permeability evolution during rock failure, as measured in triaxial deformation experiments with fluid flow on real fault rocks. Specifically those experiments performed on samples from the Perugia 2 and Fossonbrone 2 boreholes, located in the seismogenic belt of the Umbria-Marche Apennines in Italy (De Paola et al., 2009).
    3) Failure processes the fault core, which includes frictional sliding along primary slip zones and brittle and ductile mode of failure in the surrounding fault zone rocks. Failure envelops have been obtained from triaxial deformation experiments with fluid flow on real fault rocks (similar rock samples as at point two, De Paola et al., 2009).
    4) Earthquake nucleation processes and the transition of the system from stable, non-oscillatory sliding to fast and unstable, oscillatory sliding (e.g. the nucleation length criticality).

- Conducting A-, L- and S-stable numerical simulations (Dahlquist, 1963) of the highly stiff set of coupled partial differential equations (Jacobian matrix eigenvalues differing on the order of $10^{11}$) resulting from the multiphysics model above. A-, and L-stability indicate respectively that a solver given test equation $y' = ky$ subject to initial condition $y(0) = 1$, would provide a solution approximating $y(t) = exp(kt)$ and that the solution decays to zero in a single step as step size is increased to infinity for $k< 0$ (Hairer and Wanner, 1996). S-stability extends this, stating that when the applied solver is represented as a function of the jacobian and step size, that function should tend to zero as step size increases to infinity for each jacobian element .
- Introducing a novel mathematical formulation of the mode of failure problem with failure state variables as a non-smooth process. This formulation ensures tractability in reasonable computational time using a combination of an existing explicit singly diagonal implicit Runge-Kutta (ESDIRK) solver (for A-,L-,S-stability) and event detection (to ensure simulations remain mathematically well-posed near discontinuities) to minimise both the required number of simulation time-steps and cumulative truncation error.
- Conducting parameter studies to analyse the sensitivity of the fault fluid system to uncertainty or variation in typically poorly constrained lithological or pore pressure conditions.

### 1.3.1 Multiphysics Model

A multiphysics model of seismic, low-permeability fault-zones was constructed from nonlinear pore pressure diffusion, realistic fault zone architecture, pre-, co-, and post-failure permeability sub-models, as measured by triaxial deformation experiments with fluid flow, and fault-rock failure

models. As all simulations presented in this case study lie several orders of magnitude below the threshold for non-Darcy flow (Thauvin and Mohanty, 1998), fluid flow within the low-permeability medium was approximated using the equations for the non-linear diffusion of pore pressure within a classically porous medium (Silin, Korneev, & Goloshubin, 2003). Specifically, we select a model of fault zone architecture based on field observations of exhumed extensional faults in the Northern Apennines, which is analogous to that of the hypocentral fault of the 1997-98 Colfiorito seismic sequence (De Paola et al., 2008).

Both pre-, co- and post-failure permeability and associated failure envelopes of the fault rocks were approximated using measurements from triaxial deformation experiments with fluid flow on representative rocks of the fault core, retrieved from samples from the Perugia 2 and Fossonbrone 2 boreholes in the Umbria-Marche Apennines in Italy (De Paola et al., 2009). Failure by frictional sliding along the cohesionless gouges of the main principal slip zone was modelled using known friction laws (Byerlee, 1978).

Earthquake nucleation processes were also considered, by treating the fault-fluid ensemble as a non-smooth dynamical system, with transport and deformation properties evolving discontinuously at times. These earthquake nucleation processes govern the dynamics of the fault-fluid system from stable, non-acceleratory motion, when shear stress equals the fault shear strength, to the point at which unstable, accelerating oscillation begins on the fault, when the sliding patch has a size comparable to that of the critical (nucleation) length.

### 1.3.2 Numerical Simulation

During our numerical simulation, we apply the above multiphysics model to the loading during the interseismic period of a realistic, low-permeability fault zone. The range of physical conditions present at this fault are such that brittle or ductile mode of failure may occur within the fault core before and/or during the interseismic period and nucleation phase. These failure conditions ensure hysteresis in numerical simulation, in the sense that the physical state of the fault at a given time is dependent on the history of the fault and not just the current physical conditions at any given instant at the fault (e.g. a fault that undergoes failure in a spcific portion of the fault core can never return to its unfailed state, excepting hydrothermal healing). We can represent this brittle or ductile mode of failure behaviour in a state variable and treat the fault-fluid system mathematically as a non-smooth dynamical system. Not all the time derivatives with respect to physical variables (e.g. permeability) are well defined at the instant of these brittle of ductile mode of failure events.

The coupled partial differential equations (PDEs) that govern the multiphysics model can be discretized in space and time, resulting in a series of ordinary differential equations (ODEs) and a Jacobian matrix, whose eigenvalues might differ by up to 11 orders of magnitude. We select an ESDIRK method that can solve this highly stiff problem, efficiently and accurately enough to consistently resolve the nucleation phase (relative tolerance of 5E-9), specifically the MATLAB ODE23tb solver, an implementation of the ESDIRK23 algorithm (Bagterp Jørgensen and Rode Kristensen, 2018; Kristensen et al., 2004). This solver was also selected to be

capable of event detection to ensure that time integration was only performed directly for periods in which the time derivative with respect to all physical variables was continuous, and therefore that the problem was mathematically well-posed.

### 1.3.3 Parameter Studies

The inherent uncertainty in our understanding of subsurface fault systems, coupled with the evolution of their physical environment over time, means that any numerical simulation result must be robust to changes in these uncertain variables. We also need to constrain the behaviour we observe in these simulations over a broader range of values for each uncertain parameter. Hence, parameter studies allow us to expand our results to a much broader range of conditions, representative of those encountered in the brittle crust, which may affect seismic fault behaviours.

In this thesis, we use parameter studies to constrain fault behaviour during the interseismic period and earthquake nucleation phase, with respect to the following varying conditions and parameters:

- Variations in pore pressure within the reservoir at the fault core/damage zone boundary. The aim is to investigate the controls exerted on permeability and mode of failure within the fault core.
- Variations in the thickness of the overpressured reservoir. The aim is to investigate the controls exerted on the relative length of the failure patch in the fault core and the theoretically predicted nucleation length.

- Variation in the relative width of the fault zone sub-domains in the fault core, which should exert a primary control on the extent of failure, permeability evolution and magnitude of pore pressure in the fault core.

The above properties are all highly sensitive to uncertainty due to natural spatial or temporal variation, inaccuracy of subsurface measurement via indirect geophysical methods or approximated inference from outcrop analogues. Direct analysis of the impact of this uncertainty on simulation informs our understanding of the ability of our models to represent the likely behaviour of faults with poorly constrained properties or to generalise to other faults.

## 1.4 **Thesis Outline**

This thesis comprises the following chapters:

**Chapter 2**: This chapter includes a literature review of subsurface fluid flow, faulting and seismicity. Taking field examples of both natural and human subsurface fluid flow and subsequently induced seismicity.

**Chapter 3:** A description of the methodology adopted is presented in this chapter, a method for efficiently simulating fault zone pore pressure diffusion in the interseismic period with complex, realistic models of fault zone architecture and brittle and ductile modes of failure. The nucleation phase is simulated, and stable sliding and earthquake nucleation are resolved and distinguished to the order of seconds for several hundred years simulations.

**Chapter 4:** Results from a case study are presented and discussed in this chapter we model an analogue of the seismic sources at hypocentre depth of recent

seismic events in the Northern Apennines of Italy (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia earthquakes). Further, a parameter study of pore fluid factor and brittle and ductile mode of failure is presented. The format of this chapter is in journal-style" research chapters, as this is currently in review in EPSL.

**Chapter 5:** Results from a parameter study are presented and discussed in this chapter, we perform parameter studies on pore pressure diffusion and earthquake nucleation, with realistic models of ductile failure, varying the dimension of components of fault zone architecture and neighbouring lithology, outer fault core width and the height of pressurised layers abutting the fault core. The format of this chapter is in journal-style" research chapters, as this is currently in submission to JGR

**Chapter 6:** This chapter presents a discussion and conclusions of the research chapters, including future research.

# CHAPTER 2

*A literature review of fluid flow, faulting and seismicity in natural rocks*

# 2 A literature review of fluid flow, faulting and seismicity in natural rocks

## 2.1 Introduction

There is a large and growing body of evidence that both natural and human-induced subsurface fluid flow can drive faulting and seismicity. Supra-hydrostatic pore pressure gradients can drive fluid flow within fault zones, which can cause a reduction of the frictional strength of faults due to the increasing pore pressure. Fluid overpressure can basically make faults weaker and induce fault initiation (Cox, 2010; Sibson, 1990, 1992). However, a highly non-linear relationship exists between pore pressure and earthquake nucleation processes. In fact, reducing the effective normal stress of faults will increase the critical size of a rupture patch needed for unstable sliding to spread (an earthquake) (Campillo et al., 2001; Scholz, 1988; Uenishi and Rice, 2003). As overpressure acts to simultaneously weaken the fault while increasing the size of failure patch required for earthquake nucleation the relationship between overpressure and the timing of earthquake nucleation is not linear. No constant ratio exists between either interseismic period, nucleation phase length or the size of the rupture patch and the amount of overpressure found in the fault core.

Natural subsurface fluid flow has driven a number of large earthquakes (up to $M_W$ 9.0, Terakawa et al., 2013). Supercritical carbon dioxide ($CO_2$) from mantle degassing processes is thought to have driven many instances of natural seismicity. It is argued that supra-hydrostatic pore pressure gradients can be generated by $CO_2$ released from deep degassing processes (Di Luccio et al., 2010; Mahesh et al., 2012; Miller et al., 2004) and structural or lithological traps in the subsurface (Noir et al.,

1997; Nur and Booker, 1972; Parotidis et al., 2003; Terakawa et al., 2013; Yoshida et al., 2016). Migration paths of the $CO_2$ released can themselves depend on seismicity, as large volumes of stored $CO_2$ can be released coseismically triggering subsequent further seismicity (Keranen et al., 2013; Sumy et al., 2014).

There has been a recent increased interest in human-induced seismicity due to the exponential increase in the seismic rate observed in the continental United States. This has been associated with higher rates of hydraulic fracturing and wastewater storage operations (Ellsworth, 2013), occurring even in regions without any previous history of seismicity (Schultz et al., 2015). While only a small subset of these hydraulic fracturing or wastewater injection wells cause felt seismicity (Weingarten et al., 2015), the large number of them within the continental US represents a considerable hazard. Here, we review some case studies of natural and human-induced seismicity, drawing from geological, geophysical, mathematical and simulation-based analysis of observed seismicity and linking it causally to

subsurface fluid flow.

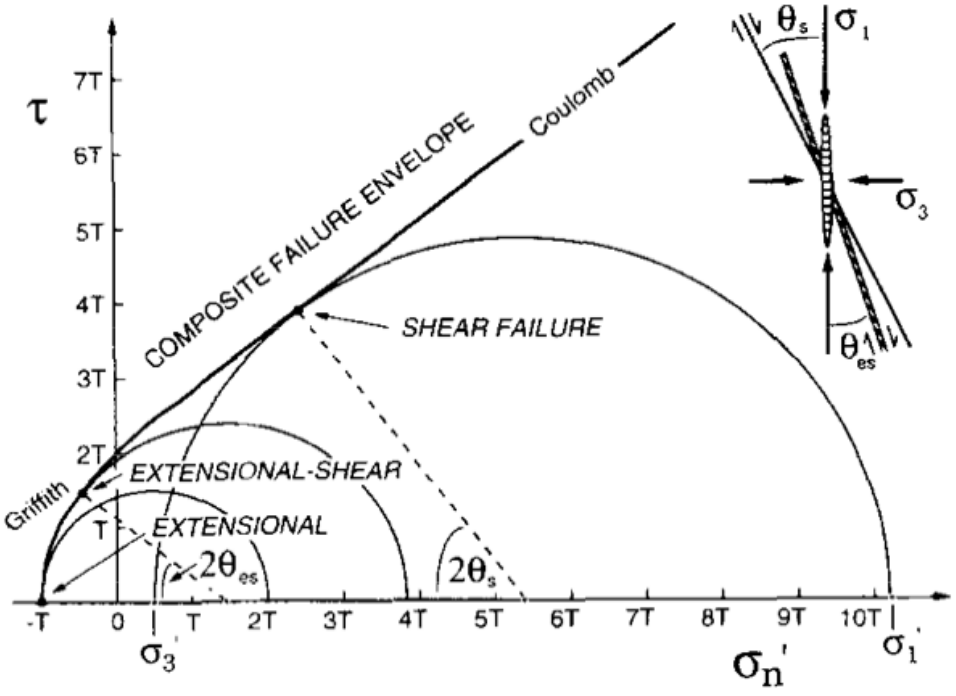



**Figure 2.1: "Mohr diagram with composite failure envelope for intact rock with tensile strength, T, illustrating the stress conditions and orientations with respect to the stress field of: (a) extensional failure; (b) hybrid extensional-shear failure; and, (c) compressional shear failure, for a particular rock-type." (Sibson, 1996)**

## 2.2 Fundamental principles of fault and earthquake mechanics

Intact rocks fail by the development of shear fractures, extensional fractures or hybrid extensional/shear fractures. Failure in these rocks initiates around randomly oriented microfractures (Griffith, 1924). Microfractures oriented parallel to the direction of maximum shear within the rock will fail first and act to control this transition. Mohr diagrams can be used to analyse this failure and the strength of intact rock (the stress at the point of failure, Fig 2.1).

The tensile strength of a rock T is the extensional stress at which the rock would fail. Similarly, the shear strength acting on a rock is the shear stress at which failure would occur. In a fluid-saturated rock, the normal stresses, $\sigma_N$, are reduced by the amount of pore fluid pressure, $P$, (Terzaghi, 1963) to give effective normal stress as

$$\sigma'_N = \sigma_N - P \qquad (2.1)$$

Considering effective stress in place of stress is a useful parameterization as linear elastic models of porous solids indicate that they deform in response to changes in effective stress rather than stress (Scholz, 2019).

The Griffith's criterion relates the shear strength of a rock to the effective normal stress acting on a rock and it's tensile strength (Griffith, 1924):

$$\sigma_N - P = \frac{\tau^2 - 4T}{4T} \qquad (2.2)$$

The Griffith criterion applies comprehensively to compressive, hybrid and tensile failure. This criterion is derived by considering the stress at which macroscopic failure would arise from the largest, most optimally oriented Griffith crack(Griffith, 1924). Griffith cracks are naturally occurring, microscopic cracks present in all natural rocks, occurring as a result of both weathering and formation."

For compressional failure, effective normal stress, i.e. $\sigma'_N > 0$, the shear failure of intact rocks can also be simplified to the Coulomb-Navier failure criterion (Sibson, 1996), where $\tau$ is the shear stress, $\mu_i$ is the coefficient of internal friction, $\sigma_N$ is the normal stress, $C = 2T$ is the cohesive strength and $P$ is the pore fluid pressure (Sibson, 1996)

$$\tau = C + \mu_i(\sigma_N - P) \qquad (2.3)$$

Most reservoir/seal rocks contain pre-existing faults, developed during previous deformation events, further increasing their structural complexity. From a mechanical point of view, faults are usually considered planes of shear failure without any cohesion and, according to Amonton's law, sliding will occur when the shear force on the fault exceeds the frictional forces acting on the fault, where $\mu_s$ is the sliding friction coefficient (Sibson, 1996)

$$\tau = \mu_s(\sigma_N - P) \qquad (2.4)$$

Rocks can undergo failure according to two main modes of failure, brittle or ductile. Both brittle and ductile failure are characterised by fracturing. During brittle failure the strain due to deformation is accommodated by a single extensive fracture, whereas during ductile deformation multiple distributed fractures each accommodate a smaller portion of the overall strain (Rutter, 1972). The brittle-ductile failure transition is controlled by confining pressure, and therefore by pore pressure through effective stress (Byerlee, 1978).

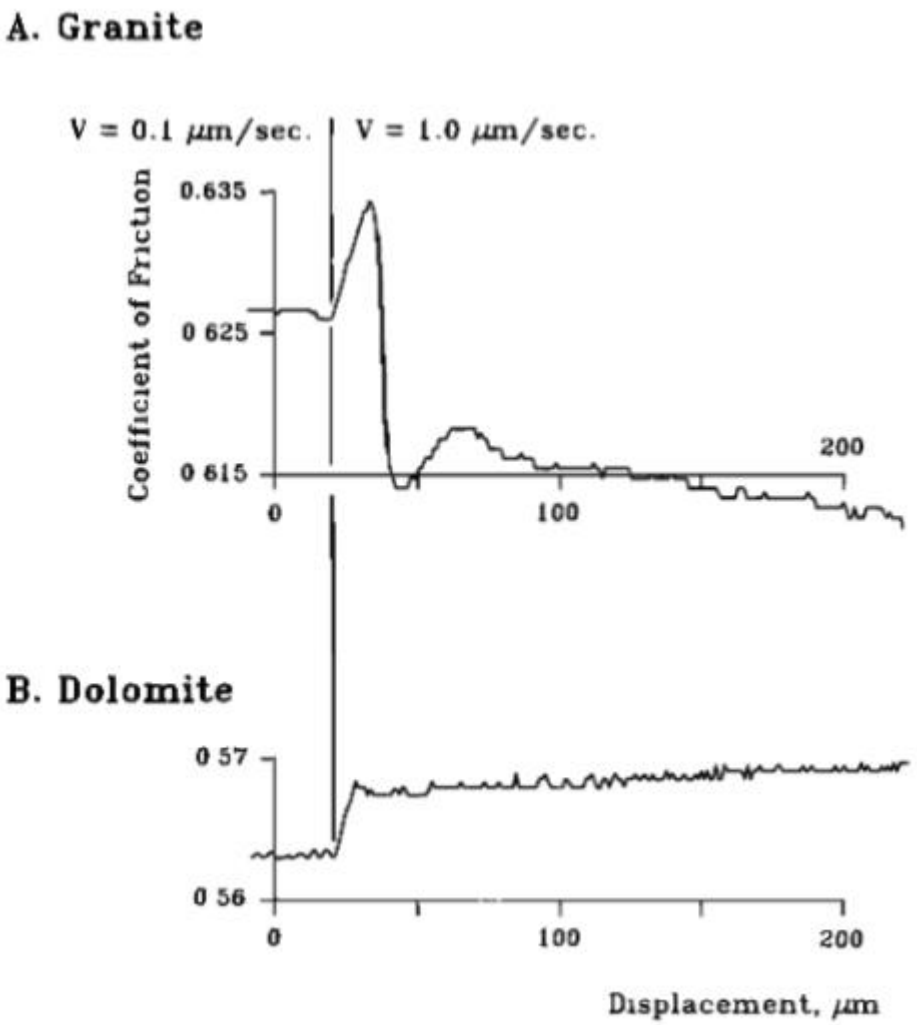



**Figure 2.2: "Comparison between granite and dolomite marble behaviour for a load point velocity jump from 0.1 to 1m/s. (a) Granite shows a transient increase in friction followed by a decay to a lower steady state frictional strength, response termed "velocity weakening." (b) Marble shows the same increase in frictional stress, with a small peak, but this is followed by a decay to a higher frictional strength, for an overall "velocity strengthening"." (Weeks and Tullis, 1985)**

Amonton's Law defines the value of shear stress required to initiate sliding along a fault surface for a given effective normal stress. However, rate and state theory predict the velocity-dependence of sliding friction, once sliding is initiated

(Dieterich, 1979). If subjected to a sudden change in sliding velocity, sliding friction coefficient evolves to a new steady value over a characteristic slip distances $D_c$ (Dieterich, 1979). The rate- and state-variable friction law describes the velocity dependence of sliding friction (Dieterich, 1979):

$$\tau = \left(\mu_0 + a\, ln\left(\frac{V}{V_0}\right) + b\, ln\left(\frac{V_0\theta}{D_C}\right)\right)\sigma'_N \qquad (2.5)$$

Where $\tau$ is shear stress, $V$ is velocity, $\mu_0$ is steady state friction at reference velocity $V_0$ and $D_C$ is the critical slip distance, $a$ and $b$ are material properties and $\theta$ the state variable which parameterises the physical state and evolution of the slip surface is given by:

$$\dot{\theta} = 1 - \frac{V\theta}{D_C}$$

For example, upon application of a sudden increase in velocity, sliding friction first increases (direct effect controlled by the rate and state parameter) then, decreases to a new steady state value (evolving according to the b rate and state parameter) (Fig. 2.2). This velocity dependence can be positive, in which case velocity strengthening behaviour ($a - b > 0$) will favour stable sliding, or negative ($a - b < 0$), in which case velocity weakening behaviour will favour frictional instability and earthquake nucleation (Scholz, 1998). The rock properties, ambient conditions and amount of slip/shear localisation control the velocity dependence of sliding friction (Marone, 1998).

Shear failure usually initiates at fault asperities, which are small fault patches, when shear stress exceeds the fault shear strength due to high shear stress or elevated pore fluid pressure reducing fault strength. Stable sliding initiates at these fault asperities and, in velocity weakening materials, can spread out with an

accelerating sliding velocity until it reaches a critical size. This stability limit is the nucleation length, $L_c$,

$$L_C = \frac{\zeta G D_C}{\sigma'_N (b-a)} \qquad (2.5)$$

where $G$ is the shear modulus, $\zeta$ is a constant of proportionality of order 1, $D_c$ is the critical slip distance, $a$ and $b$ are rate and state parameters (Dieterich, 1992; Rice and Ruina, 1983).

In velocity weakening materials, any slipping patch that extends beyond the nucleation length exhibits unstable behaviour leading to the nucleation of an earthquake. The nucleation length concept implies a complex relationship between pore pressure and induced seismicity. Increasing the pore pressure at a fault reduces the effective normal stress acting upon it and the fault strength (Eq. 2.4), bringing pressurised fault patches closer to failure. At the same time, there is a simultaneous increase in the nucleation length required to nucleate an earthquake, due to the inverse proportionality of $L_c$ to the effective normal stress. The nucleation length is a critical parameter controlling the nucleation of earthquakes. Nevertheless, it is a seismic parameter that cannot be directly measured and/or obtained from the inversion of seismological data.

### 2.3 Fault zone architecture and the role of pore fluid pressure during faulting

Pore fluid pressure reduces the frictional strength of faults and, hence, plays a primary role during faulting processes (Cox, 2010; Sibson, 1990, 1992). Indeed, there is a body of geological (Collettini et al., 2009; Hickman et al., 1995; Sibson, 1992), geophysical (De Pater and Baisch, 2011; Miller et al., 2004; Miller, 1996; Rice, 1992; Sibson, 2000, 1992) and numerical (Cappa and Rutqvist, 2011b, 2011a, 2012; Mazzoldi et al., 2012; Rinaldi et al., 2014a; Rutqvist et al., 2007, 2002, 2016,

2015, 2013b, 2013a, 2009) evidence showing that fluid migration in the upper crust controls faulting. Fluid circulation within the crust is strongly dependent on the rock transport properties (i.e., permeability), and their evolution with pressure (De Paola et al., 2009; Fischer, 1992; Morrow and Lockner, 1997, 1994; Zhu et al., 2007) and deformation, which itself controls the development and connectivity of fracture patterns across a range of scales (Caine et al., 1996; Cox, 1995; De Paola et al., 2009; Faulkner and Rutter, 2001; Mitchell and Faulkner, 2008; Peach and Spiers, 1996; Zoback and Byerlee, 1975).

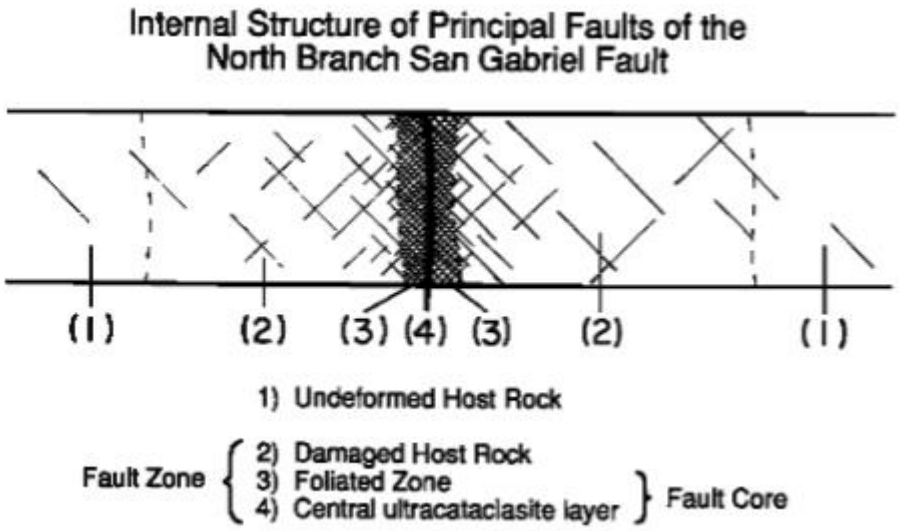



**Figure 2.3: "Schematic section across the North Branch San Gabriel fault illustrating position of the structural zones of the fault. The diagram is not to scale." (Chester et al., 1993)**

Tectonic faults are zones of finite width, whose internal architecture can be described by discrete and juxtaposed discrete fault zone domains (Chester et al., 1993; Faulkner et al., 2010): the protolith, the damage zone and the fault core (Fig. 2.3). The fault core is the centre of the fault zone where most of the displacement is

accumulated. The main structural features in the fault core are principal slip zones and principal slip surfaces, due to shear localization within volumes of fine- to very fine-grained fault gouges, cataclasites and ultracataclasites (Sibson, 1977 JGSL). On both sides of the fault core, a damage zone is usually present (Fig. 2.3), which is made of the network of subsidiary fracture patterns. Relatively little amount of slip is accommodated within the damage zone, where the main structural features are network of fractures, veins and subsidiary small shear fractures. Damage intensity and density decreases as one move away from the fault core, towards the protolith. Fault breccias are the main fault rocks found in the damage zone. Finally, the protolith is the original rock source of those fault rocks found in the damage zone and fault core. There is no damage or faulting in the protolith related to the specific fault zone activity, although background damage and fracturing may be presented in the protolith due to ancient, previous faulting events.

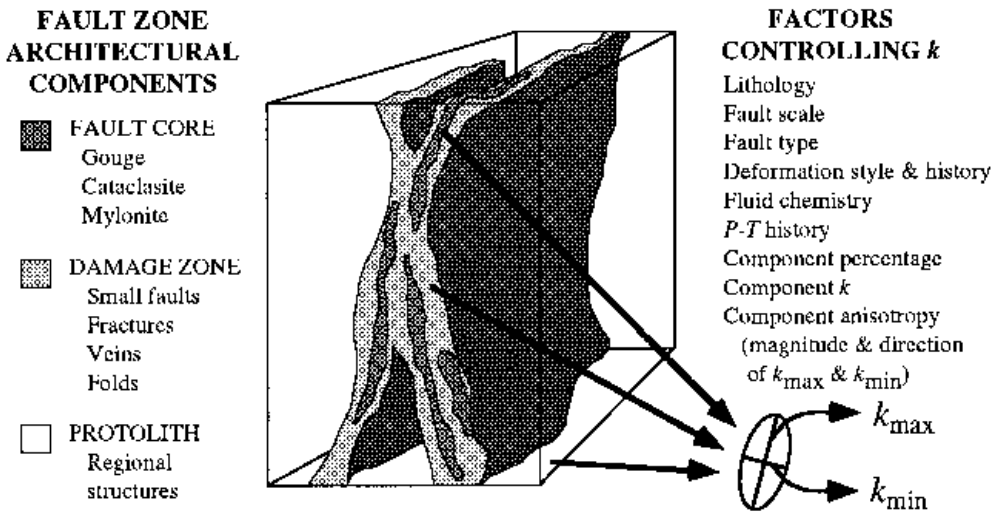



**Figure 2.4: "Conceptual model of fault zone with protolith removed (after Chester and Logan, 1986; Smith et al., 1990). Ellipse represents relative magnitude and orientation of the bulk two-dimensional permeability (k ) tensor**

**that might be associated with each distinct architectural component of fault zone." (Caine et al., 1996).**

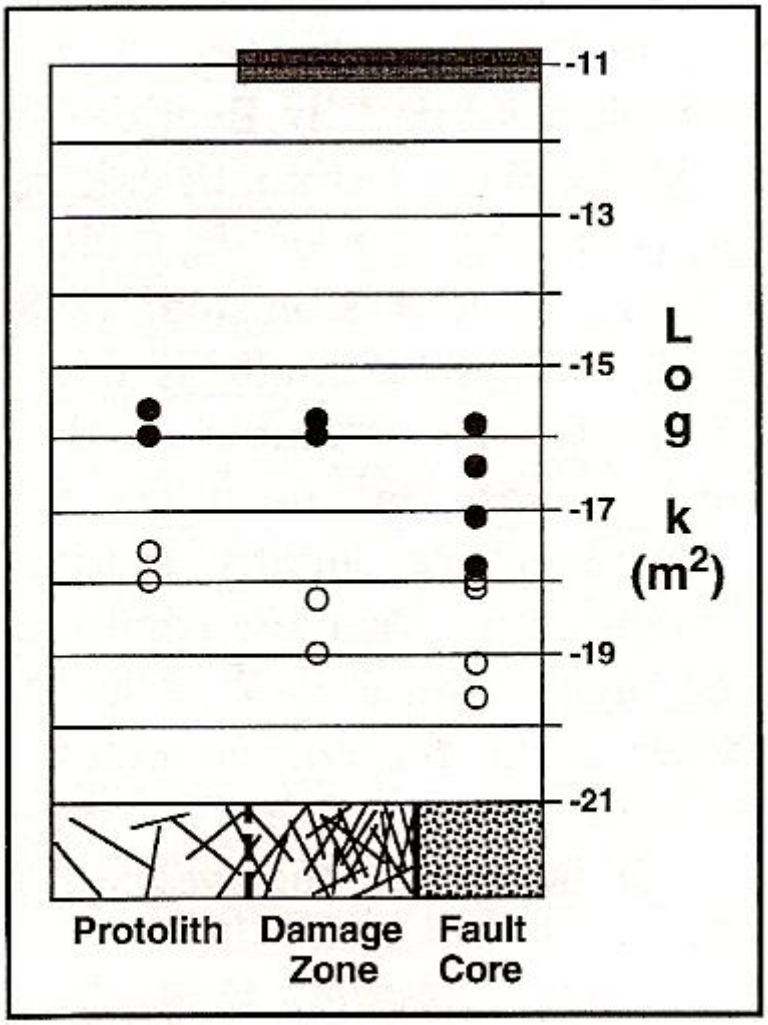



**Figure 2.5: "Summary of laboratory permeability data obtained at Pe=15 MPa (closed circles corresponding to a depth of approximately 1 km of overburden under hydrostatic pressure) and Pe=90 MPa (open circles corresponding to a depth of approximately 5 km) as a function of position within the fault zone. In situ estimates made by Barton et al. (1997)at a depth of 2.5 km are shown as the shaded bar that spans the damage zone and fault core." (Seront et al., 1998)**

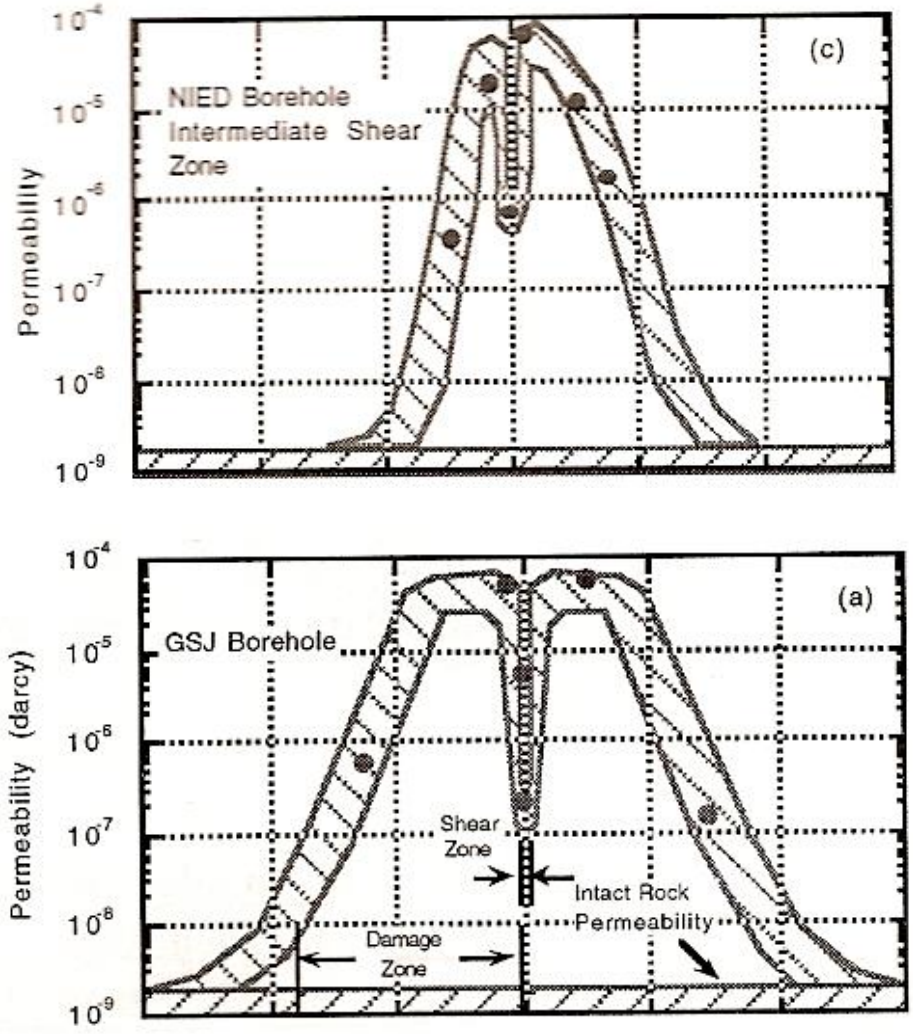



**Figure 2.6: "Profiles of matrix permeability measured at 50 MPa effective confining pressure. The three upper fault crossings show a low permeability fault core (fine-grained material containing some clay fraction) surrounded by high permeability damage zones (interlocked grains with numerous open microfractures). The deep shear zone is partially sealed and was apparently not activated by the Kobe earthquake." (Mizoguchi et al., 2008)**

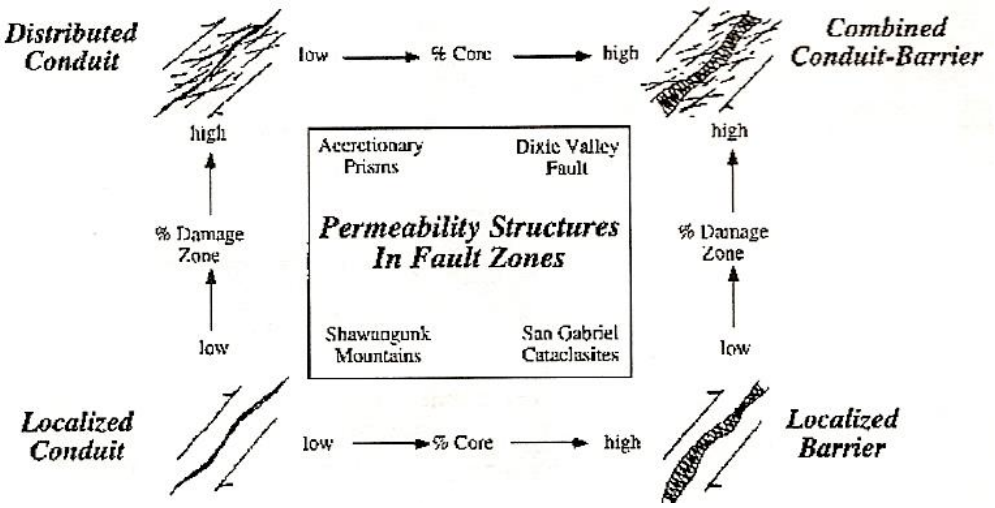



**Figure 2.7: "Conceptual scheme for fault-related fluid flow." (Caine et al., 1996)**

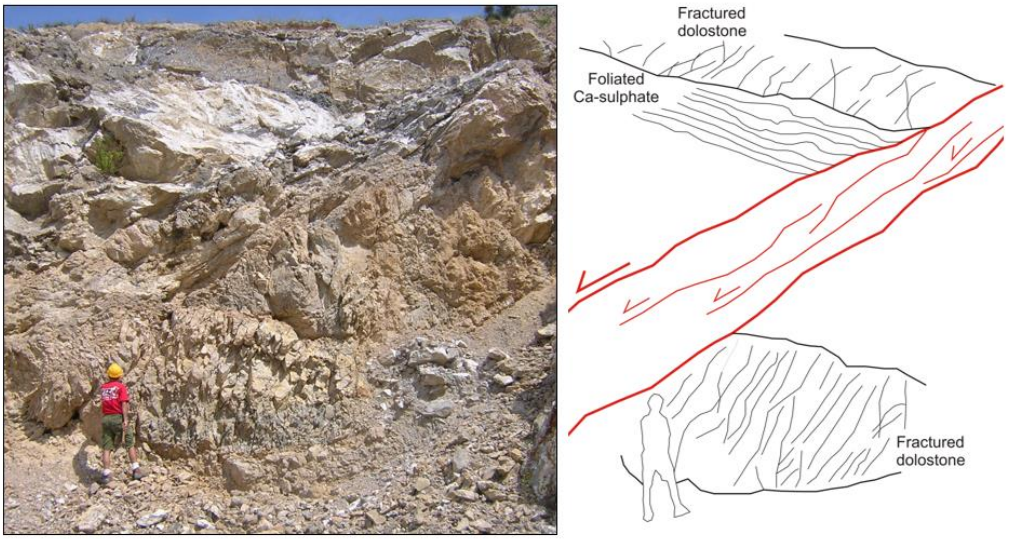



**Figure 2.8: (Left) Macroscopic large fault zone structure of the Roccastrada outcrop. (Right) Line drawing of the fault zone shown displaying the internal fault core architecture.**

The general model of fault zone architecture as comprising a fault core and a damage zone is a useful concept, despite the complexity and diversity of real faults. Different suites of fault rocks in the different damage zone domains have different transport properties (Fig. 2.5-6). Variations in the relative thickness of the damage zone and fault core exert a primary control over fluid flow across and along fault zones (Fig. 2.3, 7) (Caine et al., 1996; Caine and Forster, 1999). Deformation within the damage zone is in part controlled by the mode of failure. Either brittle deformation, elastic-frictional behaviour (localised deformation by discrete faulting) (Sibson, 1977) or ductile failure (fracturing distributed on the mesoscopic scale) (Rutter et al., 1986), e.g. ductile failure in the Roccastrada damage zone (Fig. 2.8; De Paola et al., 2008). Experimental measurements of gross fault zone permeability has shown it to increase with increasing relative damage zone width (Caine et al., 1996). While the combination of relative fault core and damage zone width can be used to group fault zones into four coarse groups with respect to fluid flow: distributed conduit, localized conduit, combined conduit-barrier, localized barrier (Fig. 2.7; Caine et al., 1996).

The deformation patterns developed within each fault domain control fluid flow across and along fault zones and, hence, fault zone architecture can further control the onset and duration of earthquake nucleation and rupture properties. This is an aspect of the earthquake nucleation process that is still poorly investigated and understood.

During industrial hydraulic fracturing operations, extensional and hybrid extensional-shear fracture systems are intentionally produced in tight reservoir rocks,

by pumping high-pressure fracking fluid and proppant into intact rock. The main aim is to increase the permeability of otherwise tight, low permeability reservoir rocks to result in an enhanced recovery of hydrocarbons. In this case, the pore pressure levels induced in the stimulated reservoir should satisfy the extensional and extensional-shear (Eq. 2.2) failure criterion for intact rocks, but it should not meet the conditions that favour either shear failure in intact rocks (Eq. 2.3) or fault reactivation in pre-existing faults (Eq. 2.4).

During industrial carbon sequestration operations, supercritical carbon dioxide is pumped into sealed lithological units in the subsurface. The pore pressure levels in the reservoir/seal system should always be below those values required to induce any failure, as predicted by Eq. 2.1 - 2.3. Predictions of the brittle or ductile mode of failure that occurs in the intact rocks of a natural reservoir depend on the balance between the differential stress, the tensile strength and coefficient of internal friction, and the level of pore fluid pressure. The intrinsic properties of rocks (e.g. $T$, $\mu_i$) are reasonably well known. However, the initial stress state in a reservoir and the evolution of rock permeability with pore fluid pressure are fundamental unknowns. The pore pressure perturbation at the local and field-scale depends on the level of connectivity of the fault/fracture patterns.

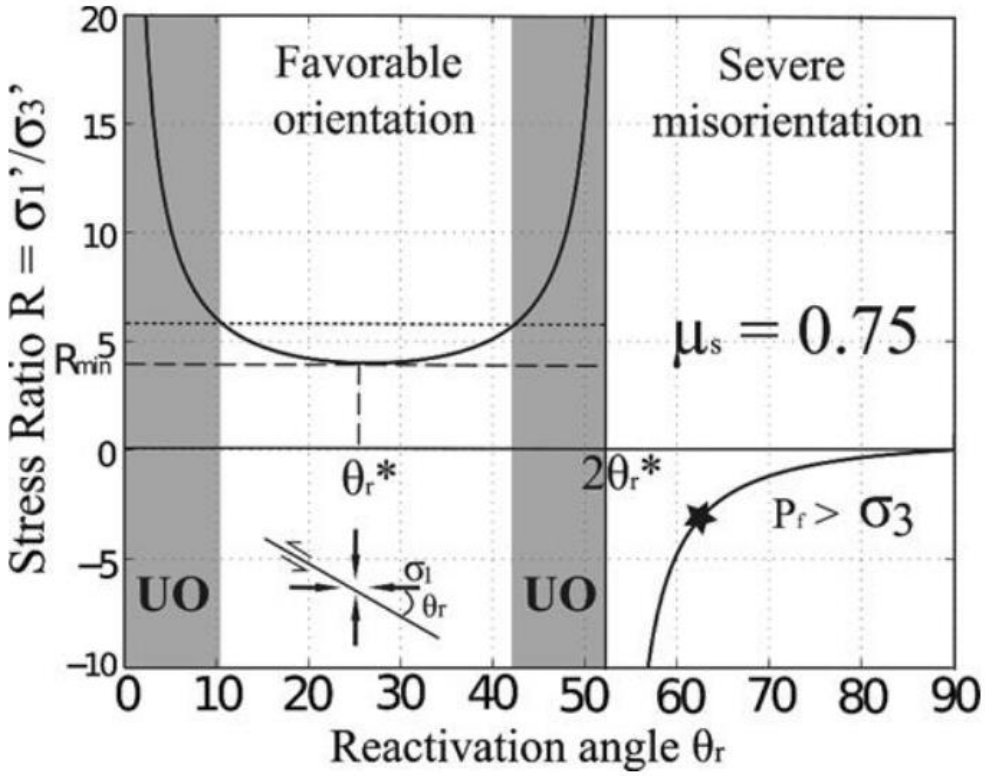



**Figure 2.9: "Variation of the effective stress ratio, R =σ1′/s3′ as a function of the coefficient of static friction, μs, with a reactivation angle θr of 63°. The light grey shaded area defines the domain where the fault is favourably oriented, the grey area where the fault is unfavourably oriented (UO) and the white area where the fault is severely misoriented." (Leclère et al., 2012)**

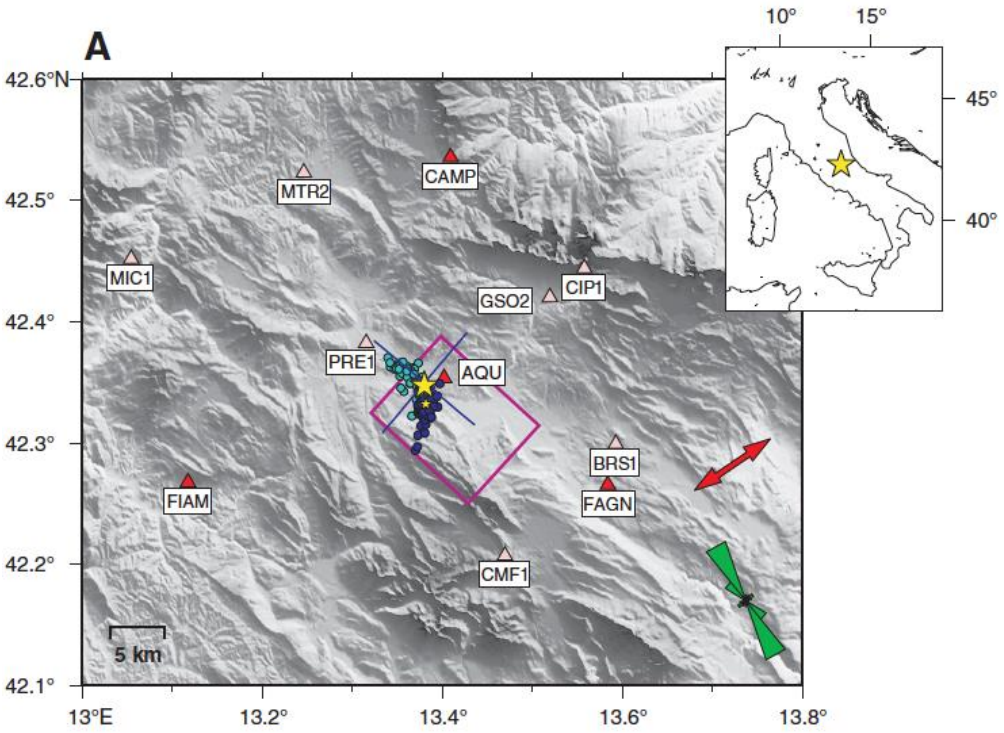



**Figure 2.10: "Foreshocks of 6 April 2009 Mw 6.3 L'Aquila earthquake. Light blue dots represent earthquakes that occurred from January to 30 March 2009. Dark blue dots indicate earthquakes that occurred from 30 March to the main shock. Smaller yellow star is $M_L$ = 4 foreshock that occurred on 30 March. Larger yellow star is main shock hypocenter. Triangles are seismic stations, used to localize earthquakes, belonging to Istituto Nazionale di Geofi sica e Vulcanologia national (red triangles) and regional (pink triangles) permanent seismic networks. Purple box is uniform slip fault (Atzori et al., 2009). Traces of cross sections are represented by blue lines. Green rose diagram represents frequency distribution of splitting fast directions measured at station AQU (length of each petal is proportional to number of measures in each direction interval). Red arrow indicates direction of minimum horizontal stress in area (from Montone et al., 2004). Star in inset is location of main shock on map of Italy Foreshocks of 6 April 2009 $M_w$ 6.3 L'Aquila earthquake. Light blue dots**

**represent earthquakes that occurred from January to 30 March 2009. Dark blue dots indicate earthquakes that occurred from 30 March to the main shock. Smaller yellow star is ML = 4 foreshock that occurred on 30 March. Larger yellow star is main shock hypocenter. Triangles are seismic stations, used to localize earthquakes, belonging to Istituto Nazionale di Geofi sica e Vulcanologia national (red triangles) and regional (pink triangles) permanent seismic networks. Purple box is uniform slip fault (Atzori et al., 2009). Traces of cross sections are represented by blue lines. Green rose diagram represents frequency distribution of splitting fast directions measured at station AQU (length of each petal is proportional to number of measures in each direction interval). Red arrow indicates direction of minimum horizontal stress in area (from Montone et al., 2004). Star in inset is location of main shock on map of Italy." (Lucente et al., 2010a)**

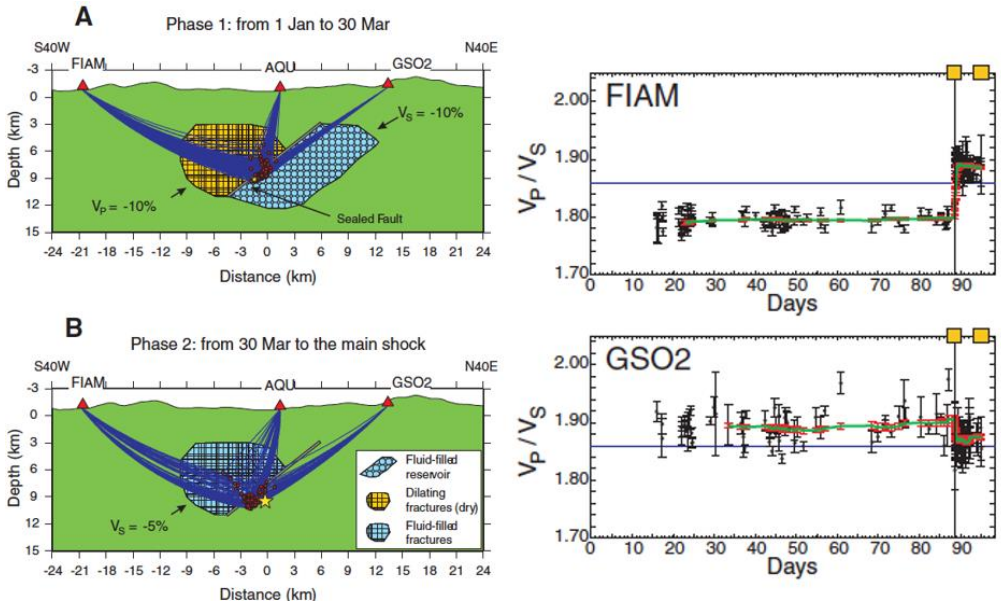



**Figure 2.11: "A: Vertical section across VP/VS (ratio between compressional-wave and shear-wave velocity) synthetic model for conditions before 30 March. B: Vertical section across VP/VS synthetic model for conditions after 30 March.**

**Green indicates unperturbed volume of model (Table DR1 [see footnote 1]). Red triangles are seismic stations. Red dots are hypocenters of foreshocks. Blue curves indicate seismic wave paths. Orange and light blue filled areas represent P- and S-wave velocity anomalies, respectively. Fault is represented by thick gray line. Smaller star on panel A indicates location of $M_L$= 4 foreshock; large star in panel B is main shock hypocenter. C: Comparison between time series of synthetic (left) and observed (right) VP/VS values." (Lucente et al., 2010a)**

As the frictional strength of a fault is typically lower than that of intact rocks, this means that favourably oriented faults will undergo shear failure while subject to less stress (Fig. 2.9. Quantitative predictions of the critical pore pressure values leading to reactivation of pre-existing faults in stimulated reservoirs (hydraulic fracturing) or reservoir/seal systems (carbon capture and storage) are routinely performed based on the application of Amonton's Law (Fig. 2.9). These predictions are affected by the same unknowns as those of intact rocks, as well as further uncertainties due to the lack of information about fault zone dimensions and internal structure, which control fluid circulation and stress/pore pressure perturbations.

### 2.4 Pore pressure and natural seismicity

There are many examples in the literature of case studies where natural fluid migration in the upper crust control faulting and earthquake processes. It is inferred that compressible fluids released from structural or lithological traps can cause subsurface pore pressure diffusion waves, which can drive seismicity (Noir et al., 1997; Nur and Booker, 1972). These lithological traps are themselves previously fed by deep, high-pressure sources of carbon dioxide ($CO_2$), which can be released by crust/mantle degassing processes (Parotidis et al., 2003; Terakawa et al., 2013; Yoshida et al., 2016) and mobilized by main earthquake and aftershock events (Di Luccio et al., 2010; Miller et al., 2004). The increasingly higher resolution of seismic tomography data can further enhance pressure diffusion analysis at the regional scale for some major seismic sequences, e.g. $M_w$ 6.3 2009 L'Aquila earthquake (Lucente et al., 2010b) and the Chamoli Region, Garhwal Himalayas (Mahesh et al., 2012). Approaching the $M_w$ 6.3 2009 L'Aquila, the elastic properties of rocks in the fault region underwent a sharp change about a week before the earthquake. This was used to infer that a complex sequence of dilatancy-diffusion processes takes place and that

fluids play a key role in the fault failure process (Fig. 2.10-11;Lucente et al., 2010a). Pressure diffusion analysis of earthquake events in the Chamoli region of the Central Himalayas (1999 $M_B$ 6.3, 2005 $M_B$ 5.3, and 2011 M 4.6.), suggested the presence of fluids percolating from depth, likely from metamorphic dehydration (Mahesh et al., 2012).

Natural fluid flow, faulting and earthquake processes have also been examined using numerical simulation techniques. For example, physical models of subsurface fluid flow resulting from the upwelling of carbon dioxide, $CO_2$, have been connected to seismicity (Cappa et al., 2009; Miller et al., 2004).

A correlation between reconstructed pore pressure diffusion waves and the distribution and timing of seismicity was established for the 1966 Parkfield-Cholame, California (Nur and Booker, 1972) and 1989 Dobi, Afar (Noir et al., 1997) seismic sequences. The analysis of pressure diffusion wave correlation can be extended to consider the frictional properties of specific faults involved in a seismic sequence. For instance, the 2011 $M_W$ 9.0 Tohoku-Oki earthquake lead a swarm of earthquakes where both favourably and unfavourably oriented faults were correlated to a fluid pressure-dependent frictional strength (Yoshida, Hasegawa, and Yoshida, 2016), suggesting that crust/mantle degassing fluid upwelling and subsequent migration after the initial $M_W$ 9.0 event initiated later earthquakes (Terakawa et al., 2013).

Similar pressure diffusion analysis has been used to link compressible fluids released from crust/mantle degassing processes to a number of instances of natural large events and associated aftershock sequences. For example, kilometre-scale pore pressure diffusion simulations correlated diffusivity to the recurrence of the nine swarms from the 2000 Vogtland/NW-Bohemia earthquake swarm at the

German/Czech border, suggesting the release of overpressured carbon dioxide ($CO_2$) (Parotidis et al., 2003). $CO_2$ pore pressure released from deep underlying metasomatized mantle wedge was invoked as the primary controlling factor in the time and space distribution of the 2009 $M_W$ 6.3 L'Aquila seismic sequence (Di Luccio et al., 2010).

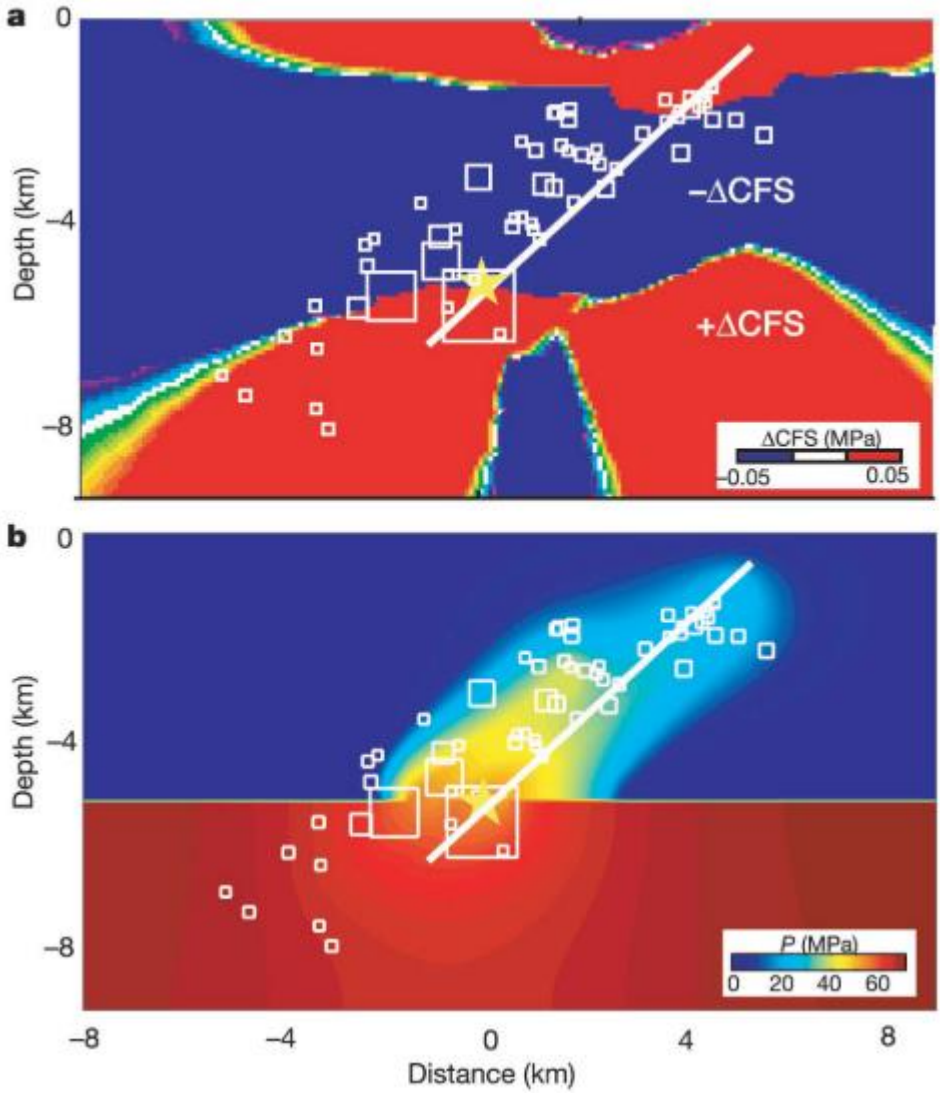



**Figure 2.12: "Comparison of aftershock data to stress changes in the DCFS formulation and pore pressure changes. a) There is no correlation between positive or negative DCFS regions and the aftershock locations. In contrast, b,**

**the same aftershock data compared to the calculated fluid pressure state after 11 days, shows a very strong correlation with the entire aftershock sequence."**

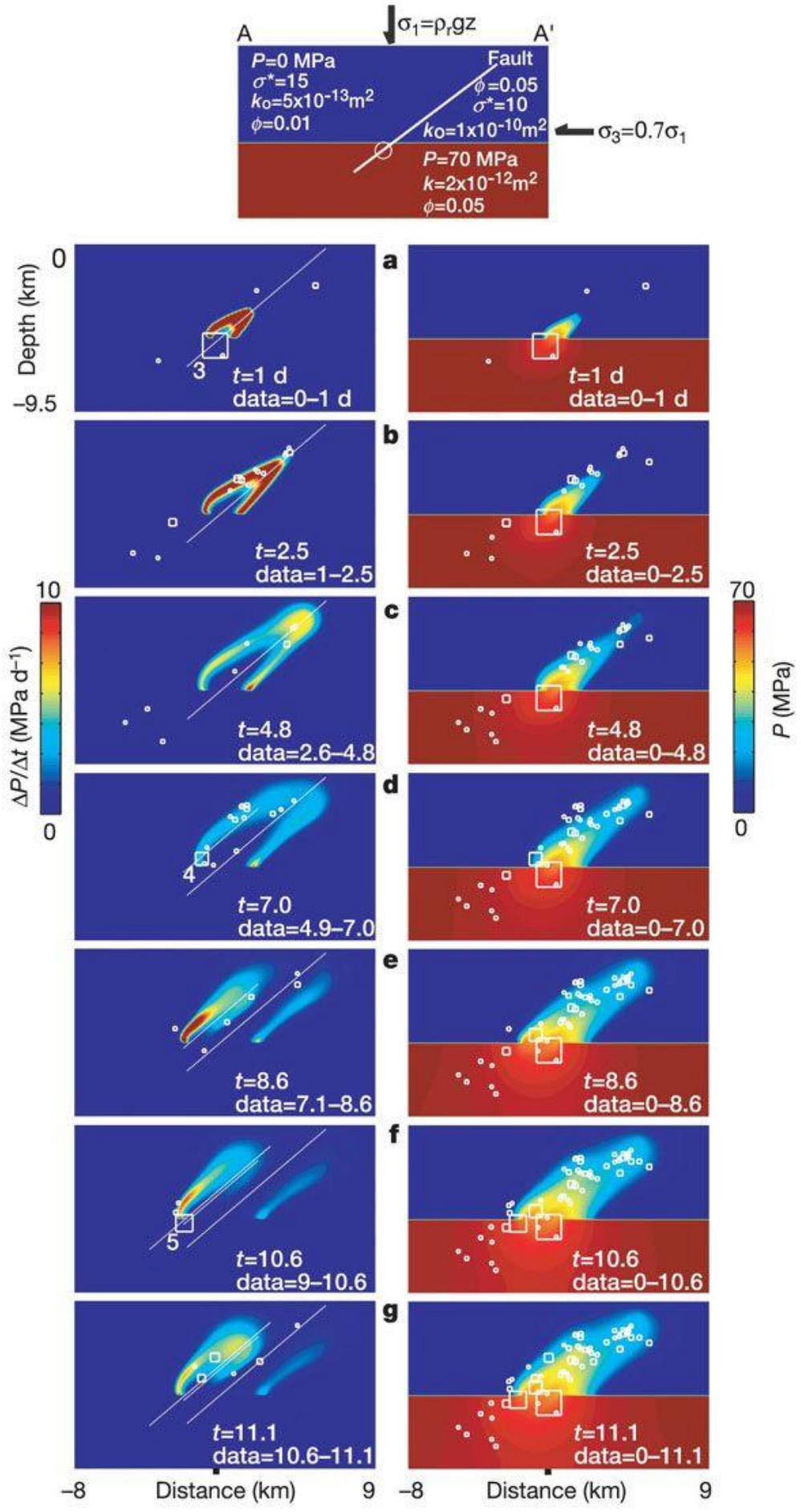

A
A'
σ1=ρfgz
P=0 MPa
σ*=15
ko=5x10-13m2
φ=0.01
Fault
φ=0.05
σ*=10
ko=1x10-10m2
σ3=0.7σ1
P=70 MPa
k=2x10-12m2
φ=0.05
Depth (km)
0
–9.5
a
b
c
d
e
f
g
t=1 d
data=0–1 d
t=2.5
data=1–2.5
data=0–2.5
t=4.8
data=2.6–4.8
data=0–4.8
t=7.0
data=4.9–7.0
data=0–7.0
t=8.6
data=7.1–8.6
data=0–8.6
t=10.6
data=9–10.6
data=0–10.6
t=11.1
data=10.6–11.1
data=0–11.1
3
4
5
10
0
ΔP/Δt (MPa d-1)
70
0
P (MPa)
–8
Distance (km)
9

**Figure 2.13: "Comparison of model results with initial conditions (top) to the hypocentres of aftershocks (bottom)" ... "a–g, Model results plotted as the rate of pore pressure increase to highlight propagation of the pressure front (left column), and the corresponding evolution of the entire fluid pressure field (right column). The left column compares the evolution of the pore pressure front to the aftershocks occurring during the times indicated. The overall fluid pressure field is superposed with the cumulative aftershock catalogue. The largest event in the sequence (event 3) and subsequent large aftershocks in the hanging wall (events 4 and 5) are indicated in a, d and e." (Miller et al., 2004)**

Pressure diffusion analysis was combined with seismic wave velocity data and the pattern of earthquake events in the Chamoli region of the Central Himalayas (1999 $M_B$ 6.3, 2005 $M_B$ 5.3, and 2011 M 4.6.). This analysis suggested the presence of fluids percolating from depth, likely from metamorphic dehydration of the Indian Crust in the Chamoli Region of the Central Himalayas (Mahesh et al., 2012).

The above linear models of pressure diffusion reconstruction can be refined using more precise non-linear models of rock transport properties in numerical simulations. These techniques have been applied in the Northern Apennines in Italy, where it has been suggested that fluids released coseismically from a deep source may have driven the $M_W$ 6.0 1997-98 Colfiorito seismic sequence (Fig. 2.12-13) (Miller et al., 2004). Miller et al. (2004) show that the timing and location of aftershock events strongly correlate with a kilometre scale simulation of the nonlinear diffusion of a 20 MPa pore pressure pulse. The results of their simulations suggest that the $M_W$ 5.7 mainshock may have released overpressured fluids and created damaged regions for these fluids to propagate through, subsequently triggering the aftershocks.

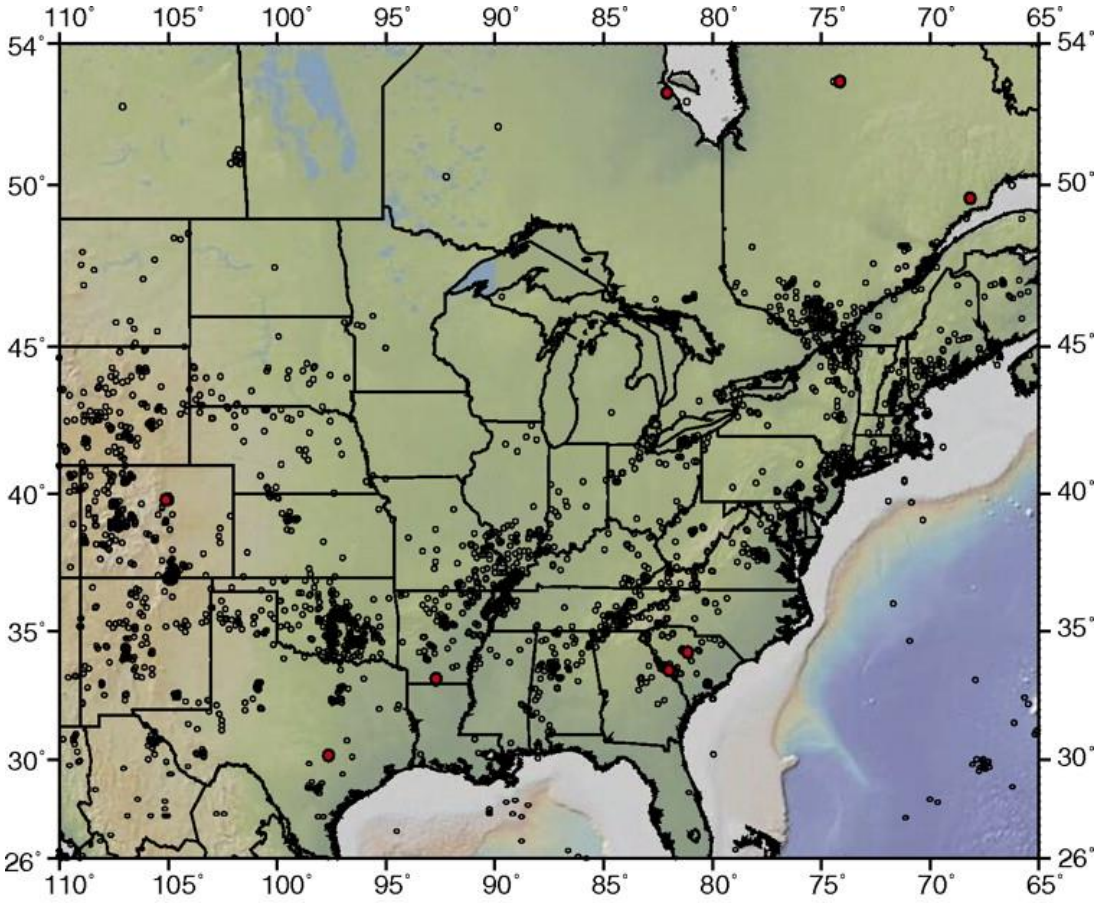
Numerical simulations coupling multiphase flow and geomechanical modelling code TOUGH-FLAC (Rutqvist et al., 2002) have also been used to validate the hypothesis that natural fluid flow may trigger earthquakes. For instance, Cappa et al., 2009 argued that the fluid pressurisation of upwelling, deep $CO_2$ rich fluids triggered the 1965-1967 Matsushiro earthquake swarm. This seismic swarm comprised more than 700,000 earthquakes, in part fed by an inferred two order of magnitude increase in permeability of the earthquake rupture patch.

### 2.5 Pore pressure and human induced seismicity

Contemporary seismic and microseismic ($M_W < -1$) measurements of human subsurface fluid injection have demonstrated the link between injection activities and induced seismicity. The careful analysis and monitoring of carbon sequestration (Zoback and Gorelick, 2012), enhanced oil recovery (Gan et al., 2013), wastewater injection (Ake et al., 2005; Frohlich, 2012; Frohlich and Brunt, 2013; Hornbach et al., 2016; Keranen et al., 2014, 2013; Kim, 2013) and hydraulic fracturing operations has allowed the collection of dense and high resolution seismic records. In particular, large volumes of data and information have been gathered about human-induced microseismicity (Baisch et al., 2009; Dicelis et al., 2017; Maxwell et al., 2002; Rutledge et al., 2004; Rutledge and Phillips, 2003; Vermylen and Zoback, 2011), seismicity (Atkinson et al., 2015, 2016; Bao and Eaton, 2016; Clarke et al., 2014; Davies et al., 2013; De Pater and Baisch, 2011; Elsworth et al., 2016; Farahbod et al., 2015b, 2015a; Friberg et al., 2014; Holland, 2013; Lei et al., 2017; McGarr, 2014; Rutqvist et al., 2013b; Skoumal et al., 2015; Sumy et al., 2014) and cascading seismicity, where coseismic Coulomb stress transfer from fluid induced earthquakes triggers further seismicity (Keranen et al., 2013; Sumy et al., 2014). The link between human subsurface fluid injection and seismicity is also apparent in

continent-scale measurements of induced seismicity in the USA, showing a sharp deviation from the typical trend after the onset of hydraulic fracturing operations (Fig. 1.1) (Ellsworth, 2013; Weingarten et al., 2015). The issue of induced seismicity has become sufficiently prominent in the continental United States that proposals for mitigation of the seismic hazard have even been forwarded (McGarr et al., 2015).

Analytical solutions to the corresponding physical equations (Shapiro and Dinske, 2009) and numerical simulations of fluid flow and faulting offer further support to the connection between human subsurface fluid injection and seismicity. In particular, coupled deformation and fluid flow models such as TOUGH-FLAC (Cappa and Rutqvist, 2011a, 2012; Rutqvist et al., 2016, 2015, 2013a, 2002) have been developed and employed to carry out such analyses. Numerical simulations, which couple fluid flow and geomechanical fault slip (e.g. fault reactivation), model the spatial evolution of both in situ stresses and fluid pressure. These models can be specifically used to estimate the maximum sustainable injection pressure during geological sequestration of $CO_2$ (Rutqvist et al., 2007), or parameter studies to determine reservoir properties which pose the most significant risk for seismicity (Mortezaei and Vahedifard, 2015). The above models have been applied from the metre to kilometre scale, and simulations have been extended to include the effects of simple fault zone architectures (Cappa and Rutqvist, 2011a; Mazzoldi et al., 2012; Rinaldi et al., 2014a; Rutqvist et al., 2013b, 2009) and effective normal stress dependent permeability (Cappa and Rutqvist, 2011b; Hsiung et al., 2005).


110° 105° 100° 95° 90° 85° 80° 75° 70° 65°
54° 50° 45° 40° 35° 30° 26°

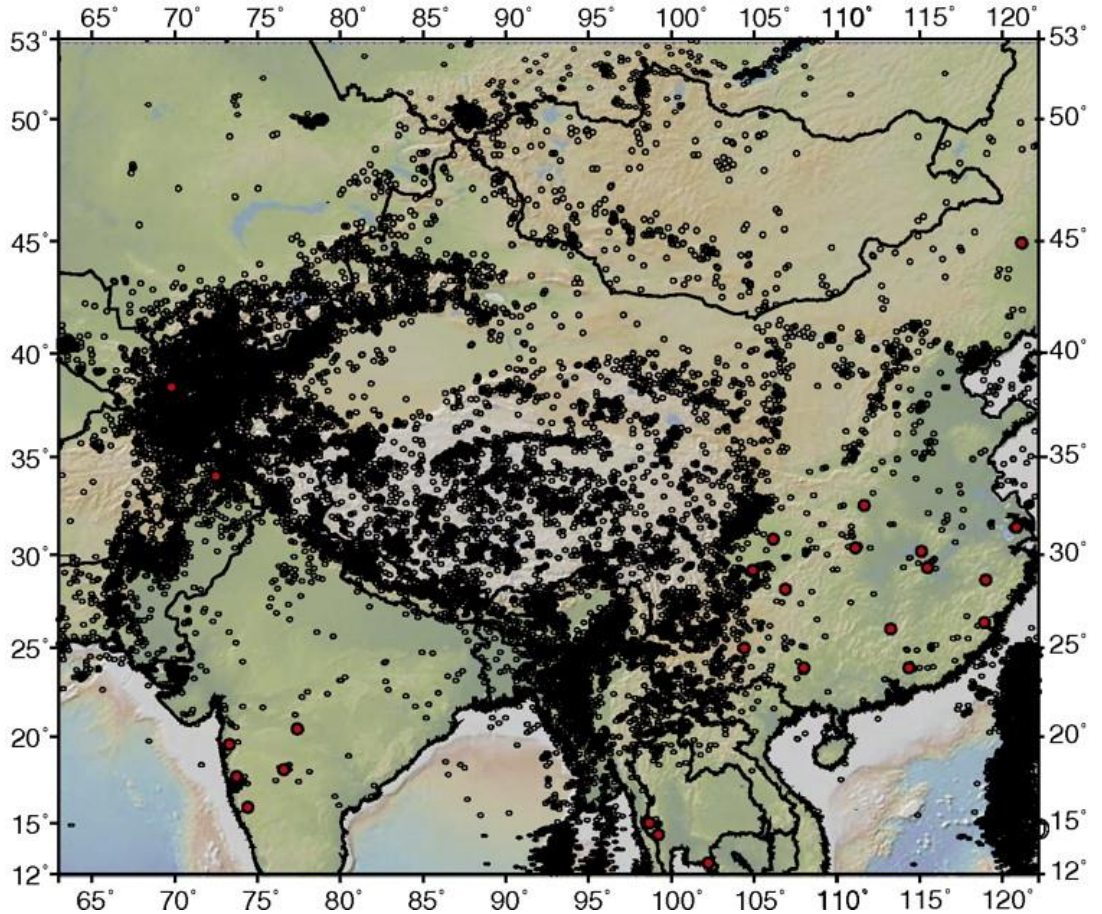

65° 70° 75° 80° 85° 90° 95° 100° 105° 110° 115° 120°
53° 50° 45° 40° 35° 30° 25° 20° 15° 12°

**Figure 2.14: "Upper: Instrumentally recorded seismicity and damaging historical earthquakes in the central and eastern United States and southeastern Canada. Red dots indicate sites of reservoir-induced seismicity. Lower : Seismicity of south and east Asia and sites of reservoir-induced seismicity." (Zoback and Gorelick, 2012).**

### 2.5.1 Seismicity induced by subsurface carbon dioxide injection

A number of human industrial activities involve the injection of supercritical $CO_2$ into the subsurface, which can induce seismicity, particularly injection as part of enhanced oil recovery and carbon sequestration. For instance, injection at the Cogdell oil field in Texas lead to seismicity exceeding $M_W$ 3.0 (Gan et al., 2013). The period of greatest seismicity 2006-2011 accompanied the injection of $CO_2$, while less significant seismicity occurred in the period of highest net water injection 1957-1982. Furthermore, a combined dataset of induced seismicity in hydrocarbon reservoirs from both the continental US and Asia and stress measurements, from the pilot carbon sequestration site at the Teapot Dome, Wyoming, establish the link between human subsurface injection of $CO_2$ and seismicity (Fig. 2.14) (Zoback & Gorelick, 2012).

### 2.5.2 Seismicity induced by wastewater injection into deep saline aquifers

Wastewater disposal by injection into deep saline aquifers is thought to be the primary cause of induced seismicity in the continental USA, driving an exponential increase of seismicity over the last two decades (Fig. 1.1) (Ellsworth, 2013). Wastewater disposal operations are typically carried out alongside hydraulic fracturing operations to dispose of flow-back fluids from recently treated wells. In the three years from 2010-2012, the continental USA experienced 300 seismic events $M_W \geq 3$ compared to an average of 21 per year in the period 1967-2000. Most of

these large seismic events occurred within 10 km of the location of wastewater disposal wells. The largest seismic events identified occurred beneath the injection interval, suggesting that increased pore pressure in the basement, transmitted from the injection site, presents the greatest seismic risk (Fig. 1.1) (Ellsworth, 2013). A series of regional studies of seismicity during this same period have more precisely examined and confirmed the link from wastewater disposal to seismicity.

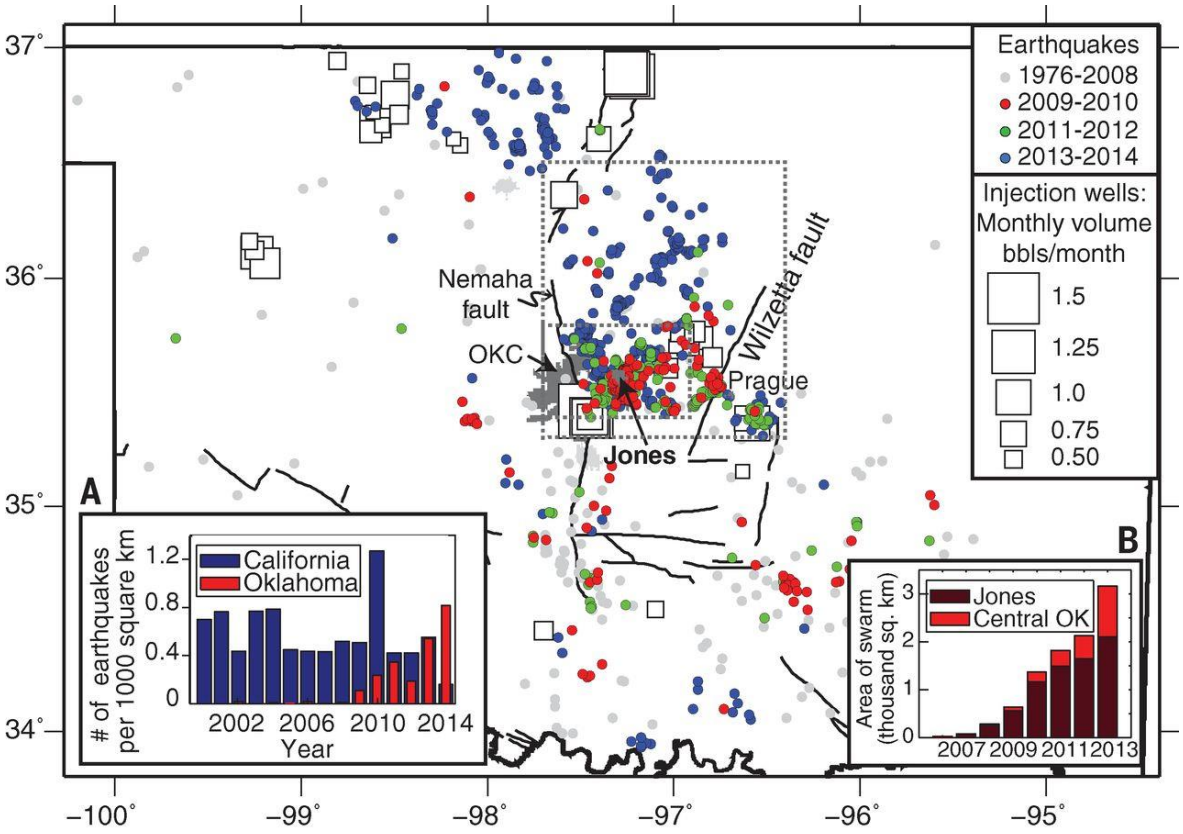



**Figure 2.15: "Earthquakes in Oklahoma between 1976 and 2014.**

**Earthquakes are M > 1 from the NEIC catalog (10). Black lines are faults (26–28). Small and large dashed gray boxes outline the areas used for analysis of the Jones swarm and of central Oklahoma, respectively, in inset B. OKC: Oklahoma City. Inset A: Comparison of M3+ earthquake rate in Oklahoma**

**and California, normalized by area. California is ~2.3 times larger than Oklahoma. 2014 earthquakes are through the first 4 months. Inset B: Expanding area of the Jones and the broader central Oklahoma swarms. Regions were divided into 5 km by 5 km grid cells, and any cell with an earthquake was considered part of the swarm. Swarm area per year is inclusive of all prior years." (Keranen et al., 2014).**

Seismicity and hydrogeological models link the sharp increase in seismicity in Central Oklahoma to wastewater injection. Pore pressure simulations demonstrate the feasibility of wastewater injection operations and the subsequent pore pressure diffusion leading to regional increases in seismicity (Fig. 2.15) (K. M. Keranen et al., 2014). Simulations were able to match the positions and sequence of seismic events, for the highest energy earthquake swarm, to a pore pressure diffusion wave in the wastewater disposal formation and upper basement, from 2-5 km in depth. Significantly while thousands of disposal wells operate aseismically, it was found that just four of the highest rate wells would be capable of inducing 20% of the reported 2008 - 2013 central US seismicity.

Further, analysis of subsurface fluid conditions and earthquakes from 2005-2014 of the Bend-Arch, Fort Wirth Basin in North Texas, shows an exponential increase in seismicity from the onset of wastewater disposal in 2008 (Hornbach et al., 2016). With the largest event being a $M_W$ 4.0, in 160 years of habitation and 40 years of monitoring, no felt earthquakes had been recorded in the area before 2008. A robust connection exists between 1.7 billion barrels of wastewater injected, a cumulative increase average fluid pressure at depth (1.7 - 4.5 MPa) across the Bend-Arch, Fort Wirth Basin, and the increased occurrence of seismicity in the area and up to 10 km away.

In similar circumstances, an area with no known prior earthquakes, Youngstown Ohio, experienced 109 earthquakes ($M_W$ 0.4 - 3.9) over 14 months of wastewater injection activities from January 2011 - February 2012 (Kim, 2013). Initial seismicity occurred in the vicinity of the wellbore, occurring at greater distances over time. This migration of seismicity indicates pore pressure diffusion resulting from human subsurface fluid injection as a cause. Strong temporal correlations between wastewater injection and seismicity were further supported by the observation that periods of low wastewater injection volume were accompanied by a period of seismic quiescence.

In the longest running monitoring study, a well-monitored continuous (1991 - present), deep (4.3 - 4.8 km) wastewater injection operation was examined in Paradox Valley, Colorado. Here, 15 events exceeding $M_W$ 2.5 occurred, with the largest event being a $M_W$ 4.3 as of 2003. Isolated seismic events and swarms both exhibit a strong spatial correlation with the zone of fluid injection (Ake et al., 2005).

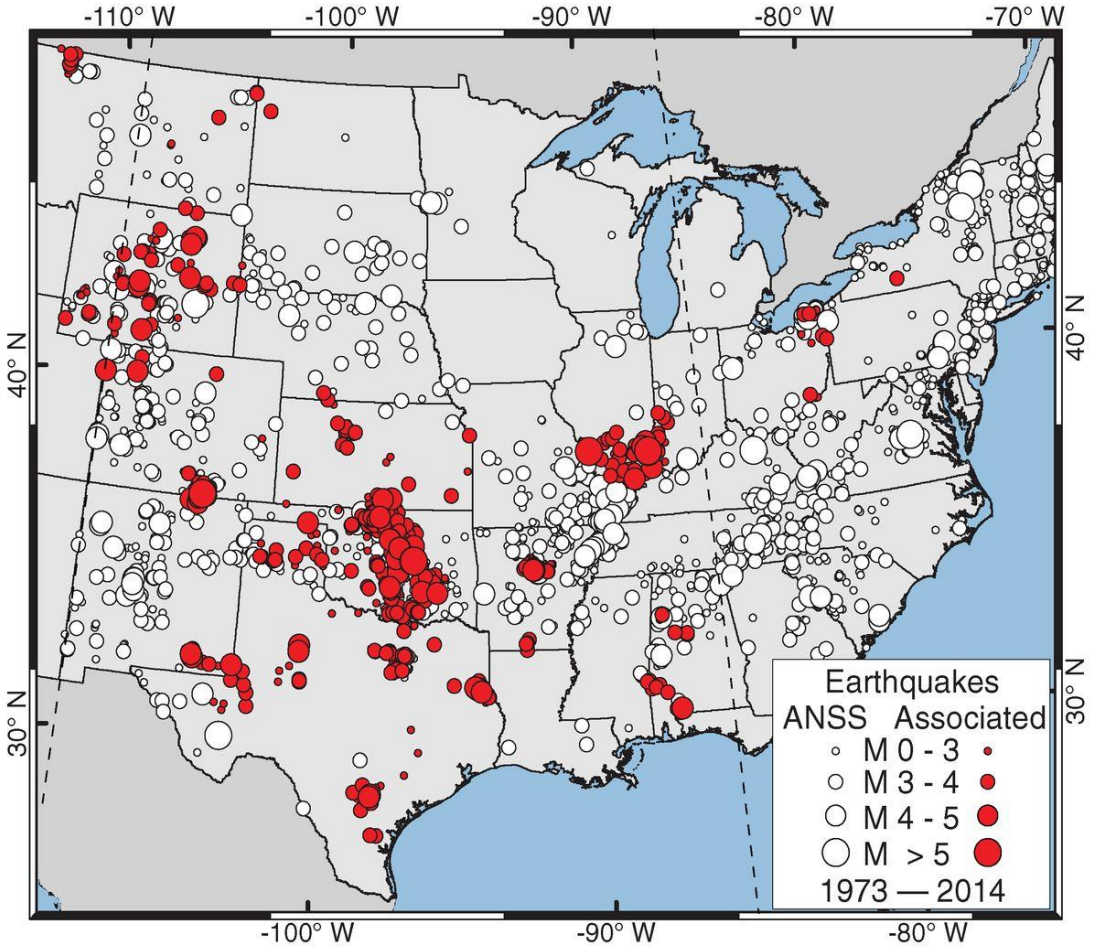



**Figure 2.16: "Associated earthquakes in the [central and eastern United States] from 1973 to 2014. Map showing the locations of $M \geq 0.0$ earthquakes in the [Advanced National Seismic System's comprehensive earthquake catalogue] from 1 January 1973 through 31 December 2014. White dots denote earthquakes that are not spatiotemporally associated with injection wells. Red dots denote earthquakes that are spatiotemporally associated with injection wells. Following Ellsworth" … "the U.S. mid-continent is defined by the dashed lines inside of the greater central and eastern United States." (Weingarten et al., 2015).**

### 2.5.3 Seismicity induced by hydraulic fracturing

Seismicity induced by hydraulic fracturing has now been observed in a number of regional studies, independently of wastewater injection operations. It has also contributed to the previously mentioned exponential increase in seismicity in the continental USA (Fig. 1.1) (Ellsworth, 2013). Hydraulic fracturing induced seismicity has been observed in a number of countries: USA (Friberg et al., 2014; Frohlich, 2012; Frohlich and Brunt, 2013; Holland, 2013; Keranen et al., 2013; Kim, 2013; McGarr, 2014; McGarr et al., 2015; Rutledge et al., 2004; Rutledge and Phillips, 2003; Skoumal et al., 2015; Weingarten et al., 2015), Canada (Atkinson et al., 2015, 2016; Bao and Eaton, 2016; Farahbod et al., 2015a, 2015b; Schultz et al., 2015), China (Lei et al., 2017) and UK (Clarke et al., 2014; De Pater & Baisch, 2011).

There has been an exponential increase in mid-continental seismicity within North America. Notably the USA, correlated strongly with the rise of high rate (≥ 300, 000 barrels per day) wastewater injection wells since 2009 (Fig. 2.16) (Weingarten et al., 2015). A breakdown of data from the Advanced National Seismic System's comprehensive earthquake catalogue, from the 1st of January 1973 to the 31st of December 2014, indicates that this exponential increase is only present at sites associated with injection wells. No such increase in seismicity occurs for the earthquakes not associated with injection wells. Further dissecting the injection well data shows high rate wells were significantly more likely to be associated with earthquakes than lower rate injection wells.

In fact, human subsurface injection has altered the seismic landscape of the United States sufficiently that as of 2016, Oklahoma, an area of previously low seismic activity, was now experiencing a greater volume of $M_W$ 3.0 or greater

earthquakes than naturally seismically active areas such as California (McGarr et al., 2015). Concern over this abrupt deviation to the spatial distribution of seismicity on the continental scale was enough for the previous authors to propose a series of interventions to reduce seismic hazard and manage social licence.

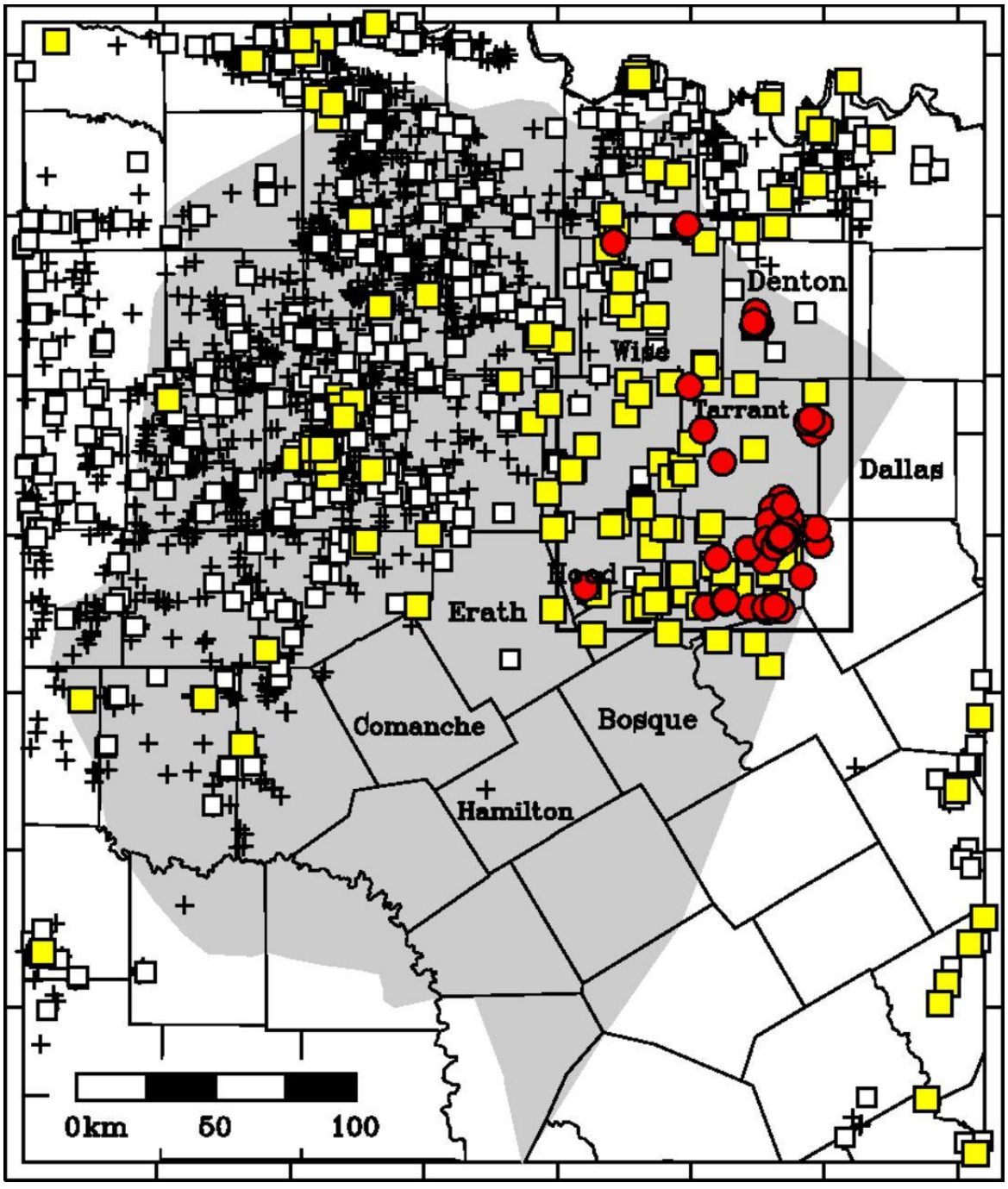



**Figure 2.17: "Map of Barnett Shale area" … "showing earthquakes located in this study (red circles) and injection wells in use since 2006 (squares and +**

**symbols). Yellow squares are wells reporting maximum monthly injection rates exceeding 150,000 BWPM (24,000 m3/mo); white squares, exceeding 15,000 BWPM (2,400 m3/mo); + symbols, exceeding 1,500 BWPM (240 m3/mo)." (Frohlich, 2012).**

Observations of induced seismicity have increased in line with an increased number of hydraulic fracturing operations. In the following part of the chapter, we review some of the most relevant and recent case studies of injection induced seismicity.

For instance, extensive monitoring of the Barnett Shale, Texas, on a 70 $km^2$ grid between November 2009 and September 2011, demonstrate that all of the 24 most reliably located earthquake hypocentres, of a set of 67 total detected earthquake hypocentres, were within 3.2 km of at least one hydraulic fracturing injection well. These earthquakes were all located in the vicinity of 9 of the 27 high rate wells (> 150,000 barrels per month) (Fig. 2.17) (Frohlich, 2012). The distribution of favourably stressed faults in the area likely explains why injection wells of a similarly high rate did not all induce seismicity. In fact, high-pressure fluids needed to contact a critically stressed fault to reduce effective normal stress on the fault plane and induce seismicity. Similar monitoring of the Eagle Ford Shale, Texas again, on a 70 $km^2$ grid between November 2009 and September 2011, detected 62 probable earthquakes, clustered into 14 foci. Ten of these foci located near wells involved in the injection of subsurface fluids or the extraction of recently injected subsurface fluids. Shortly after the cessation of monitoring, a Mw 4.8 event occurred at nearby Fashing on the 20th October 2011, without any previous increase in the injection of subsurface fluids and felt earthquakes had happened in the area before significant injection operations in 1973 and 1983 (Frohlich and Brunt, 2013). The prior history of seismicity not associated with injection and the weaker correlation

between injection-related activities indicate that a more complex relationship exists between human subsurface fluid injection and seismicity in this region.

Also, five hydraulic fracturing treatments of the Carthage Cotton Valley Gas Field, in Texas, initiated microseismicity. Initially, dense microearthquake clusters in the targeted layers would diffuse with time from the onset of a treatment, indicating fluid movement into the surrounding lithology (Rutledge and Phillips, 2003; Rutledge, Phillips, and Mayerhofer, 2004). The focal mechanisms and event locations suggest that the microseismicity was primarily comprised of motion on the reservoir's natural fractures.

Further, ten widely observed positive magnitude earthquakes of the October 2013 seismic sequence in Harrison County, Ohio, were spatially and temporally linked with the hydraulic fracturing operations at the Ryser wells (Friberg et al., 2014). The detection of other seismic events, which were cross correlated with the ten positive magnitude earthquakes, tapered off with time following hydraulic fracturing operations. These observations, together with the similarity of seismic waves detected from all events, makes it probable that hydraulic fracturing operations were responsible for the entire October 2013 seismic sequence.

Hydraulic fracturing operations triggered the 2011$M_W$ 5.7 earthquake sequence near Prague, Oklahoma (Keranen et al., 2013). Its $M_W$ 5.0 foreshock was connected directly to fluid injection (Keranen et al., 2013). By detecting and locating 110 earthquakes in the sequence, Sumy et al. (2014) demonstrated that the subsequent cascade of seismic events, triggered by coseismic Coulomb stress transfer, resulted from the foreshock. This indicated that contrary to what argued by

McGarr (2014), the volume of fluid injected might not limit mainshock magnitude or cumulative seismic moment release.

Also, analysis of 77 earthquakes spatially and temporally correlated with hydraulic fracturing activities in Poland Township, Ohio, suggested a causal link between the two. However, nearly 100 stimulation stages in nearby wells did not coincide with felt seismicity, suggesting it did not occur in all cases (Skoumal et al., 2015). A series of events were recorded up to $M_W$ 3 (one of the largest detected at the time, 2014) and the observed seismicity shared a lot of characteristics with nearby induced seismicity in Youngstown, 18 km to the northwest (Kim, 2013).

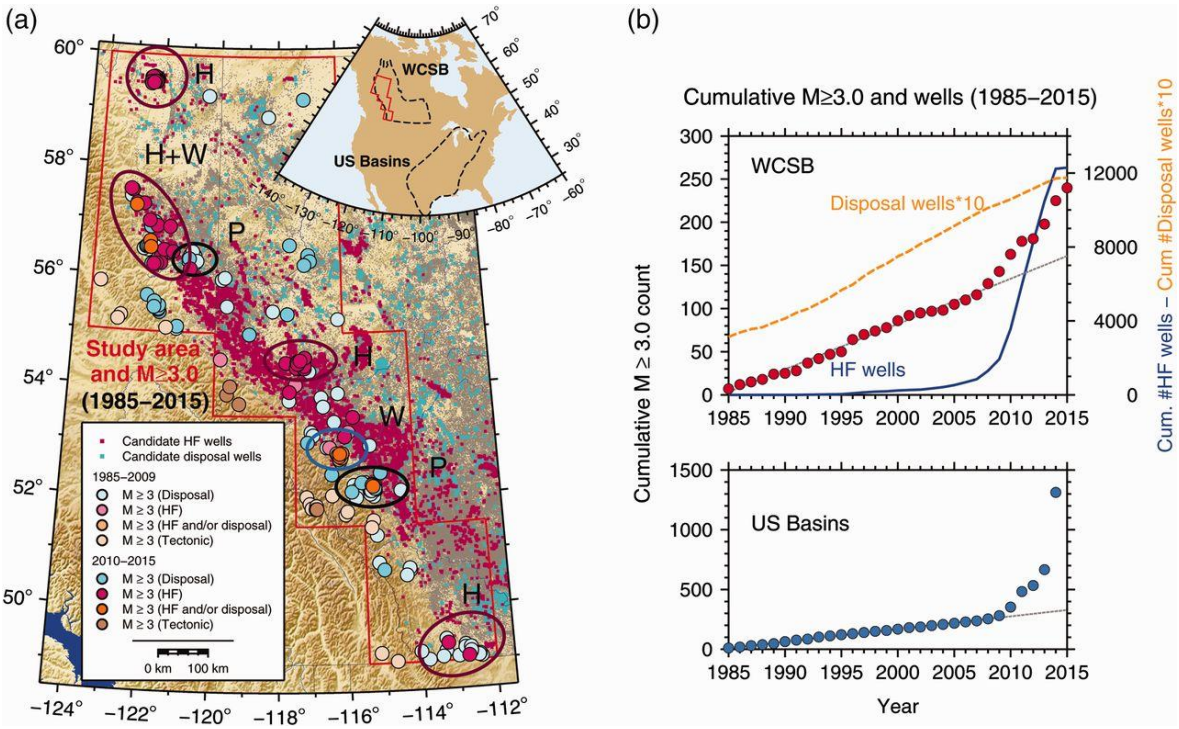



**Figure 2.18: "Seismicity and wells in the Western Canada Sedimentary basin (WCSB). (a) Red lines delineate the study area, which parallels the foothills region of the WCSB. Ovals identify areas where induced seismicity has been previously attributed to hydraulic fracturing (H), wastewater disposal (W), and production (P). Red/pink circles show M ≥ 3 earthquakes correlated with**

**hydraulic fracture (HF) wells. Turquoise circles show M ≥3 earthquakes correlated with disposal wells. Orange circles are correlated with both. Small squares in the background show locations of examined HF wells (dark pink) and disposal wells (turquoise). Gray squares in the far background are all wells.**

**(b) Cumulative rate of seismicity within the WCSB, commencing in 1985; numbers of disposal wells and HF wells for the WCSB as compiled in this study are indicated (top). A roughly synchronous increase in rate is evident in the basins of the central and eastern United States.**

**(bottom; data plotted from Ellsworth, 2013) (Well information is not available in the Ellsworth study, but most activity is considered to**

**be related to wastewater disposal.) The gray lines show the expected counts for a constant seismicity rate." (Atkinson et al., 2016)**

At least 86 earthquakes accompanied hydraulic fracturing operations in South Central, Oklahoma, from the 16th-23rd January 2011, with 16 of these events exceeding $M_W$ 2.0. A cross-correlation analysis showed no similar seismic waveforms outside of the window of hydraulic fracturing operations (Holland, 2013). Poor weather conditions at the well-site led to hydraulic fracturing stages being separated by approximately two days, increasing the precision of the temporal correlations between seismicity and fluid injection operations.

A study of hydraulic fracturing and seismicity in the Western Canada Sedimentary Basin shows that the seismic events correlated strongly with hydraulic fracturing operations (Fig. 2.18) (G. Atkinson et al., 2015; G. M. Atkinson et al., 2016). However, the rate of seismicity did not appear to obey the expected relationship between the volume of fluid injected and the maximum observed

seismic magnitude. A significant number of events exceeded predictions of seismic magnitude indicating the seismic hazard of hydraulic fracturing may be larger than routine analysis would suggest.

Further, a region of previous seismic quiescence within central Alberta, Canada, experienced a sequence of earthquakes beginning 1st December 2013, and comprising 160 events as of the end of 2014. Seismic monitoring showed that events clustered at each of the sites of horizontal drilling. The data could be further resolved into five temporal sub-sequences, with the first-order relations to hydraulic fracturing operations (Schultz et al., 2015). Analysis of the seismic waveforms was sufficiently precise to indicate that seismicity would stop when hydraulic fracturing operations would stop, and resume when they would restart months later, strongly implying direct causation.

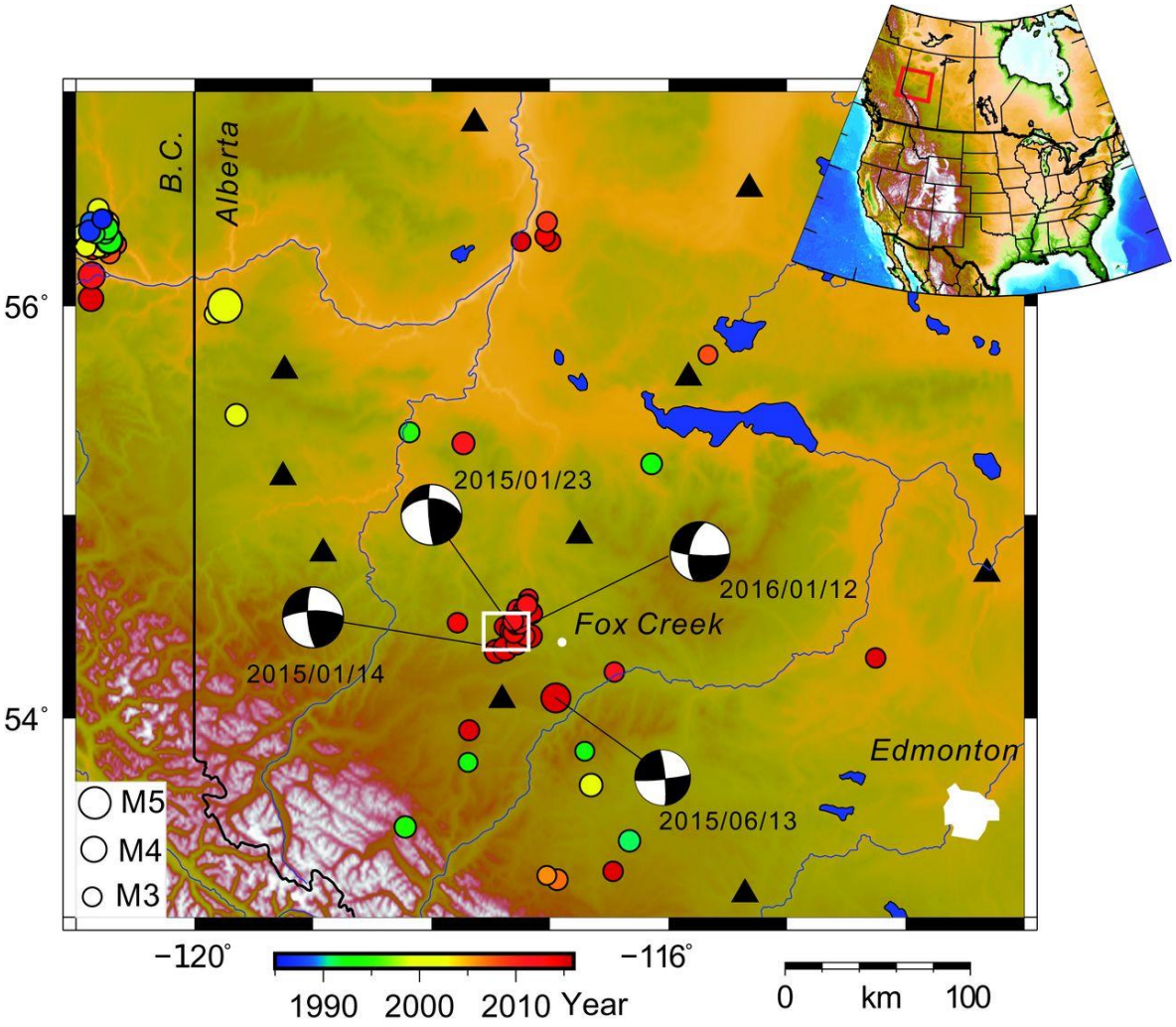



**Figure 2.19: "Seismicity of northwestern Alberta, Canada, for the period 1985−2016. Symbol size indicates magnitude, and color denotes date of occurrence. B.C., British Columbia. Seismicity west of Fox Creek commenced in December 2013 and correlates in space and time with local hydraulic-fracturing operations (9). Focal mechanisms of the largest earthquakes, from (32–34), are labeled by year/month/ date of occurrence." (Bao and Eaton, 2016).**

Analysis from Western Canada, specifically north-western Alberta, of hydraulic fracturing and seismicity over a four-month period, again finds a strong spatial and temporal correlation between hydraulic fracturing and seismicity. One large ($M_W$ 3.9) event occurred several weeks after injection, along a fault that

extends from the site of hydraulic fracturing operations into the crystalline basement (Fig. 2.19) (Bao & Eaton, 2016). Predictions of the stress change during seismicity suggests that fault activation could be possible more than 1 km away from the site of injection, and that direct pressurisation of faults locally could lead to episodic seismicity persisting for months.

Also, hydraulic fracturing operations in the Horn River Basin, northeast British Columbia, coincided with a sharp increase in seismicity (131 events per year) above background levels (24 events per year), as well as with an increase in the maximum magnitude, from $M_L$ 2.9 to 3.6 (Farahbod, Kao, Cassidy, et al., 2015). The analysis of the natural background and hydraulic fracturing seismogram data supported a physical link between hydraulic fracturing operations in the area and induced seismicity (Farahbod, Kao, Cassidy, et al., 2015).) The dominant factor controlling induced seismicity in the area appeared to be the volume of fluid injected, more so than the injection pressure. There was no change from background seismicity when the volume of injected fluid was less than 20,000 $m^3$ per month, and the largest seismic releases occurred with monthly injected volumes exceeding 150,000 $m^3$ (Farahbod, Kao, Walker, et al., 2015). The time lag from initiation of hydraulic fracturing subsurface injection and seismicity could be days or months depending on the local geological conditions, particularly the distribution and geometry of faults in the area.

Observations of hydraulic fracturing induced seismicity have also been made in China. The Sichuan Basin has experienced a series of earthquakes up to $M_W$ 4.7, resulting from fault reactivation driven by fluids injected into the subsurface for hydraulic fracturing (Lei et al., 2017). The combination of precisely relocated aftershock hypocenters, focal mechanism solutions of 13 significant events ($M_W$ 3.5)

and Coulomb failure stress analyses all indicate that injection over the course of several months at a single well pad, at depths of 2.3-3 km, induced each of the earthquakes.

Hydraulic fracturing operations in the Carboniferous Bowland Shale in the UK, at the well Preese Hall 1, resulted in a series of seismic events, the largest of which was $M_L$ 2.3, 1.8 km from the well at a depth of 3.6 km (Clarke et al., 2014; De Pater & Baisch, 2011). Furthermore, this sizeable seismic event immediately followed the injection of 2245 $m^3$ of fluid, and 117 tons of proppant at the well. Some small shear movements were detected slightly before the highest energy event. The $M_L$ 2.3 event likely resulted from fluid leaking from induced fractures to natural ones, before migrating onto the fault plane of a pre-existing, critically stressed fault. The fluids lowered the effective normal stress on the fault, which was reactivated triggering the seismicity.

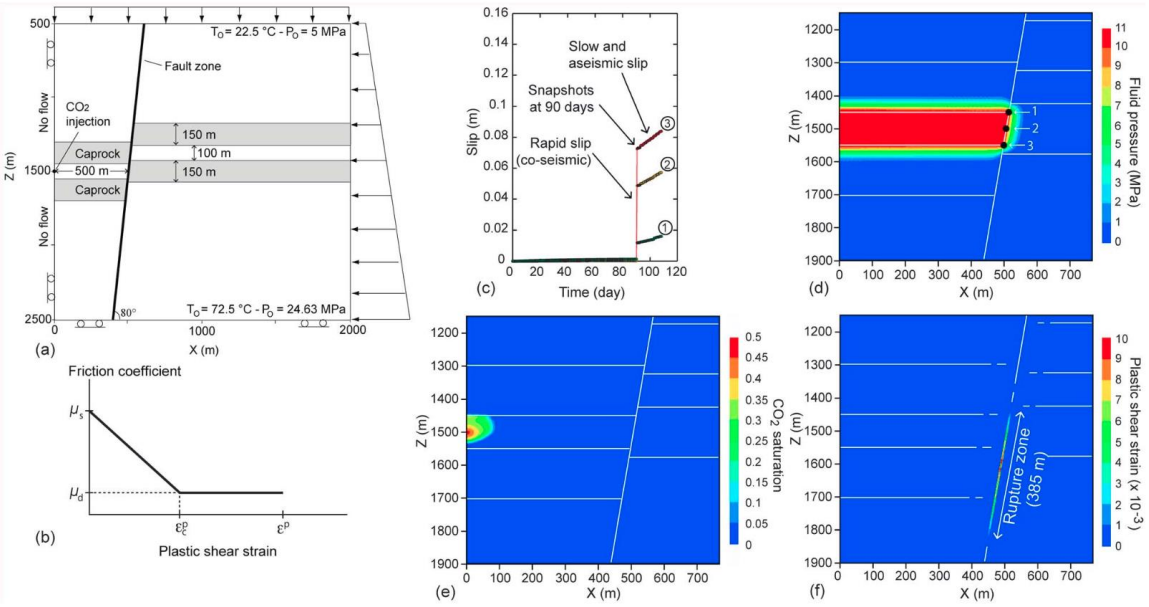



**Figure 2.20: "(a) Numerical model geometry and initial conditions. We assumed a normal fault with a 125 m offset through a 100 m thick reservoir bounded at**

**the top and the bottom by a 150 m thick caprock. (b) A plastic shear strain-weakening friction law that governs the propagation of rupture along the fault zone. (c) Fault slip versus time at three points located at the (1) top, (2) middle and (3) bottom of the reservoir, respectively" … "Snapshots of change (relative to the initial state) in (d) fluid pressure, (e) $CO_2$ saturation, and (f) plastic shear strain at the end of the sudden slip event (after 90 days of $CO_2$ injection)" (Cappa and Rutqvist, 2011a).**

### 2.5.4 Numerical Simulation of Induced Seismicity

Examination of the reservoir, caprock and fault systems with numerical simulation techniques have predicted human subsurface injection-induced seismicity and constrained conditions under which it might occur during carbon sequestration operations (Frederic Cappa & Rutqvist, 2012; Frédéric Cappa & Rutqvist, 2011a, 2011b; J. Rutqvist et al., 2002; Jonny Rutqvist, Cappa, et al., 2013). Credible stress ranges for microseismicity in caprock embedded faults during similar carbon sequestration simulations (J. Rutqvist et al., 2002). Seismic movement and a sudden stress drop were predicted within a few months from the beginning of $CO_2$ injection into a reservoir, for a schematic fault embedded in a caprock for realistic physical parameter ranges in initial horizontal-to-vertical stress ratio and fault permeability (Fig. 2.20) (Frédéric Cappa & Rutqvist, 2011b). These simulations were refined by including effective stress dependent permeability. The results demonstrated a relationship between the physical parameters stress ratio and fault permeability, the size of the rupture patch and earthquake magnitude. In a model like the previous case with simple fault zone architectures were implemented within simulations of $CO_2$ injection to examine their impact on predictions of seismicity, fluid flow and the mechanical response of faults (Frédéric Cappa & Rutqvist, 2011a). Here simple fault

zone architecture refers to a damage zone and fault core. This reservoir with caprock embedded fault simulation was extended to make predictions of ground acceleration and seismic wave propagation in the case of a critically stressed fault (Frederic Cappa & Rutqvist, 2012). Showing that within a few days of fluid injection sliding can begin on a small patch of a few centimetres, which can develop rapidly into a more massive earthquake if fluid flow and fault weakening act to reduce the effective coefficient of friction for the fault plane.

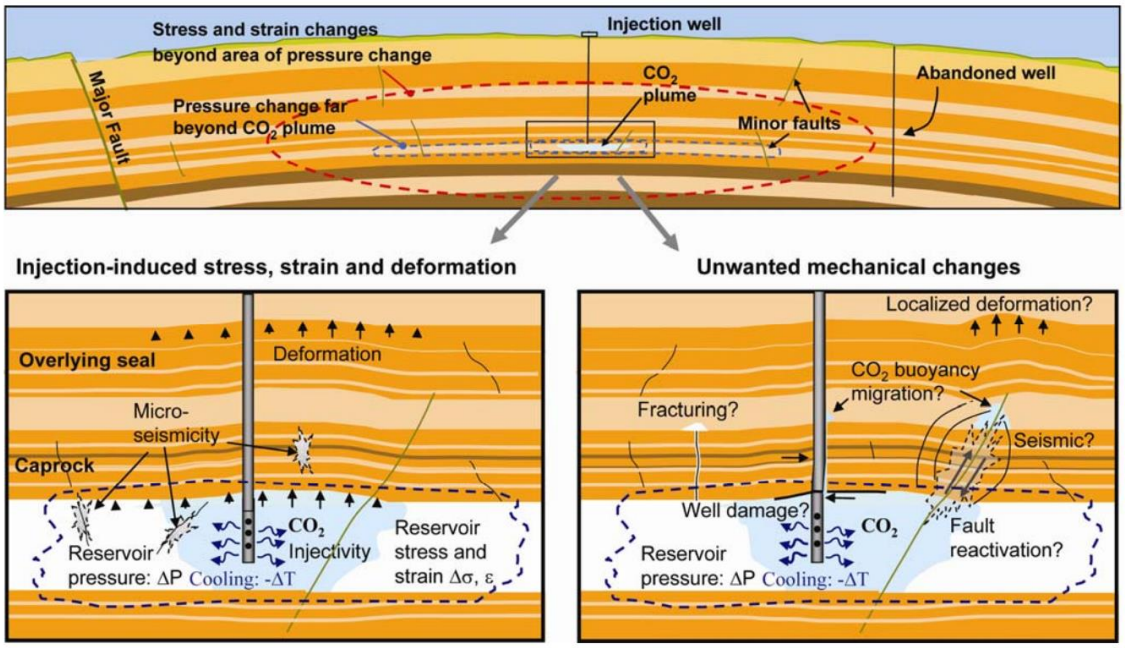



**Figure 2.21. "Geomechanical processes and key technical issues associated with GCS in deep sedimentary formations. Top the different regions of influence for a CO2 plume, reservoir pressure changes, and geomechanical changes in a multilayered system with minor and major faults. Bottom left injection-induced stress, strain, deformations and potential microseismic events as a result of changes in reservoir pressure and temperature, and**

**bottom right unwanted inelastic changes that might reduce sequestration efficiency and cause concerns in the local community." (Rutqvist, 2012).**

Simulations of natural fault systems subjected to human injection activities have been validated using the $CO_2$ injection and storage project at In Salah, Algeria (Fig. 2.21) (Jonny Rutqvist, 2012; Jonny Rutqvist et al., 2009). A thin, low permeability, Carboniferous sandstone formation was targeted for $CO_2$ storage at a depth of 1.8 - 1.9 km. Three long reach 1-1.5 km horizontal injection wells were utilised, with injection occurring at 18 MPa of pressure. A realisation of the TOUGH-FLAC simulator was able to predict microseismicity that was comparable to that detected at the site. Modelling results also predicted the distribution of pore pressure in the reservoir and the evolution of stress predicted was consistent with those at In Salah. TOUGH-FLAC simulations in general infer earthquake properties using empirical seismological relations instead of directly simulating the earthquake nucleation phase.

Specific operational constraints (J. Rutqvist et al., 2007) and hazards (Mazzoldi et al., 2012; Mortezaei & Vahedifard, 2015) were evaluated by simulating the pressure distribution of caprock embedded faults overlying reservoirs targeted for carbon sequestration. A fully coupled numerical analysis of schematic faults was performed using TOUGH-FLAC and simple models of fault zone architecture. While these models did not simulate earthquake nucleation directly, they were able evaluate the maximum sustainable $CO_2$ injection pressure that would avoid seismicity for carbon sequestration operations (J. Rutqvist et al., 2007). As well as operational constraints, estimates of the magnitude of functional seismic hazard have been calculated with numerical simulation techniques using the TOUGH-FLAC suite, for a schematic domal structure targeted for the deep storage of $CO_2$, focusing

specifically on reactivating sub-seismic resolution faults (Mazzoldi et al., 2012). The seismic magnitudes for movements on these sub-seismic resolution faults was inferred from seismological relations to be ($2 \leq M_W \leq 3.9$). Another study evaluated seismic hazard in a similar way on sub-seismic resolution faults using TOUGH-FLAC simulations, posing the problem as a parameter study of permeability and reservoir thickness (Mortezaei & Vahedifard, 2015).

TOUGH-FLAC simulations where extended and refined for the simulation of shale fault activation during hydraulic fracturing operations. Specific examinations of steeply dipping faults, at depths of 1000 to 2500 m, indicated that hydraulic fracturing could induce shear failure, and hence microseismicity (Jonny Rutqvist et al., 2015). The seismic moment magnitudes predicted by the TOUGH-FLAC shale fault simulation during typical hydraulic fracturing operations ranged from $M_W$ -2.0 to 0.5, excepting one $M_W$ 2.3 simulation of a very brittle fault with low residual shear strength. They conclude that felt seismicity is unlikely to result from hydraulic fracturing operations in the vicinity of steeply dipping faults.

TOUGH-FLAC simulations have been extended further to incorporate faults embedded in lithologically complex, layered systems as part of a caprock-reservoir system targeted for carbon sequestration (Rinaldi et al., 2014). These simulations demonstrate that the inclusion of heterogeneities strengthens the fault and decreases the magnitude of earthquakes by preventing the propagation of rupture to shallow depths. The complex hydraulic properties of the multilayer also impede the flow of fluids along the fault. The simulations were even able to predict that while thin caprocks and/or aquifers might produce smaller magnitude events, they also increased the volume of leaked fluid.

## 2.6 Conclusion

Many case studies in the literature, clearly support the evidence that subsurface fluid flow can trigger both natural and human induced seismicity, up to $M_W$ 9.0 events for natural seismicity (Terakawa et al., 2013). There is also evidence that human activities, such as fluid injection, have led to an exponential increase in seismicity in the continental USA (Ellsworth, 2013; Weingarten et al., 2015) and also in other areas of the world.

Fault mechanics theory predicts that increasing pore pressure reduces fault frictional strength and can favour the reactivation of faults at lower stress levels or even when faults are in their stability stress field. Although the rock mechanics principles and laws that govern fault reactivation are simple, fault frictional strength can depend in a highly non-linear way on supra-hydrostatic pore pressure gradients, potentially driving seismicity (Cox, 2010; Sibson, 1990, 1992).

Numerical simulation techniques have been used to analyse fault reactivation more precisely, modelling subsurface fluid flow and pore pressure distribution within faults, eventually causing fault reactivation (Cappa and Rutqvist, 2011a, 2012; Rutqvist et al., 2015, 2013a, 2002). Further, a number of metre- to kilometre-scale models have refined these results to include simplistic models of fault zone architecture, and pore pressure dependent fault zone transport properties (Cappa et al., 2009; Cappa and Rutqvist, 2011; Hsiung et al., 2005; Mazzoldi et al., 2012; Rinaldi et al., 2014; Rutqvist et al., 2013, 2009). However, there are several previously unconsidered model refinements that could more realistically predict the complexity of fluid flow and reproduce the behaviour of natural faults. These are treated in the next chapters of this thesis and include the implementation into models of simple to complex and more realistic models of fault zone architecture. In

particular, the models output of fluid flow and fault reactivation conditions account for the evolution of permeability in the different fault zone domains due to its dependence on evolving pore pressure and mode of failure, e.g. brittle vs. ductile (Caine et al., 1996; Caine and Forster, 1999; Collettini et al., 2009; De Paola et al., 2008)

Finally, there are several previously unconsidered model refinements that could more realistically reproduce the behaviour of natural faults and accurately characterise and forecast their seismicity. Here, I attempt to reproduce in simulations the earthquake nucleation phase, as opposed to the approach adopted in previous studies, where seismological relations were used to infer earthquake nucleation (Cappa and Rutqvist, 2011a, 2012; Rutqvist et al., 2015, 2013a, 2002). This is achieved by considering the effects of reducing effective normal stress, which would result in an increasing nucleation length – the size of the rupture patch needed for earthquake nucleation (Campillo et al., 2001; Scholz, 1988; Uenishi and Rice, 2003).

# CHAPTER 3

*Simulating fluid overpressure in low-porosity faults with brittle and ductile mode of failure and earthquake nucleation*

# 3. Simulating fluid overpressure in low-porosity faults with brittle and ductile mode of failure and earthquake nucleation

## 3.1 Introduction

Failure and deformation processes can occur continuously throughout the interseismic period and aren't necessarily isolated to primary slip zones involved in active fault-slip and earthquake nucleation. These brittle and ductile deformation processes act as a primary control on fluid flow and pore pressure evolution and hence earthquake nucleation (Rowland and Sibson, 2004). We model the effect of the processes on the aforementioned physical properties using the triaxial deformation with fluid flow measurements of De Paola et al., 2009.

Here we refer to distributed fracturing as ductile failure, there is not a full transition to viscous behaviour as we might see at high temperatures, it is a discrete failure event (a collapse), which alters the porosity of the rock and transport properties. It does not engage in fully fluid-like behaviour for an extended period in a way which would require simulation. Treating these brittle and ductile mode of failure events as a field of discrete failure states, simplifies models of fault-fluid evolution and earthquake nucleation both conceptually and computationally. Chopping sub-milli-second and -metre continuous failure and deformation events into discontinuous failure states and transitions reduces the number of physical processes considered while recovering the natural, irreversible, hysteretic behaviours resulting from brittle and ductile mode of failure, not typically recovered in continuous models.

Further, numerical simulations and multiphysics models of the natural world, as well as their inputs, are necessarily approximations. Very few non-fundamental macroscopic physical relationships capture the full complexity of natural behaviours. Also, we cannot constrain any physical quantity with perfect precision, particularly in the subsurface. Any numerical simulation result must either be robust to changes in these uncertain variables otherwise we will need to constrain the behaviour we observe in these simulations over a range of values for each uncertain parameter. Understanding how simulation results vary within the range of possible uncertainty, we would expect for a fault in the natural subsurface allows us to distinguish which of our results would apply widely to similar faults and conversely what variation we might expect to between different examples of similar faults..

Here we model the nucleation of earthquakes, which depend on the frictional behaviour of faults and the normal stresses acting on fault planes. These in turn depend on the fluid pressure acting on the fault plane. The fluid pressure acting on the fault plane is controlled by fluid flow throughout the fault zone, this fluid flow can be facilitated by the deformation of the rocks comprising the fault zone. This study will model the evolution of stress, fluid flow, pore pressure and deformation throughout the fault zone and predict subsequent earthquake nucleation. We establish a method for efficiently simulating fault zone pore pressure diffusion in the interseismic period with complex, realistic models of fault zone architecture and brittle and ductile modes of failure. The nucleation phase is simulated, and stable sliding and earthquake nucleation are resolved and distinguished to the order of seconds for several hundred years simulations. As all the simulations presented in this thesis are consistently several orders of magnitude within the region for Darcy flow (Thauvin and Mohanty, 1998).

### 3.2 Governing Equations

All simulations presented in this thesis consider fluid flow through porous media. By considering any three-dimensional element of a porous medium for mass to be conserved mass flux into this element minus mass flux out equals the increase in amount stored by the element, we represent this mathematically as follows (Table 3.1):

$$\nabla.(\rho q) = -\frac{d(\rho\varphi)}{dt} \qquad (3.1)$$

As all the simulations presented in this thesis are consistently several orders of magnitude within the region for Darcy flow (Thauvin and Mohanty, 1998), we take the left hand side of this equation and substitute Darcy's law, whilst also assuming incompressibility (Table 3.1):

$$\nabla.(\rho q) = -\frac{\rho}{\eta}\nabla.(k\nabla P) \qquad (3.2)$$

Taking the right-hand side of this equation and differentiating using the product rule and using the definitions of pore and fluid compressibility ($\beta_\varphi$ and $\beta_f$ respectively) gives (Table 3.1):

$$\frac{d(\rho\varphi)}{dt} = \rho\frac{d\varphi}{dt} + \varphi\frac{d\rho}{dt} = \rho\frac{d\varphi}{dP}\frac{dP}{dt} + \varphi\frac{d\rho}{dP}\frac{dP}{dt} = \rho\varphi\left(\frac{1}{\varphi}\frac{d\varphi}{dP} + \frac{1}{\rho}\frac{d\rho}{dP}\right)\frac{dP}{dt}$$

$$= \rho\varphi\left(\beta_\varphi + \beta_f\right)\frac{dP}{dt} \qquad (3.3)$$

By combining equations 3.1 and 3.2 we arrive at a relationship for the diffusion of pore pressure in a classically porous medium, for a laminar flow (Eq. 3.4; Table 3.1; Zimmerman, 2018),

$$\frac{dP}{dt} = \frac{\nabla\cdot(k\nabla P)}{\beta\eta\varphi} \qquad (3.4)$$

with $\beta = \beta_\varphi + \beta_f$, viscosity η and transport and rock property relationships derived from (De Paola et al., 2009). As the compressibility of the porous medium is several orders of magnitude greater than that of the pore fluid the compressiblity of the porous medium will have an outsized impact on flow. We use equation 3.4 throughout this thesis to approximate fluid flow within low permeability porous medium. Inherent in our approach is an assumption that the fluid present is a dry single phase CO2 fluid. However, most naturally occurring subsurface CO2 would contain a proportion of water. The presence of moisture would lead to dissolution/precipitation of the rock matrix, would swell clays if present and would also alter frictional rock behaviours. We accept this dry fluid assumption as an approximation, leaving the complexity of wet CO2 for future work.

$$\beta_\phi = \frac{\alpha(1-2\upsilon)}{E} \qquad (3.5)$$

The compressibility of the porous rock matrix can be expressed in terms of Poisson's ratio $\upsilon$, Biot coefficient α and Young's modulus E (Eq. 3.5, Detournay and Cheng, 1993). The compressibility of a porous medium is the change in volume of that medium in response to a change in effective pressure. When a rock deforms according to two modes of deformation, one along the axis which stress is applied, characterised by the Young's modulus E (the strain in response to applied stress), and secondly a transverse expansion in response to compression along the initial stress axis (characterised by the Poisson ratio $\upsilon$, relating axial to transverse strain. In porous media containing fluids a third factor must be considered, the Biot coefficient $\alpha$ which characterises the amount of fluid which would be expressed in response to a

change in volume. When these factors are combined as in Eq. 3.5 we arrive at the true total volume change in response to a change in pressure.

No macroscopic natural system is going to undergo genuinely discrete transitions with respect to time. on some low level of time or spatial resolution there will be observable continuous processes governing the transition from one state to another. to simulate all of these processes would be computationally intensive. Therefore we approximate failure state in our numerical simulations with a discrete variable as a simplification, this discontinuous approach to failure state allows us to analyse the problem on a space and time resolution of millimetres or milliseconds and above. Specifically, permeability, pressure sensitivity and porosity all vary discontinously with the failure state of the rock (prefailure, localised brittle fracturing or distributed ductile fracturing) and the component of fault zone architecture.

In natural rocks changes in porosity drive changes in permeability, these porosity changes are in turn controlled by the effective stress acting on the rock. as is indicated by linear elastic models (Berryman, 1992). In the simulations in this thesis we do not directly consider porosity except for discontinuous failure transitions. In continuous permeability changes, porosity changes are treated implicitly, and hence continuous permeability changes are modelled as driven only by effective stress. Permeability is represented by the following function (De Paola et al., 2009; Faulkner, 2004; Faulkner and Rutter, 2003, 2000; Zhang et al., 1999):

$$k = k_0 \exp(-\gamma \sigma') \quad (3.6)$$

$\gamma$ represents the pressure sensitivity, and $\sigma_{\text{eff}}$ is effective stress:

$$\sigma' = \sigma_3 - P \quad (3.7)$$

$\sigma_3$ is the principal minimum stress.

When considering fluid flow in the fault fluid system, we employ several more standard relationships between physical variables. For instance, the normal stress acting on a plane $\sigma_N$ at angle $\theta$ to the orientation of principal maximum stress $\sigma_1$ is given by (Cox, 2010, Fig. 2.1; Table 3.1):

$$\sigma_N = \frac{1}{2}(\sigma_1 + \sigma_3) + \frac{1}{2}(\sigma_1 - \sigma_3)\cos(2\theta) \quad (3.8)$$

The shear stress acting on a plane $\sigma_N$ at angle $\theta$ to the orientation of principal maximum stress $\sigma_1$ is given by (Cox, 2010; Table 3.1):

$$\tau = \frac{1}{2}(\sigma_1 - \sigma_3)\sin(2\theta) \quad (3.9)$$

In a fluid-saturated rock, the effective normal stresses are the normal stresses reduced by the amount of pore fluid pressure (Eq 2.1; Sibson, 1990; Table 3.1).

For compressional effective normal stress the shear failure of intact rocks is described by the Coulomb-Navier failure criterion (Sibson, 1996) where $\tau$ is the shear stress, $\mu$ is the coefficient of internal friction, $\sigma_N$ is the normal stress (Sibson, 1990; Table 3.1):

$$\tau = C + \mu_i(\sigma_N - P) \quad (3.11)$$

For the faults considered in this thesis, the shear failure envelope further decomposes into two regions one representing a brittle mode of failure and one representing ductile, with the transition occurring at a critical effective stress, derived from (De Paola et al., 2009). The coefficient of friction and cohesion also vary with mode of failure and fault zone architecture component, similarly to porosity, permeability and pressure sensitivity before.

Hybrid extensional-shear failure, can also develop in intact rocks; the Griffith criterion describes this phenomenon (Griffith, 1924).

Pre-existing faults, developed during previous deformation events, are usually considered planes of shear failure without any cohesion and, according to Amonton's law (Eq. 2.4; Table 3.1; R. H. Sibson, 1990), sliding will occur when the shear force on the fault exceeds frictional forces acting on the fault.

Stable sliding initiates at fault asperities and can spread out (in velocity weakening materials) with an accelerating sliding velocity until it reaches a critical size. This stability limit is a nucleation length, $L_c$ where G is the shear modulus, $\zeta$ is a constant of proportionality of order 1, $D_c$ is the critical slip distance, a and b are rate-and-state parameters (Dieterich, 1992; Rice and Ruina, 1983). The rate parameter a controls the variation of friction with velocity, the state parameter b controls the variation of friction with 'state', e.g. how much healing has occurred at the sliding interface since the last movement (Table 3.1):

$$L_C = \frac{\zeta G D_C}{\sigma'_N F} \qquad (3.13)$$

In our simulations, the frictional strength of the fault is taken to be homogeneous, and without asperities, with any variation in frictional strength modelled as being dependent only on effective normal stress. For the situations in which simulations are run in this thesis the 'asperity' due to fluid pressure would be the most significant as it is the most spatially expansive and the nucleation phase evolution would be dominated by it (Campillo et al., 2001). However, our analysis would not apply to situations in which fluid and fault asperities are both of similar scales or the fault asperity is greater. When shear stress exceeds the fault shear strength, for a given pore pressure, sliding begins along the fault plane in the primary slip zone. The

effective normal stress of the sliding fault patch is taken to be constant after sliding begins if it can no longer accumulate or dissipate stress energy locally as any change in energy will instead accelerate or decelerate sliding. We assume that the nucleation length ($L_N$) of a failure patch ($L_F$) is equal to that of its strongest point, which represents an upper limit (Campillo et al., 2001; Uenishi and Rice, 2003).

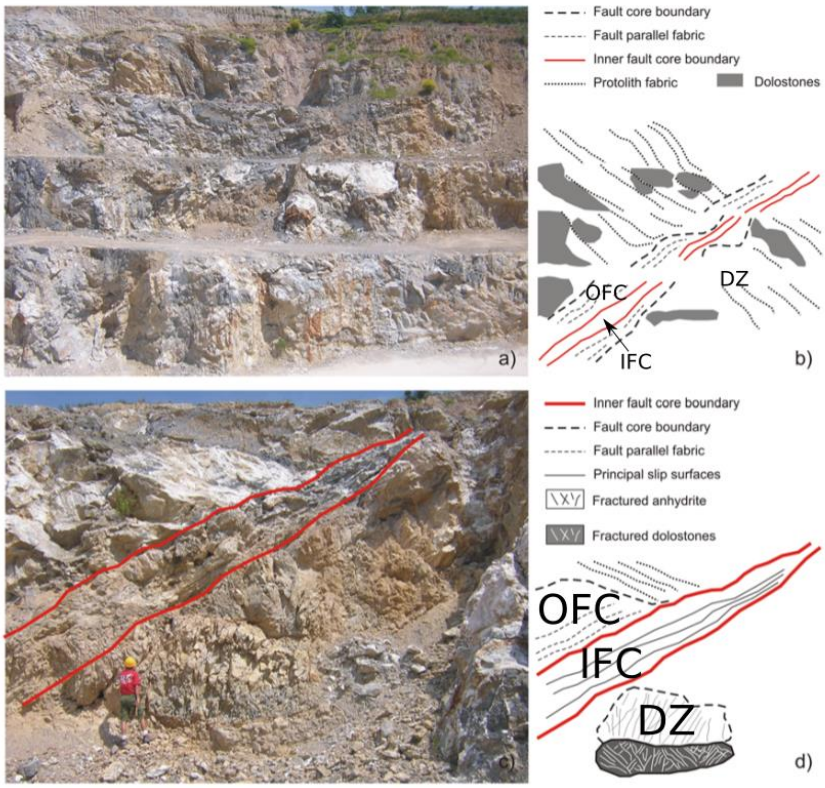



**Figure 3.1: "Macroscopic large-scale fault zone structure. (a) Panoramic view of a large-scale normal fault zone within the Triassic Evaporites. Note that the major fault zones crosscuts the former synorogenic mesoscale ''gneissic'' fabric (b) Line drawing of the fault zone shown in Figure [3.1]a. (c) Detail of the fault core of the large fault zone shown in Figure [3.1]a. The inner fault core boundary is highlighted. (d) Line drawing of the fault zone shown in Figure [3.1]c, displaying the internal fault core architecture." (De Paola et al., 2008) Adapted to show fractured dolostones and foliate anhydrite and outer fault core**

**(OFC) - inner fault core (IFC) boundary and damage zone (DZ) (De Paola et al., 2008).**

### 3.3 Multiphysics model

A single model of the coupled fault-fluid system is constructed using the physical relationships given in the previous section, evaluating at each simulated timestep: 1) the fields of intensive physical variables (e.g. pressure, stress), 2) transport properties (e.g. permeability), 3) quantities relating to failure and earthquake nucleation. There is a complex, nonlinear interdependence between each of these variables. For instance, permeability increases exponentially with effective stress as pore space in the rock increases and discontinuous transitions in porosity accompany brittle or ductile mode of failure (De Paola et al., 2009). We construct a multiphysics model of seismic low-permeability fault-zones from nonlinear pore pressure diffusion, realistic fault zone architecture, pre-, co-, and post-failure permeability sub-models as measured by triaxial deformation experiments with fluid flow and fault-rock failure models. We incorporate a schematic model of fault zone architecture comprising a damage zone (DZ) of interbedded fractured dolostones and foliated anhydrite and outer fault core (OFC) of foliated anhydrite and inner fault core (IFC) of fine-grained cohesive cataclasites and a principal slip zone (PSZ) of incohesive fault gouge (Fig. 3.1). The case studies considered in this thesis take as a base a model of fault zone architecture that is typical of extensional faults of the Northern Apennines, which is analogous to the central fault of the 1997-98 Colfiorito seismic sequence (De Paola et al., 2008).

Triaxial deformation measurements on real fault rocks with fluid flow were used to approximate pre-, co-, and post-failure permeability and the failure envelopes of the fault rocks on real fault rocks. These measurements were taken on fault rocks

corresponding to the OFC above. Separate measurements were taken with foliation both parallel and perpendicular to the direction of principal maximum stress, using samples from the Perugia 2 and Fossonbrone 2 boreholes in the Umbria-Marche Apennines in Italy (De Paola et al., 2009). Failure in the cohesionless PSZ was modelled using known friction laws (Byerlee, 1978).

A sub-model of earthquake nucleation processes was also considered, by treating the fault-fluid ensemble as a non-smooth system, where the rate of change of a physical parameter is undefined for at a least a point in time. Earthquake nucleation processes govern the dynamics of the fault-fluid system from stable non-acceleratory motion as shear strength is exceeded to the point at which the critical (nucleation) length is exceeded. Once this critical length is exceeded unstable, accelerating oscillation begins on the fault.

One of the critical components of our fluid-driven earthquake simulations is the inclusion of brittle and ductile mode of failure within models, as measured in the laboratory. Simulations of fault-fluid systems which neglect brittle and ductile mode of failure do not exhibit hysteresis, as the relationship between the transport properties and pore pressure is a function of only pressure, any unique distribution of pressure uniquely defines the state of the system. The state of the fault fluid system at a future instant in the interseismic period can be specified entirely using the state of the fault fluid system at the previous instant, and the history of the fault over the course of the interseismic period is irrelevant. The introduction of brittle and ductile mode of failure introduces discontinuous, irreversible failure behaviour and hysteresis, e.g. localised brittle failure that has occurred several years or tens of years earlier can impact fluid-driven earthquake nucleation and represents systematic time-dependence far beyond the previous instant in time.

### 3.4 Fault zone architecture and model setup

We validate our methodology with a case study of seismic extensional fault zones in evaporite rocks (Collettini et al., 2009; De Paola et al., 2009, 2008), from these faults we infer fault zone geometries, physical properties and mechanical behaviours.

In particular, the simulated fault zones in this thesis are exhumed normal faults in evaporite sequences (Collettini et al., 2009; De Paola et al., 2008) and analogues of the seismic sources in the hypocentre zone of the Northern Apennines seismic belt (e.g. Mirabella et al., 2008).

$CO_2$ fluxes in the Northern Apennines seismic belt (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.3 2009 L'Aquila extensional earthquakes) have been measured at greater than 0.45 t $day^{-1}$ $km^{-2}$ (Chiodini et al., 2004; Collettini et al., 2008). While, overpressured $CO_2$ was encountered in boreholes within the Triassic Evaporites, at ~80% of the lithostatic load, at depths of 4-4.8 km (Chiodini and Cioni, 1989; Collettini and Barchi, 2002; Miller et al., 2004). Our simulations assume that the modelled fault zone is saturated with supercritical $CO_2$ and treat it as being a single phase at a depth of interest 7 km, as the pressure conditions at the relevant depths indicate that both $CO_2$ and brine would be exists as a single miscible supercritical fluid (Miller et al., 2004).

The compressibility of supercritical $CO_2$ exceeds that of evaporite rocks by several orders of magnitude and we approximate it as being only the fluid compressibility (Burke, 2011; Robertson et al., 1958). We also assume that the variation of viscosity $\upsilon$ and compressibility with effective stress is negligible, for the conditions simulated (Burke, 2011).

The modelled fault zone is a 1.5-2 km thick sequence of 6 – 19 m interbedded anhydrite/gypsum and dolostones, within the Triassic Evaporite formation (Barchi, 2002; Trippetta et al., 2013). Seismological data and observations indicate shallow dipping (45°) faults from hypocentre depths in the Northern Apennines seismic belt, which may favour the generation of fluid overpressure leading to fault reactivation (Barchi, 2002; Miller et al., 2004; Mirabella et al., 2008). Field observations report a 1m wide fault core, where most of the slip accommodated by the fault is localised (Fig. 3.1a; Collettini et al., 2009; De Paola et al., 2008). A well-developed damage zone (DZ) is observed within thick (a few meters to tens of meters) fractured dolostones, extending at least 10 m in either direction from the fault core. Conversely, no macroscopic fracturing is observed within the foliated anhydrite layer immediately adjacent to the fault core, on either side of the fractured dolostones (Fig. 3.1a) (De Paola et al., 2008).

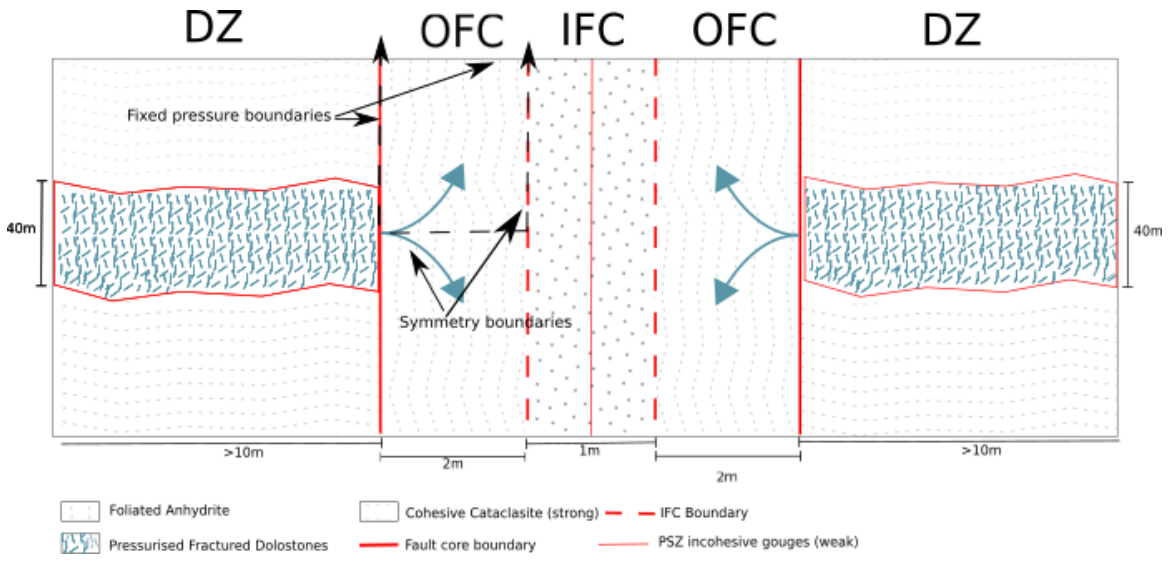



**Figure 3.2: The idealised fault segment in the base case as considered for this study with the directly simulated area in the dashed box. (Angles between**

**bedding planes and fault in damage zone (DZ) are indicative only and are not directly recreated in simulations. The simulated fault is at 45º to vertical.)**

Schematically, we further subdivide the fault core into: 1) an inner fault core (IFC), containing fine-grained cohesive cataclasites, 2) a 2m wide outer fault core (OFC) containing the IFC, made of cohesive foliated anhydrites which are not fractured (Fig. 3.1b) (De Paola et al., 2008). Within the IFC slip is localised along straight, millimetre scale principal slip zones (PSZ) of ultra-fine grained incohesive anhydrite and dolomite-rich gouges in the IFC (Fig. 3.1b) (De Paola et al., 2008). We approximate the IFC as made of cohesive cataclasites and contains a single PSZ of zero thickness, made of incohesive fault gouges (Fig. 3.2).

The fault models assume fault-valve behaviour (Sibson, 1990) so that any overpressure within the fault core is released after the seismic event. As a consequence, initial pore pressure within the fault core is assumed to be distributed uniformly and hydrostatically (Miller et al., 2004), with an imposed extensional fault unloading rate of 0.15 MPa/year, based on the tectonic setting (Chiaraluce et al., 2003).

We simulate the OFC and IFC directly with boundary conditions defined using pore pressure conditions from the damage zone (DZ). This is comprised of interbedded fractured dolostones and foliated anhydrites, the overpressure is largely contained within the fractured dolostone layer, and the anhydrites are taken to be hydrostatic. In our base case, we simulate an area of 2.5 by 1000m, representing the upper left quadrant of the fault core, with the width of the OFC (2.5m) varied in some parameter studies (Fig. 3.2). Pressure boundary conditions are hydrostatic on fault parallel boundaries, except for a 40 m thick overpressured region on the

damage zone/outer fault core boundary, which models the effects of an external, infinitely wide, permanently overpressured reservoir made of fractured dolostones (Fig. 3.2; Trippetta et al., 2013).

The idealised fault section has two planes of symmetry, the fault parallel plane bisecting the fault, and the fault perpendicular plane bisecting the overpressure contacts. We exploit this symmetry to reduce computational costs. We take these planes as symmetry boundaries, with the gradient of pore pressure normal to these boundaries set to zero.

Domain dimensions are chosen large enough that the top and bottom boundary are sufficiently removed from overpressure to not significantly affect pressure distribution within the model domain. Specifically, the length of fault that is considered is selected ensuring that the pressure gradient at the upper perpendicular boundary is less than 1% of hydrostatic pressure per metre. All models are run from an initial stress state with minimum principal stress set at 85% of lithostatic load (Miller et al., 2004).

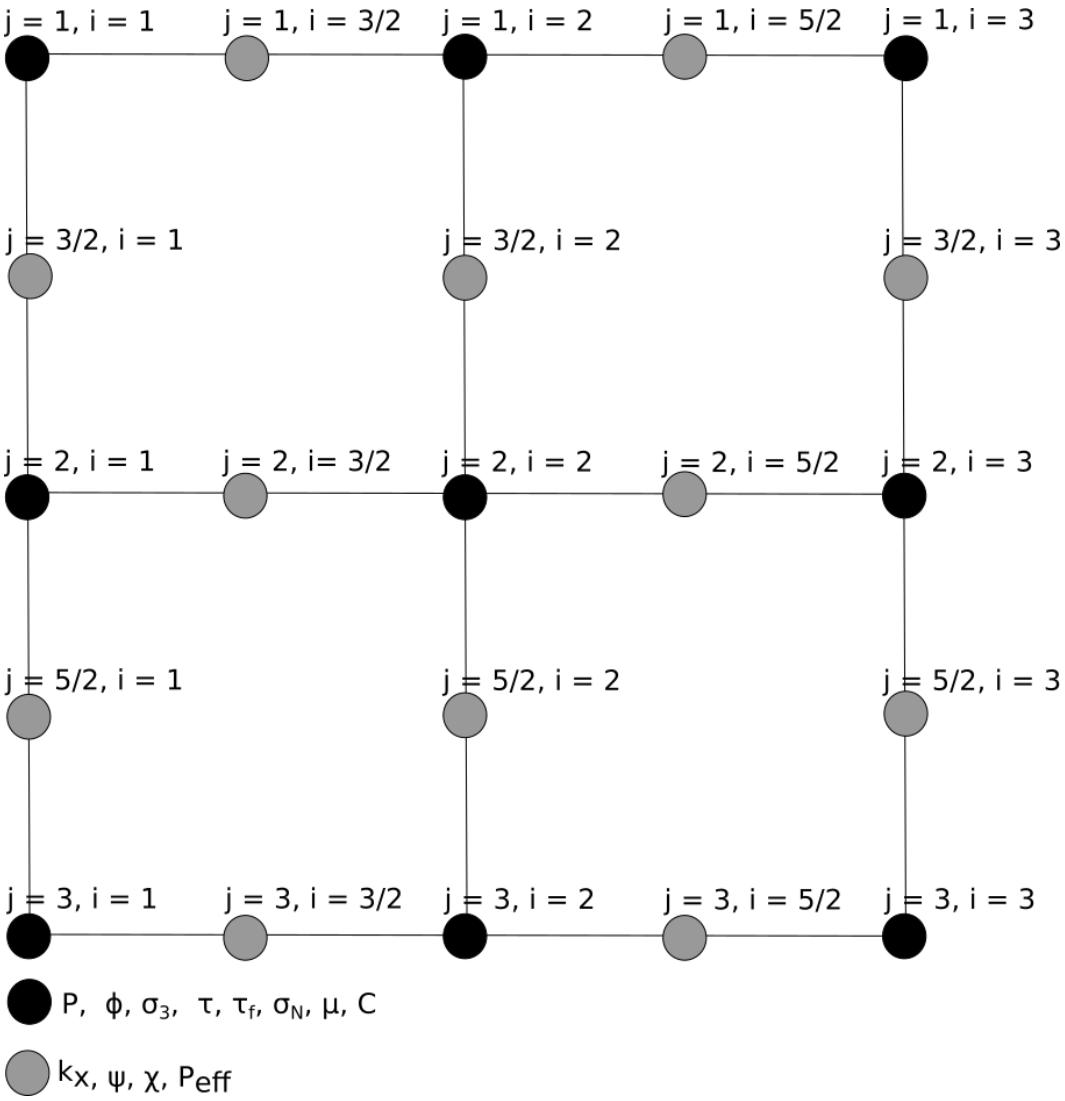



**Figure 3.3: Diagram of spatial grid used in numerical simulations, indicating variables defined at points and mid-points.**

### 3.5 Numerical Method

We discretise the second order partial differential equation (Eq. 3.4; Table 3.1) over the spatial grid defined above:

$$\frac{\Delta P_{j,\ i}}{\Delta t} = \frac{1}{\beta\eta}\left(\frac{k_{j+\frac{1}{2},\ i}\left(P_{j+1,\ i} - P_{j,\ i}\right) - k_{j-\frac{1}{2},\ i}\left(P_{j,\ i} - P_{j-1,\ i}\right)}{\varphi_{j,\ i}\left(z_{j+\frac{1}{2},\ i} - z_{j-\frac{1}{2},\ i}\right)} + \frac{k_{j,\ i+\frac{1}{2}}\left(P_{j,\ i+1} - P_{j,\ i}\right) - k_{j,\ i-\frac{1}{2}}\left(P_{j,\ i} - P_{j,\ i-1}\right)}{\varphi_{j,\ i}\left(x_{j,\ i+\frac{1}{2}} - x_{j,\ i-\frac{1}{2}}\right)}\right) \qquad (3.14)$$

We then define physical variables on an $N_x$ by $N_z$ spatial grid, and its midpoints, where $N_x$ is the number of fault perpendicular array points and $N_z$ is the number of fault parallel array points (Fig. 3.3). We define the fault parallel and perpendicular directions relative to the plane of the primary slip zone. All physical variables except for the parallel and perpendicular components of permeability, the failure state variable, architecture component and effective stress are set on the grid points and not midpoints. Values are interpolated to midpoints by averaging as required. Multiple simulations were run in all cases with a decreasing grid size until further reductions in the grid spacing did not affect simulation results.

The initial nonlinear pore pressure diffusion relationship (Eq. 3.4; Table 3.1), being the only differential equation, is the only one that is not defined entirely on either central or midpoints. In all other cases, all variables in the relationship can be taken to be at the same point or midpoint under consideration.

This discretisation gives us a set of $N_z$ by $N_x$ equations to solve at each simulated timestep. The change in pressure at a given point is dependent only on the pressure at the point itself ($i, j$) and its four neighbouring points ($i\pm1, j\pm1$). For computational ease and to allow us to employ standard ODE (ordinary differential equation) solvers on this set of equations. Ordinary differential equations are composed of formulas in only one variable and derivatives of that variable, and ODE solvers are software tool for integrating these equations.

The variables represented by Eq. 3.5-13 are evaluated at each time step and used to estimate the rate of change of pressure with time at each array point (Eq. 3.4; Table 3.1) and passed to the ODE23tb solver which returns the value at the subsequent time step. Time step size is determined by the solver to ensure numerical stability.

We evaluate shear stress (Eq. 3.8, 3.12; Table 3.1) and shear strength (Eq. 3.10; Table 3.1) at all array points where failure might occur. Shear and normal stress and shear strength between the points undergoing failure are interpolated to precisely determine the length of the failure patch and nucleation length.

We apply two types of boundary condition in the simulations in this thesis, constant pressure and volumetric flux (Dirichlet) boundary conditions and volumetric flux symmetry (Neumann) boundary conditions. The former boundary conditions are applied by setting a constant value for a physical parameter at a given boundary array point, while the latter entails a more complicated condition on flux divergence. This condition arises from the assumption that the volumetric flux vector $\underline{q}$ is equal and opposite immediately on the other side of the symmetry boundaries, in our case at the bottom and right:

$$(\nabla \cdot q)_j = \frac{2\, q_j}{\Delta(\Delta \mathrm{j})};\ j = x\ on\ a\ z\ boundary\ or\ z\ on\ a\ x\ boundary \quad (3.15)$$

$$\Delta\tau = \tau_f - \tau \quad (3.16)$$

$$\Delta L_f = \frac{L_N - L_f}{L_f} \quad (3.17)$$

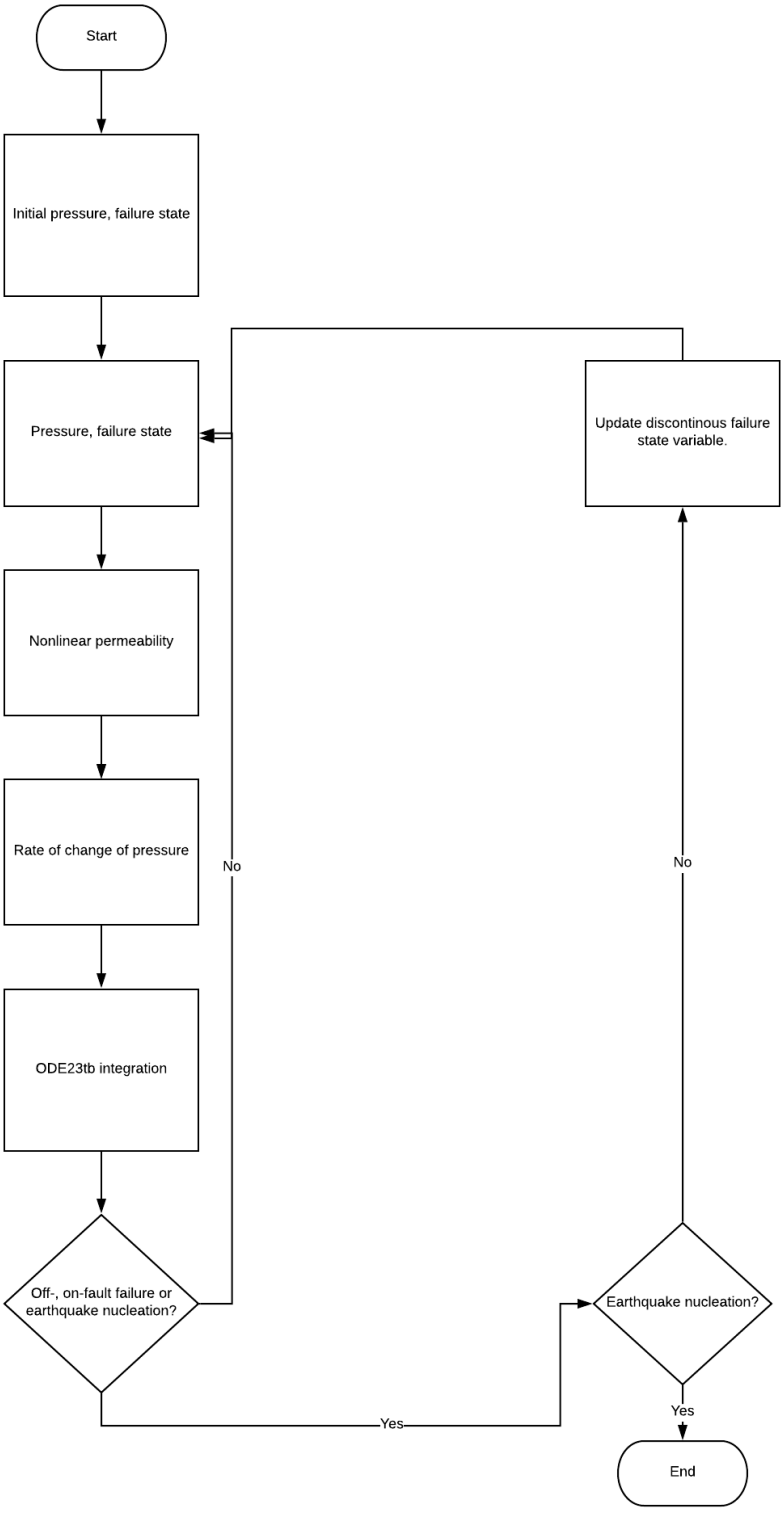



**Figure 3.4: Flow-chart of failure-event switching in fault fluid flow and earthquake simulations.**

### 3.5.1 Technical Considerations

Event-location is used to determine the failure events which drive each of these discontinuous failure state transitions. We treat the system as non-smooth, to ensure the problem is mathematically well-posed and computationally efficient. Simulations must be run to extremely low relative tolerances (5x10^-9) to consider the earthquake nucleation phase directly in simulations (directly simulating the earthquake nucleation phase as opposed to inferring it from seismological relationships). Nucleation events might last seconds over a simulated period of several hundred years.

$$\xi_e = \begin{cases} min(\Delta\tau)_{(x,z)prefail};\ for\ |\Xi\Delta L_f| > |min(\Delta\tau)_{(x,z)prefail}|\ and\ \Xi\Delta L_f > 0 \\ \Xi\Delta L_f; else \end{cases} \quad (3.18)$$

Off-fault, on-fault and earthquake nucleation are non-smooth processes (the evolution of a physical quantity transitions discontinuously from one value to another). These transitions arise due to discontinuous approximations of the true system to limit the range of space- and timescales we consider. The event-location function of the MATLAB ODE solver suite is used to detect the point at which either failure occurs at an array point or the onset of unstable sliding (an earthquake) when failure length exceeds nucleation length. The locations of these events halt the time integration of the system of ODEs and the discontinuous failure state and physical quantities which depend on it (permeability and porosity (Eq. 3.6; Table 3.1)) are updated. The MATLAB event-location function requires specifying a function which returns a single value $\xi_e$ which is zero for all failure events, positive for prefailure and negative for post-failure (Eq. 3.18; Table 3.1). The function Eq. 3.18 considers both off-fault and fault failure ($\Delta\tau$, Eq. 3.16; Table 3.1) and earthquake nucleation

($\Delta L_f$, Eq. 3.17; Table 3.1) by isolating the array point that has just begun failure and excluding points which have already failed. It must also ensure that both failure and earthquake nucleation quantities vary over the same approximate range of values for the consistent application of relative tolerance, we enforce this by multiplying the earthquake nucleation parameter $\Delta L_f$ by a factor $\Xi$. As events are detected at the instant when $\xi_e$takes a zero value timing is unaffected by this factor (any multiple of zero is still zero.)

Switching and updating on failure events like this ensures both that the integration is well-posed and increases computational efficiency as typically fewer evaluations are required. The switching mentioned above is necessary as at the instant of failure we model a discontinuous permeability change, the ODE solvers used assume that all changes are continuous. Ignoring such changes would result in numerical errors in the returned solution, equivalent to the solver encountering a singularity in permeability. Solvers in MATLAB's ODE suite evaluate the pressure derivative at multiple minor steps between full step evaluations. To calculate pressure before and after this event takes fewer time steps than it would were the solver to model it instead as a near-discontinuous but finite time derivative in the corresponding physical variable (e.g. permeability), particularly if the suggested initial step size post-event is judiciously specified.

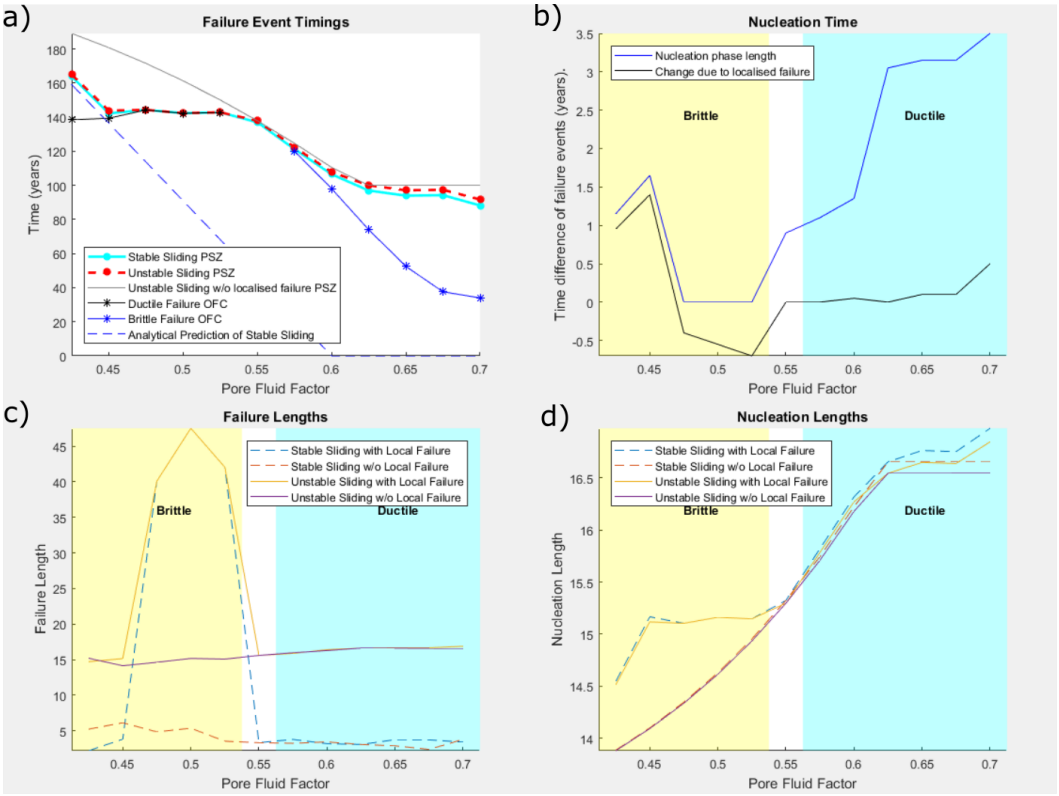



**Figure 3.5: Variations of earthquake parameters vs. pore fluid factor, for simulations demonstrating numerical instability with the ODE15s solver. Length of interseismic period (a), duration of the nucleation phase (b), length of rupture patch at failure (c) and length of the rupture patch at nucleation) are plotted against variation of the pore fluid factor across multiple simulations.**

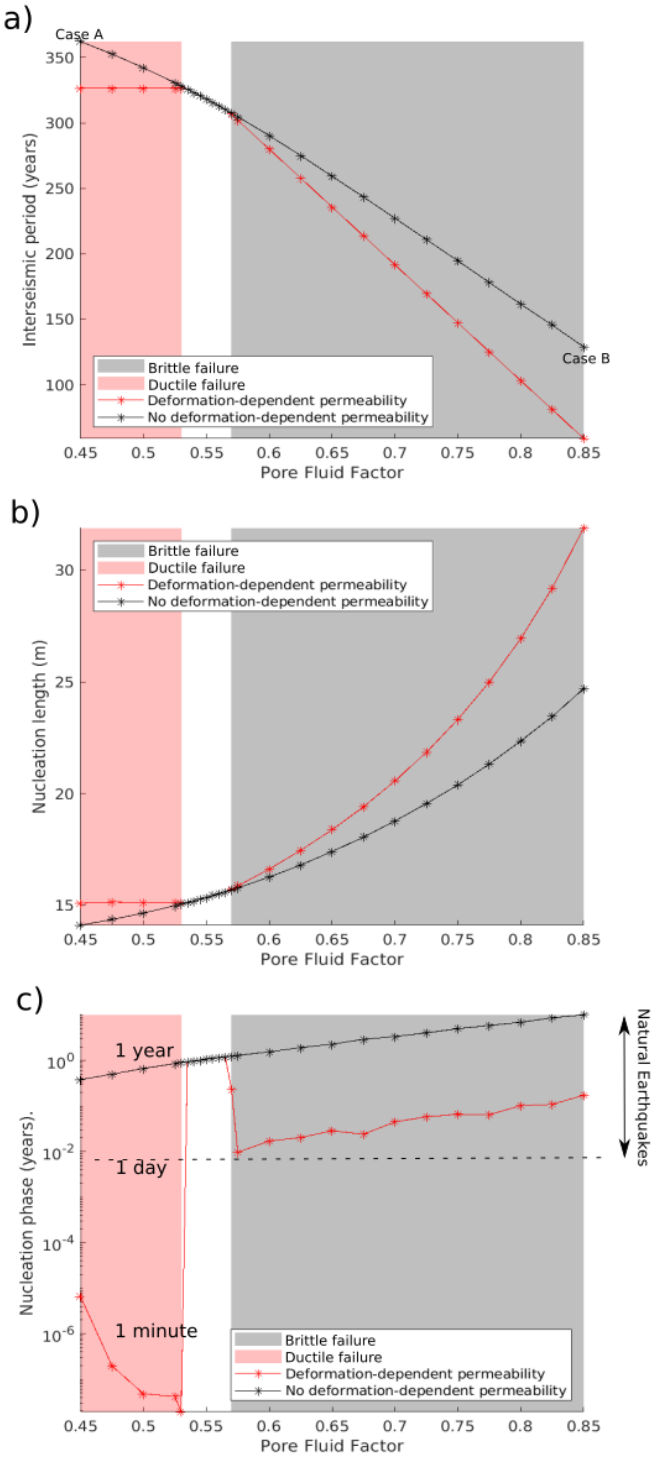



**Figure 3.6: Variations of earthquake parameters vs. pore fluid factor, for numerically stable simulations, with the ODE23tb solver. Length of interseismic period (a), nucleation length (b) and duration of nucleation phase (c) are plotted against variation of the pore fluid factor across multiple simulations.**

### 3.5.2 ODE Solver Selection and Numerical Stability

The Jacobian matrix, the first order partial derivatives of pore pressure at each of the grid points, varies over time due to the nonlinear dependence of transport properties on pressure, as do its eigenvalues. The instantaneous values of the Jacobian matrix and its eigenvalues can be inspected directly during simulation. The discrete failure state formulation leads to an extremely stiff set of differential equations, with Jacobian eigenvalues at certain points during simulation differing by up to 11 orders of magnitude, when applied to the case studies in this thesis. These stiff equations are solved using MATLAB's ODE23tb solver, an implementation of the ESDIRK23 algorithm (explicit singly diagonal implicit Runge-Kutta method Bagterp Jørgensen and Rode Kristensen, 2018; Kristensen et al., 2004).

When numerically integrating systems of ODEs it is expected that for numerical stability a smaller step size is required in regions where the solution curve shows more variation and vice versa. Systems are stiff if they require a small step size for numerical stability even in smooth solution curve regions, systems of equations exhibiting this phenomenon. The system of ODEs defined above exhibit extremely stiff behaviour as applied to the case studies in this thesis (Lambert and D., 1991), consistently the eigenvalues of the Jacobian matrix considered above can differ by up to 11 orders of magnitude in these cases.

The stiffness of a system of equations is a primary factor in ODE solver selection. MATLAB was selected as a language to write the numerical simulations in this thesis for their ODE suite and the ability to provide a sparse Jacobian pattern to the solver to increase computational efficiency. The columns of the $N_z$ by $N_x$ physical variable arrays can be stacked into at $N_zN_x$ long vector, with corresponding $N_zN_x$ by

$N_zN_x$ Jacobian matrix pattern (indicating with ones the non-zero elements of the Jacobian matrix):

$$J_{pat} = \begin{array}{c} \\ \\ \\ \\ \\ N_Z \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccccc} & & & & & N_Z & & \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots & 1 & 0 & \cdots \\ 1 & 1 & 1 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 1 & 1 & 1 & \cdots & 0 & 0 & \cdots \\ 0 & 0 & 1 & 1 & \cdots & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \cdots & \cdots & \cdots \\ 1 & 0 & 0 & 0 & \vdots & 1 & 1 & \cdots \\ 0 & 1 & 0 & 0 & \vdots & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array} \quad (3.19)$$

The use of a Jacobian pattern reduces the number of calculations required at each time step by the ratio of the sum of non-zero elements to total elements, in this case, $\sim \frac{5}{N_x N_z}$.

MATLAB provides a series of stiff ODE solvers, and ODE23tb was selected an explicit singly diagonal implicit Runge-Kutta (ESDIRK) solver, being the only solver which is A-, L-, S-stable (Hosea and Shampine, 1996; Shampine and Reichelt, 1997). ODE23tb is an implementation of the ESDIRK23 algorithm specifically (Bagterp Jørgensen and Rode Kristensen, 2018; Kristensen et al., 2004). The A-stability requirement ensures that solutions corresponding to all negative Jacobian matrix eigenvalues tend to zero as time tends to infinity and L-, S-stability place conditions on the solver which ensure that these solutions would approach zero in a single step as step size goes to infinity, ensuring no numerical oscillation.

Preliminary testing with the ODE15s and ODE23s solvers showed numerical instability in low-tolerance nucleation phase simulations (Fig. 3.5 vs. Fig 3.6) arising from the lack of A-, L-, S-stability in the underlying adaptive order and Rosenbrock methods (Shampine and Reichelt, 1997) respectively. The examples in Fig. 3.5-6 are

ran from different initial tectonic stresses, but the relative variation in solutions is clear. The greatest errors were introduced when failure events were incorrectly located during simulation, as mistiming these events during which permeability would increase by several orders of magnitude would have a huge cumulative impact. We suspect this instability resulted from the inaccurate estimation of truncation error associated with the timestep expansion for extremely stiff problems and hence relative tolerance during simulation. However, MATLAB solvers being closed source cannot be interrogated directly during simulation.

### 3.5.3 Parallelisation

Any numerical simulation result must either be robust to changes in uncertain variables otherwise we will need to constrain the behaviour we observe in these simulations over a range of values for each uncertain parameter. With any numerical case study using uncertain observations of natural subsurface faults, there is a need to explore the impact of varying the most uncertain parameters, so that we can distinguish behaviours and conclusions that would apply generally to the class of similar faults, from those that would apply only to faults with physical properties very similar to the assumed base case. Parameter studies allow us to expand our dataset for numerical experiments; we can build an initial base case using observations of a fault from the natural world and alter one or more parameters over a range of values likely found in other natural faults.

A parameter study is a set of simulations which are necessarily independent of one and other, as such they are candidates for parallelisation. Scripts were constructed using MATLAB's object-oriented language features to ensure each simulation existed as a distinct object in memory, isolated for parallelisation, taking each of the varied parameters as an input argument. This object isolation in memory allows

discontinuous physical variables to be updated and persist between each time-integration (between located failure events) as object properties and not as global variables, without affecting simulations with different sets of parameters run concurrently. Both parallelisation and performant scripts were necessary as a single high spatial resolution (175 x 200), low relative tolerance simulations (~5E-9) would require of the order of $10^7$ simulation steps. Computation time for a single parameter study (typically involving varying a parameter over a range of roughly 20 values) could be conducted on a desktop computer, in less than three days at worst and an hour at best. The number of localised failure events and hence dominant mode of failure significantly affects the number of discontinuous permeability transitions and hence time steps required for simulation. Brittle failure dominated parameter studies might take on the order of days where ductile failure dominated studies were typically on the order of hours.

### 3.6 Model Testing and Verification

To validate the MATLAB scripts used in simulations the fluid flow model, the component of simulation produced directly from the integration of differential equations, is tested under all conditions where analytical solutions exist (constant, nonlinear, discontinuous permeability). All other physical variables used in simulations are produced from direct algebraic equations were tested in development by ensuring that the correct answer was returned for a given input.

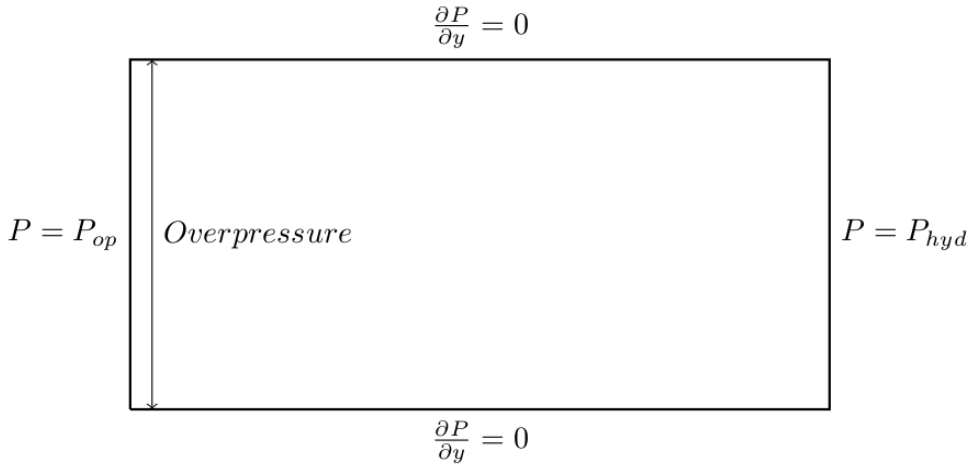



**Figure 3.7: Boundary conditions used to test fixed pressure Dirichlet boundary conditions. Top and bottom boundary conditions chosen to produce solutions independent of y.**

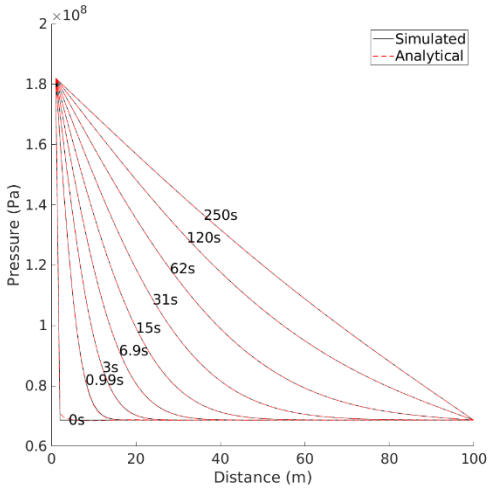



**Figure 3.8: Simulation results for homogeneous isotropic permeability case at y=50m, compared to known analytical solution. Analytical results (black) and simulation results (red) are colinear excepting externally imposed initial conditions.**

### 3.6.1 Constant permeability

To validate the pore pressure diffusion code and boundary conditions a reduced complexity version of the model was considered. A 100 m by 100 m region, in the x

and y directions, overpressured at the left boundary, with the right boundary held at hydrostatic pressure and initially hydrostatic everywhere else. Neumann symmetry boundary conditions for pressure are imposed at the top and bottom boundary, representing seals, to produce a pore pressure distribution that is independent of vertical position (Fig. 3.7). For this test we selected. homogenous, isotropic permeability of $10^{-16}$ $m^2$.

The above initial-boundary-value problem has the following solution:

$$P(x,t) = P_{hydro} + P_{op}\left(1 - \gamma - \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{\sin(\pi n\gamma)}{n}e^{-(\pi n)^2\alpha t}\right) \qquad (3.20)$$

Where

$$\gamma = \frac{x-1}{L_x - 1}$$

and

$$\alpha = \frac{k_x}{\beta\eta\varphi}$$

Figure 3.8 shows the simulation results at each timestep that was evaluated by the solver, over the horizontal line y=50m. The analytical results and simulation result are colinear except under initial conditions and vary by less than the ODE solver relative tolerance, we take this is validation that the numerical simulation code reproduces both transient and steady state behaviour for simple homogeneous porous media. The simulation exhibits a variation from initial conditions when compared to the analytical solution, this is as the externally imposed initial conditions are not actually a valid solution to the physical system.

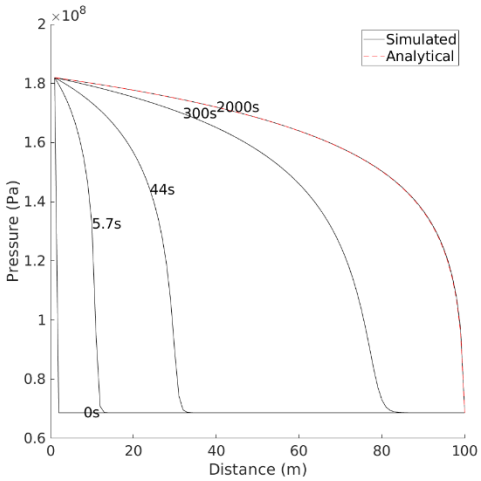



**Figure 3.9: Simulation results for nonlinear homogeneous isotropic permeability case at y = 50m, compared to known analytical solution. Analytical results (black) and simulation results (red) are colinear at steady state (where analytical solution exists).**

### 3.6.2 Pressure-dependent permeability

To validate simulations of nonlinear pressure dependent permeability, the same scenario was considered now considering a permeability model of the form $k = k_0 e^{-\gamma P}$. Setting a permeability of $10^{-16}$ m$^2$ at zero pressure with pressure sensitivity $\gamma = 5\times10^{-8}$. This initial-boundary-value problem has the following steady-state solution:

$$P = \ln \frac{(cx+d)}{\gamma} \quad (3.21)$$

where:

$$c = \frac{\exp(\gamma P_{hyd}) - \exp(\gamma P_{op})}{L}$$

and:

$$d = \exp(\gamma P_{op})$$

Figure 3.6 similarly shows the simulation results at each timestep that was evaluated by the solver, over the horizontal line y=50m. At steady state, the analytical results and simulation result are almost perfectly colinear and vary by less than the ODE solver relative tolerance. No analytical solution exists for the transient problem. We take this consistency between solutions as validation that the numerical simulation code reproduces steady state behaviour when compare to situations with pressure dependent permeability.

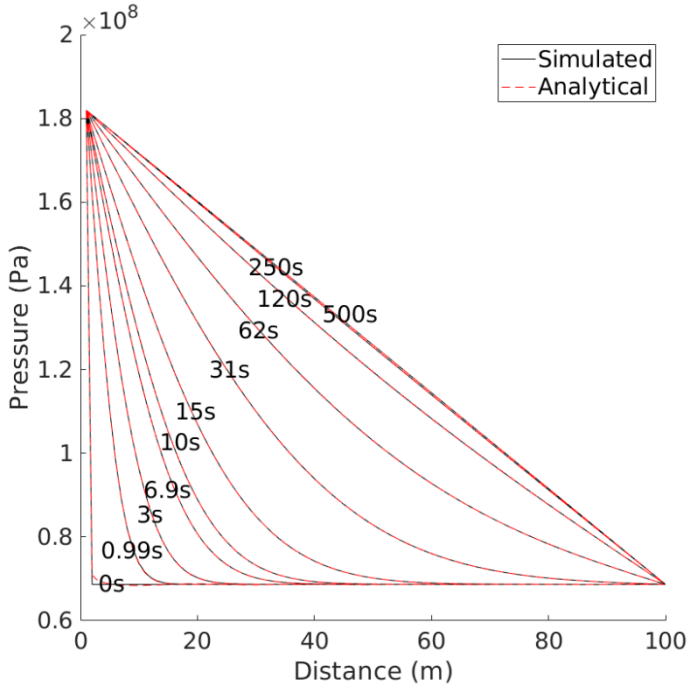



**Figure 3.10: Simulation results for discontinuous change in homogeneous isotropic permeability case at y=50m, compared to known analytical solution. Analytical results (black) and simulation results (red) are colinear except for externally imposed initial conditions.**

### 3.6.3 Discontinuous permeability transition

To validate the discontinuous transitions in permeability introduced in our simulations, the case of isotropic, homogenous, constant permeability is once more considered, taking the same initial value as the previous case. However, after ten seconds the permeability is instantaneously doubled throughout the modelled area. Prior to this change in permeability the solution equation 3.20 holds, subsequently the following solution holds:

$$P(x,t) = P_{hydro} + P_{op}\left(1 - \gamma - \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{\sin(\pi n\gamma)}{n}e^{-(\pi n)^2(\alpha' t + \alpha t_0)}\right) \qquad (3.22)$$

Where

$$\alpha' = \frac{k_x'}{\beta\eta\varphi}$$

with $t_0$=10s, the time of the instantaneous permeability change. Again, as the solutions are colinear to within solver tolerance we take this as validation that fluid flow script can accurately recover known analytical solutions for both transient and steady state conditions (Fig. 3.10). Due the large disparity in timescales between initial fast fluid flow changes before a quasi-steady state is reached and the unloading of the fault, to a somewhat close approximation (at least the closest approximation with analytical solution) the step change in permeability is effectively a timed change in permeability.

Further as each of the fundamental permeability behaviours of the simulated system match known analytical solutions where available we take this as validation of the partial differential equation component of our fluid flow solver script in general.

Table 3.1: Symbols of variables and constants

| Symbol | Description |
|---|---|
| $\mathbf{x}$ | Fault perpendicular spatial coordinate |
| $\mathbf{z}$ | Fault parallel spatial coordinate |
| $\boldsymbol{\chi}$ | Fault zone architecture component index |
| $\boldsymbol{\psi}$ | Mode of failure index |
| $\boldsymbol{\beta}$ | Compressibility |
| $\boldsymbol{\eta}$ | Viscosity |
| $\boldsymbol{\phi}$ | Porosity |
| $\boldsymbol{\mu}$ | Friction coefficient |
| $\mathbf{C}$ | Cohesion |
| $\mathbf{G}$ | Shear modulus |
| $\mathbf{D_C}$ | Critical slip distance |
| $\mathbf{F}$ | Difference of the rate and state parameters |
| $\boldsymbol{\xi}$ | Event location function |
| $\boldsymbol{\Xi}$ | Earthquake nucleation scale factor |
| $\boldsymbol{\zeta}$ | Failure patch geometry factor |
| $\boldsymbol{\gamma}$ | Pressure sensitivity |
| $\mathbf{t}$ | Time |
| $\mathbf{P}$ | Pressure |
| $\mathbf{P_e}$ | Effective pressure |
| $\mathbf{k}$ | Permeability tensor |
| $\boldsymbol{\sigma_N}$ | Normal stress |
| $\boldsymbol{\sigma'_N}$ | Effective normal stress |
| $\boldsymbol{\sigma_1}$ | Principal maximum stress |
| $\boldsymbol{\sigma_3}$ | Principal minimum stress |
| $\boldsymbol{\tau}$ | Shear stress |
| $\boldsymbol{\tau_f}$ | Shear strength |
| $\mathbf{L_N}$ | Nucleation length |

# CHAPTER 4

*Modelling fluid flow in complex natural fault zones: implications for natural and human-induced earthquake nucleation.*

# 4. Modelling fluid flow in complex natural fault zones: implications for natural and human-induced earthquake nucleation.


**Abstract**

Pore fluid overpressures in active fault systems can drive fluid flow and cause fault weakening and seismicity. In return, deformation accommodated by different mode of failure (e.g. brittle vs. ductile) also affects fault zone permeability and, hence, fluid flow and pore fluid pressure distribution. The resulting non-linear, complex feedback between fluid flow, fluid pressure and fault deformation control the length of the nucleation phase of an earthquake and the duration of the interseismic period. Current numerical simulation techniques model how fluid flow controls fault reactivation and associated seismicity. However, the control exerted by pore fluid pressure on the transition from aseismic slow fault sliding to seismic fast sliding, during the earthquake nucleation phase, is still poorly understood. Here, we model overpressured, supercritical $CO_2$ fluid flow in natural, exhumed faults in evaporite sequences, which represent an analogue of the seismic sources at hypocentre depth of recent seismic events in the Northern Apennines of Italy (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia earthquakes). Our modelling results of Darcy fluid flow show that the duration of the nucleation phase is significantly reduced, from a few years to a few months, when realistic models of fault zone architecture and pore pressure- and deformation-dependent permeability are considered. Interestingly, a few months is also the time scale of aseismic slip measured during the nucleation phase of some recent large earthquakes (e.g. Fig. 4.7c; Mavrommatis et al., 2014; Kato et al., 2012; Socquet et al., 2017). These findings have significant

implications for earthquake early warning systems, as any significant extension of the nucleation phase can increase the likelihood of precursory signal detection. In addition, our results have important implications for short- and long-term earthquake forecasting, as crustal fluid migration during the interseismic period may control fault strength and earthquake recurrence intervals.

## 4.1. Introduction

Pore fluid pressure plays a primary mechanical role during faulting as it reduces the frictional fault strength (Cox, 2010; Sibson, 1990, 1992). There is strong geological and geophysical evidence that fluid migration in the upper crust controls faulting (Collettini et al., 2009; Cox, 1995; Cox et al., 1987; De Paola et al., 2008; Hickman et al., 1995; Miller, 1996; Rice, 1992; Sibson, 1992, 1990, 2000), and natural (Di Luccio et al., 2010; Miller et al., 2004; Nur and Booker, 1972) and human induced (Ellsworth, 2013; McGarr et al., 2015; Sumy et al., 2014) seismic activity.

Fluid circulation within the upper crust is strongly dependent on the transport properties of rocks (i.e., permeability). Rock permeability and porosity vary with pressure conditions and deformation (De Paola et al., 2009; Fischer, 1992; Hangx et al., 2010; Morrow and Lockner, 1997, 1994; Paterson and Wong, 2005; Zhu et al., 1997), which control the development and connectivity of fracture patterns across a range of scales (Caine et al., 1996; Cox, 1995; Mitchell and Faulkner, 2008; Peach and Spiers, 1996; Wong et al., 1997; Zoback and Byerlee, 1975).

In previous modelling efforts, the link between fluid flow and faulting has been investigated using coupled deformation and fluid flow modelling software, such as TOUGH-FLAC (Cappa and Rutqvist, 2011a, 2012; Rutqvist et al., 2015, 2013a, 2002). Coupled fluid flow and geomechanical fault slip (e.g. fault reactivation) analysis have been used, for example, to model the spatial evolution of both in situ stresses and fluid pressure, to estimate the maximum sustainable injection pressure during geological sequestration of $CO_2$ (Rutqvist et al., 2007). In these studies, fluid

flow was modelled for metre to kilometre scale fault zone features, considering permeability as a continuous function of porosity, volumetric strain, average effective stress (Davies et al., 2001), and fault shear strain (Rutqvist et al., 2007). This approach has been extended to also include the effect of simplistic fault zone architectures (Cappa et al., 2009; Cappa and Rutqvist, 2011; Hsiung et al., 2005; Mazzoldi et al., 2012; Rinaldi et al., 2014; Rutqvist et al., 2013, 2009; Leclère et al., 2015). Overall, previous results show that pressure increase due to shear-enhanced permeability plays an important role, as it can facilitate the propagation of fault instability and extend permeability enhancement through the overlying caprock.

Here, we model fluid flow in exhumed faults in evaporite sequences with complex architecture, and pore pressure- and deformation-dependent permeability. These faults represent an analogue of the seismic sources at hypocentre depth of recent seismic events in the Northern Apennines of Italy (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia earthquakes). Modelled fluid flow is then used to investigate the effects of pore fluid pressure distribution during the nucleation phase that precedes an earthquake. During this phase, stable sliding spreads out from an initial small patch until it reaches a critical size, the nucleation length, at which unstable fast sliding results in the propagation of the rupture (Marone, 1998; Scholz, 1998).

Identifying the factors that control the duration of the nucleation phase of earthquakes has significant implications for earthquake early warning systems, as any significant extension of the nucleation phase can increase the likelihood of early premonitory signal detection. Furthermore, modelling pore fluid pressure evolution during the interseismic period has relevant implications for long term earthquake forecasting, as it controls fault strength and earthquake recurrence intervals.

### 4.2. Numerical method

We perform numerical simulations of nonlinear diffusion to model fluid flow in fault zones with realistic, complex fault zone architecture (all symbols and values used are explained in Table 1). In our model, fault zone permeability is assumed to vary as a function of effective stress and mode of failure (e.g. brittle and localised vs. ductile and distributed).

#### 4.2.1. Porous media flow and numerical solution

We develop an approach based on the diffusion of pore pressure within a classical porous medium using Eq. 3.4 which relates pore pressure $p$ and permeability $k$ to the rate of change of pressure with time $t$. The compressibility $\beta$ is approximated as being only the fluid compressibility, because the compressibility of supercritical $CO_2$ exceeds that of evaporite rocks by several orders of magnitude (Burke, 2011; Robertson et al., 1958). It is also assumed that the variation of viscosity $\upsilon$ and compressibility with effective stress is negligible for the range of conditions simulated (Burke, 2011), where effective stress as defined in Eq. 2.2.

Following the experimental permeability relations observed in low porosity evaporite rocks (De Paola et al., 2009; Hangx et al., 2010), we consider that the solid rock is an ideal porous medium. Its permeability can be expressed as a function of effective stress in the presence of ductile deformations, accommodated by small, distributed fracture patterns (Detournay and Cheng, 1993). We also consider singularities in the time derivative of permeability when localised brittle failure occurs, leading to instantaneous increase of permeability within the fault (De Paola et al., 2009).

We make the simplifying assumption that single-phase dry, supercritical CO2 saturates the modelled fault zone and that no precipitation or dissolution occurs between the fluid and rock matrix. This assumption is in accord with field data supporting large $CO_2$ fluxes in the epicentre areas of the Northern Apennines seismic belt (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.3 2009 L'Aquila extensional earthquakes), where large deep-seated $CO_2$ flux greater than 0.45 t $day^{-1}$ $km^{-2}$ have been measured (Chiodini et al., 2004; Collettini et al., 2008).

Although fault zone geometries, physical properties and mechanical behaviour used in our modelling are inferred and constrained from a seismic extensional fault zone in evaporite rocks (Collettini et al., 2009; De Paola et al., 2009, 2008), the methods and results can be generalised and applied to any natural fault zone subject to natural single phase flow and in the absence of significant rock matrix dissolution with known fault zone architecture and constrained physical and mechanical properties.

### 4.2.2. Model input parameters

#### *4.2.2.1. Fault zone architecture*

The modelled fault zone is formed within the Triassic Evaporite formation: a 1.5-2 km thick sequence of 6 – 19 m interbedded anhydrite/gypsum and dolostones (Barchi, 2002; Trippetta et al., 2013). Seismological data and observations from hypocentre depths in the Northern Apennines seismic belt indicate the presence of shallow dipping (45°) faults, which may favour the generation of fluid overpressure leading to fault reactivation (Barchi, 2002; Miller et al., 2004; Mirabella et al., 2008).

Fault zone architecture is constrained by field observations reporting a 1 m wide inner fault core, where most of the slip accommodated by the fault is localised (Fig. 2.8a; Collettini et al., 2009; De Paola et al., 2008). Outside the fault core, a well-developed damage zone (DZ) is observed within thick (a few meters to tens of meters) fractured dolostones, extending at least 10 m in either direction from the fault core. Conversely, no macroscopic fracturing is observed within the foliated anhydrite layer immediately adjacent to the fault core, on either side of the fractured dolostones (Fig. 2.8a) (De Paola et al., 2008).

The fault core can be subdivided into an inner fault core (IFC), containing fine-grained cohesive cataclasites, which is enclosed on both sides by a 2 m wide outer fault core (OFC), containing cohesive foliated anhydrites, which are not fractured (Fig. 2.8b) (De Paola et al., 2008). Within the IFC, slip is further localised along straight principal slip surfaces (PSS), which are located within thin (millimetre-scale) principal slip zones (PSZ) of ultra-fine-grained incohesive anhydrite and dolomite-rich gouges (Fig. 2.8b) (De Paola et al., 2008).

A schematic, yet realistic, fault zone architecture is used within the model (Fig. 3.2), where it is assumed that seismic slip occurs along a single PSZ of zero thickness, made of incohesive fault gouges and located in the centre of the IFC, which is made of cohesive cataclasites (Fig. 3.2).

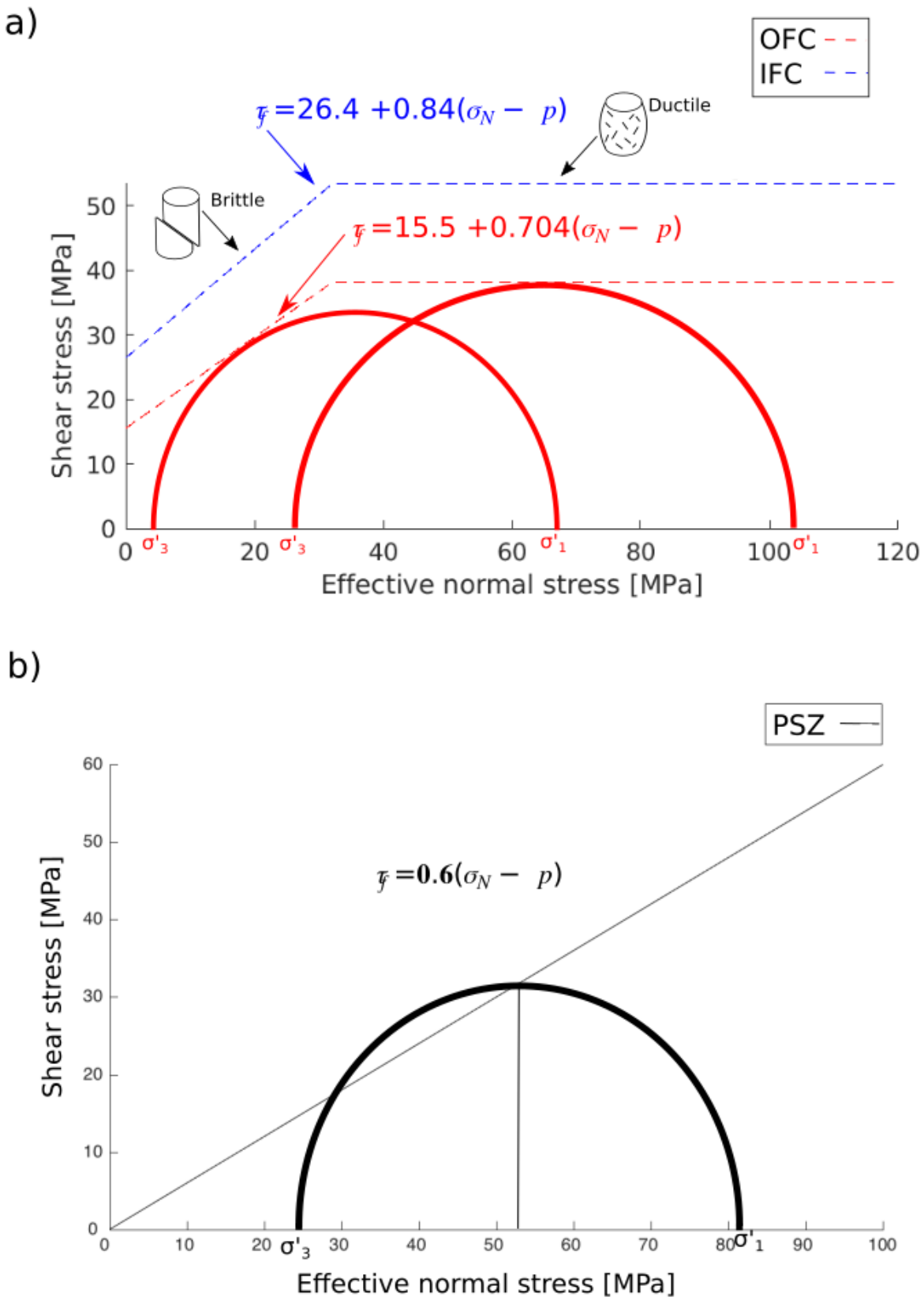



**Figure 4.1: Failure envelopes and schematic Mohr circles for the different fault zone domains: a) The brittle (localised deformation) and ductile (distributed**

**failure) regions of the OFC and IFC failure envelopes are indicated with core plug sketches. The schematic Mohr circles show the onset of brittle and ductile failure in the OFC, respectively. b) The Mohr circle shows the onset of frictional sliding along a cohesionless principal slip surface within the PSZ (Black line within the Mohr circle).**

#### *4.2.2.2. Failure envelopes and mode of failure*

The failure envelopes, mode of failure and transport properties of rocks within the OFC and IFC fault zone domains have been obtained from triaxial deformation experiments with fluid flow, performed on borehole samples of Triassic Evaporites rocks (De Paola et al., 2009; Hangx; et al., 2010).

The strength of intact anhydrite rocks is controlled by the presence and orientation of fabric anisotropy, with the weakest rocks being those where foliation is sub-parallel to the loading direction. On the other hand, the transition between localised brittle to distributed ductile mode of failure is controlled by effective stresses, and occurs at about 20 MPa regardless of grain size, presence of fabric anisotropy, and its orientation (De Paola et al., 2009).

Mohr-Coulomb failure envelopes have been constructed for each fault zone domain, i.e. the OFC, the IFC and the PSZ (Fig. 4.1). The failure envelope of the foliated anhydrite in the OFC is obtained from triaxial loading experiments performed on anhydrite borehole samples with foliation oriented sub-parallel to the loading direction (Fig. 4.1a; De Paola et al., 2009). A sharp transition from localised brittle to distributed ductile mode of failure is observed at effective stresses of about 20 MPa (Fig. 4.1a). The failure envelope of the IFC, made of cohesive, anhydrite bearing fine-grained cataclasites, with no fabric, is obtained from triaxial loading experiments performed on fine-grained, homogeneous anhydrite borehole samples

(Fig. 4.1a; De Paola et al., 2009). The failure envelope of the PSZ – the actively slipping plane within the IFC – is assumed consistent to that of a cohesionless fault plane, with Byerlee's sliding friction coefficient of 0.6 (Fig. 4.1b; Scuderi et al., 2013).

In our simulations, the frictional strength of the fault is taken to be homogeneous and without asperities. Any variation in frictional strength is modelled as being dependent only on effective normal stress, which is taken to be constant on the sliding fault patch, after sliding begins. This approximation assumes that the fault patch can no longer accumulate or dissipate stress energy locally, as any change in energy will instead accelerate or decelerate sliding. Similarly to what demonstrated in Campillo et al. (2001), and using a similar approximation to Uenishi and Rice (2003), we approximate the nucleation length ($L_N$) of a failure patch ($L_F$) as the nucleation length of the stiffest point in the patch (largest effective normal stress).

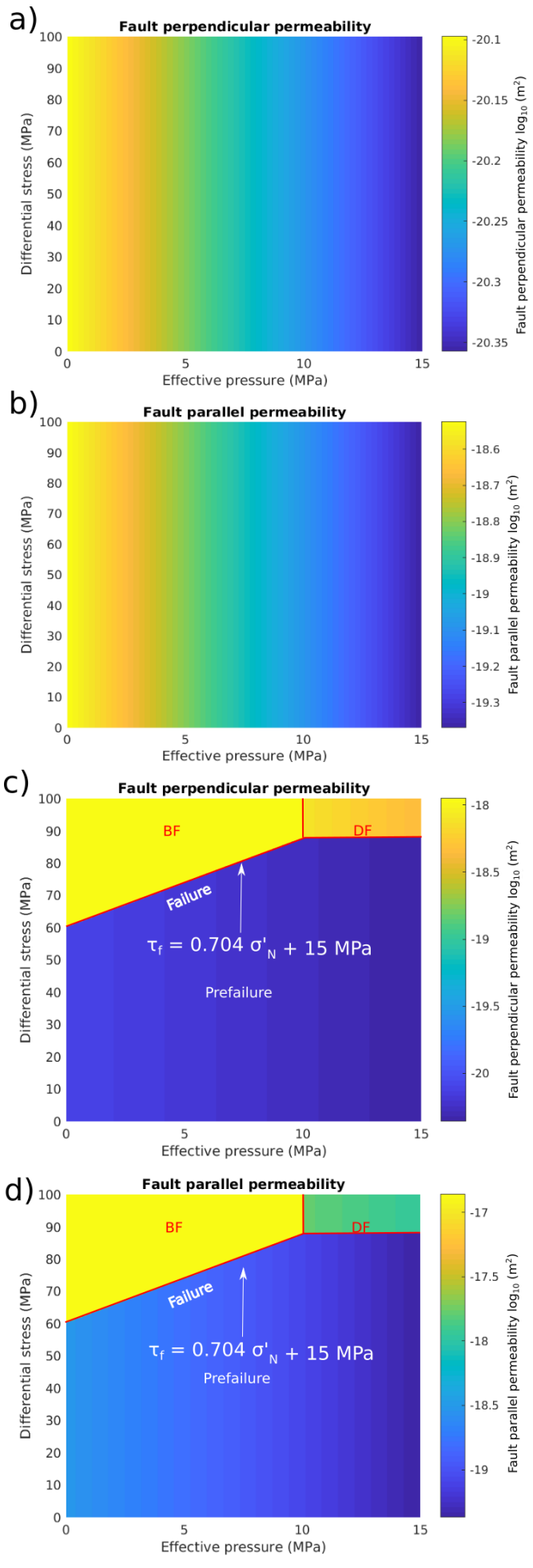

a)
Fault perpendicular permeability
Differential stress (MPa)
Effective pressure (MPa)
Fault perpendicular permeability log10 (m2)
b)
Fault parallel permeability
Differential stress (MPa)
Effective pressure (MPa)
Fault parallel permeability log10 (m2)
c)
Fault perpendicular permeability
BF
DF
Failure
τf = 0.704 σ'N + 15 MPa
Prefailure
Differential stress (MPa)
Effective pressure (MPa)
Fault perpendicular permeability log10 (m2)
d)
Fault parallel permeability
BF
DF
Failure
τf = 0.704 σ'N + 15 MPa
Prefailure
Differential stress (MPa)
Effective pressure (MPa)
Fault parallel permeability log10 (m2)

**Figure 4.2: Log-plot of permeability against differential stress and effective pressure based on triaxial experiments performed on OFC Triassic Evaporites samples. a - b) Fault parallel (a) and fault perpendicular (b) permeability evolution with effective pore pressure derived from static triaxial experiments with no loading of the sample. c – d) Fault parallel (c) and fault perpendicular (d) permeability evolution with effective pore pressure and stress dependence obtained during dynamic triaxial experiments, when samples are loaded to failure.**

**NB: The raw values for this plot are taken from De Paola et al., 2009, in which the terminology effective pressure is used in place of effective stress.**

#### *4.2.2.3 Fault zone transport properties*

The permeability tensor relations have been constructed for the OFC using available data from triaxial deformation experiments with fluid flow (De Paola et al., 2009), and are illustrated in Fig. 4.2. In particular, the fault parallel and perpendicular components of the permeability tensor in the OFC are obtained from loading experiments to failure with fluid flow imposed parallel and perpendicular to fabric, respectively. To a first approximation, laboratory experiments show that the permeability of anhydrite rocks before failure are controlled by the combined effect of (De Paola et al., 2009): 1) effective stress, as permeability decreases with increasing effective stress, due to porosity reduction (Fig. 4.2a-b); and 2) deformation, as permeability increases with increasing loading due to the creation of fractures within the rock (Fig. 4.2c-d). For a given value of pore pressure, a sudden increase in permeability is observed at failure, and its magnitude is controlled by the brittle and ductile mode of failure (Fig. 4.2c-d), respectively. The pore pressure

sensitivity of permeability *k* is described by the general, experimentally derived, empirical Eq. 3.6.

At the onset of distributed ductile failure, permeability will rapidly increase (Fig. 4.3c-d). Then, for a given value of effective stress during ductile failure, permeability will reach a plateau value when a percolation threshold state is attained in the sample, due to the development of a fully connected network of microfractures (Fig. 4.3c-d; De Paola et al., 2009). Permeability of samples deforming in a ductile mode is sensitive to effective stress variations (Fig. 4.3c-d; Table 1), which can reduce or enhance the porosity of the sample by closing or opening fractures, respectively (De Paola et al., 2009).

Conversely, at the onset of localised brittle failure, permeability will rapidly increase to a relatively high value (Fig. 4.3c-d). After the occurrence of brittle failure, we assume that permeability will not be sensitive to effective stress variations (Fig. 4.3c-d; Table 1), as the macroscopic fault/fracture can act as an effective conduit for fluid migration (De Paola et al., 2009). We also assume that all fractures created during the pre- and co-seismic phase will be fully healed soon after the main seismic event. This is due to the efficiency of hydrothermal healing processes, acting during the interseismic period, which may seal micro- and macro-scale fractures within a few years of a slip event (Keulen et al., 2008; Nakatani and Scholz, 2004; Niemeijer et al., 2008; Scuderi et al., 2013; Yasuhara et al., 2005).

The permeability of the fine-grained cataclasites in the IFC and gouges in the PSZ are assumed to be anisotropic in the fault-parallel and fault-orthogonal direction (Evans et al., 1997; Wibberley and Shimamoto, 2002), but otherwise in the OFC, they are not assumed to depend on pore pressure and deformation (Table 1).

### 4.2.3 Model setup

The model setup assumes that pore fluid overpressure within the damage zone (DZ) is largely contained within the fractured dolostone layer (Fig. 3.2). Hydrostatic pore fluid pressure occurs within the layers of foliated anhydrite in the DZ, which act as a perfect seal at their contacts immediately above and below the overpressured dolostone reservoir (Fig. 3.2). The initial pore pressure distribution within the fault core is assumed to be uniform and hydrostatic. This is due to fault-valve behaviour (Sibson, 1990), as any overpressure build-up during the interseismic period within the fault core is being quickly released during and soon after the seismic event (Miller et al., 2004).

We simulate an area of 2.5 by 1000 m, representing the upper left quadrant of the fault core (Fig. 3.2), located at a hypocentre model depth of 7 km and subject to extensional tectonic loading by reduction of the least principal stress axis at a rate of 0.15 MPa/year, based on the tectonic setting (Chiaraluce et al., 2003). All models are run from an initial stress state with minimum principal stress set at 85% of lithostatic load (Miller et al., 2004).

Our simulations only directly model the OFC and IFC, with boundary conditions defined using the pore pressure conditions from the damage zone. Pressure boundary conditions are hydrostatic on all boundaries, except for a 40 m thick overpressured part of the side boundaries, which models the effects of an external, infinitely wide, permanently overpressured reservoir made of fractured dolostones (Fig. 3.2; De Paola et al., 2008; Trippetta et al., 2013).

To reduce computational costs, we exploit the model's symmetry properties. The idealised fault section has two planes of symmetry, the fault parallel plane bisecting the fault, and the fault perpendicular plane bisecting the overpressure contacts. In our

model, these planes become symmetry boundaries, with the gradient of pore pressure normal to these boundaries set to zero.

The top and bottom fixed pressure boundaries maintain a hydrostatic pressure, and domain dimensions are chosen large enough that this boundary doesn't significantly affect pressure distribution within the model domain. For each simulation, the length of fault that is considered is selected ensuring that the pressure gradient at the upper perpendicular boundary is less than 1% of hydrostatic pressure per metre.

**4.3. Results**

A series of numerical simulations have been performed for a range of initial pore pressures, at a depth of 7 km, for a fixed tectonic unloading rate in the minimum principal stress direction. Fluid flow in the fault core during the interseismic period is modelled for two end-member scenarios, Case A and B. In the simpler Case A scenario, permeability evolves during the interseismic period in the OFC solely controlled by pore pressure variations and lithological factors (e.g. fabric presence and orientation; Fig. 4.2a-b). In the more complex Case B scenario, permeability evolution during the interseismic period is additionally controlled by deformation, via brittle or ductile failure in the OFC (Fig. 4.2c-d).

The effect of pore pressure evolution in the fault core on the duration of the nucleation phase and on the size of the nucleation patch is then investigated.

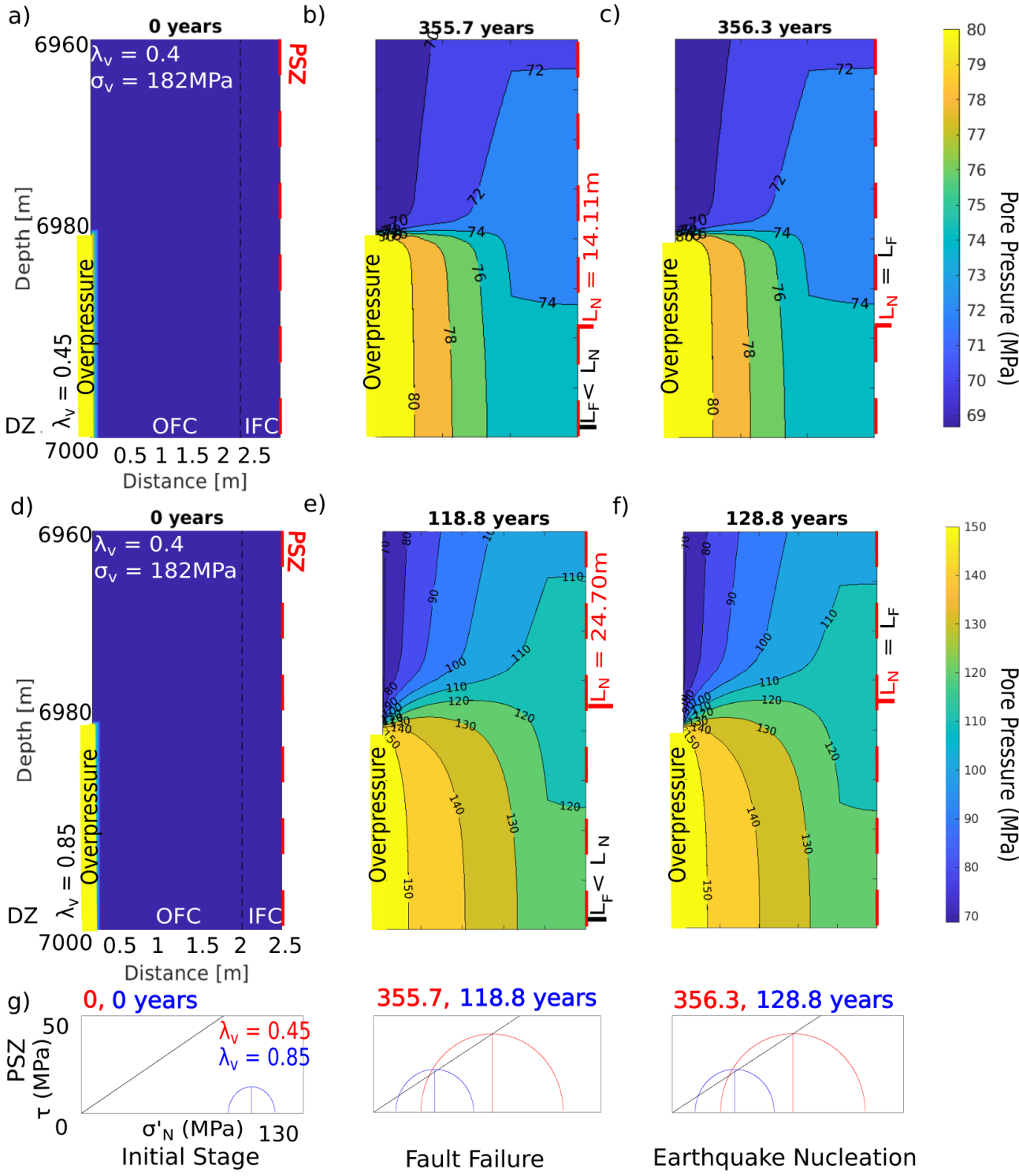



**Figure 4.3: Simulation results of pore pressure evolution and onset of failure – Simple Case A. a, d) Plots are provided for slightly supra-hydrostatic, $\lambda_v$ = 0.45 (a), and sub-lithostatic, $\lambda_v$ = 0.85 (d) initial pore pressure conditions in the damage zone reservoir, compared to initially hydrostatic ones ($\lambda_v$ = 0.4) in the**

**fault core. b, e) Pore pressure evolution from initial conditions to the time at which failure initiates in a failure patch ($L_F$), along the main principal slip zone (PSZ) in the inner fault core (IFC). Note that the size of the failure patch, $L_F$, is not to scale in these panels, as $L_F$ is infinitesimally small at the onset of failure. c, f) Pore pressure conditions at the time an earthquake nucleates, when the size of the failure patch, $L_F$, matches that of the theoretical predicted nucleation length, $L_N$. g) Mohr failure analysis for the PSZ at initial conditions (a, d), onset of fault failure (b, e) and earthquake nucleation (c, f). Results are presented for 40 m of 1 km simulated region shown vertically, and 2.5 m fault core exaggerated horizontally. During simulations a millimetre scale horizontal spatial grid was used, and vertically an initially millimetre scale logarithmic grid was used.**

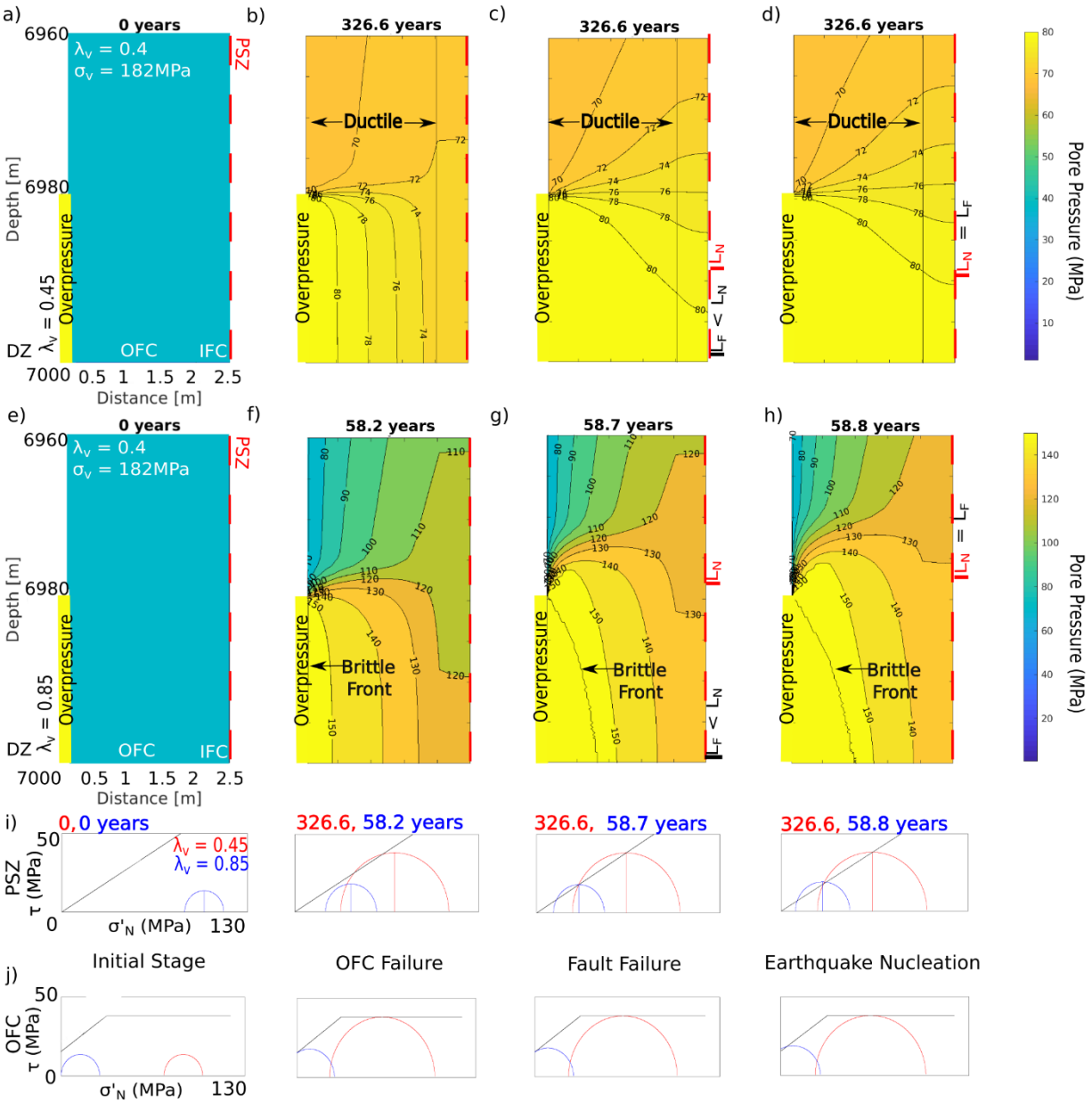



**Figure 4.4: Simulation results of pore pressure evolution and onset of failure – Complex and realistic Case B. a, e) Plots are provided for slightly supra-hydrostatic, $\lambda_v = 0.45$ (a), and sub-lithostatic, $\lambda_v = 0.85$ (e) initial pore pressure conditions in the damage zone reservoir, compared to initially hydrostatic ones ($\lambda_v = 0.4$) in the fault core. b, f) Pore pressure evolution from initial conditions to the time at which ductile (b) and brittle (f) failure initiates in the outer fault core (OFC). White arrows indicate the extent of ductile and brittle deformation**

**front in the OFC. c, g) Pore pressure conditions when fault failure initiates in a patch ($L_F$), along the main principal slip zone (PSZ) in the inner fault core (IFC). d, h) Pore pressure conditions at the time an earthquake nucleates, when the size of the failure patch, $L_F$, matches that of the theoretical predicted nucleation length, $L_N$. i - j) Mohr failure analysis for the PSZ (i) and OFC (j) at initial conditions (a, e), onset of ductile (b) and brittle (f) failure in the OFC, onset of fault failure (c, g) and earthquake nucleation (d, h). Results are presented for 40 m of 1 km simulated region shown vertically, and 2.5 m fault core exaggerated horizontally. During simulations a millimetre scale horizontal spatial grid was used, and vertically an initially millimetre scale logarithmic grid was used.**

### 4.3.1 Pore pressure evolution and onset of failure

#### 4.3.1.1 Simple Case A scenario

During the interseismic period, permeability evolves with pore pressure variations and lithological factors in the OFC, while it is constant (but anisotropic) along the fault-parallel and -orthogonal direction in both the IFC and PSZ (Fig. 3.2). The pore pressure conditions in the fault zone are represented by the pore fluid factor $\lambda_v$, defined as the ratio between pore pressure and lithostatic load. We model fluid flow for two pore pressure regimes in the damage zone reservoir, with slightly supra-hydrostatic ($\lambda_v = 0.45$) and sub-lithostatic ($\lambda_v = 0.85$) initial pore pressure conditions, compared to initially hydrostatic ones ($\lambda_v = 0.4$) in the fault core (Fig. 4.3a, d).

Our results show that at the beginning of the interseismic period, soon after an earthquake event, pore pressure excess is concentrated in the vicinity of the overpressure contact at the DZ/OFC boundary (Fig. 4.3a, d). High time resolution simulations show that pore fluids start to rapidly diffuse within the OFC first and

then into the IFC and PSZ, where pore pressure increases along the fault-parallel and -perpendicular direction. A quasi-steady state pore pressure regime is attained in the fault zone on the order of days.

Failure by sliding along the PSZ will start at 356 and 119 years, for $\lambda_v = 0.45$ and 0.85, respectively, when the shear stress level, which is controlled by the tectonic loading rate, matches the fault strength, which is dependent on pore fluid pressure (Fig. 4.3g). At this time, failure patches begin to develop along the PSZ in the supra-hydrostatic and sub-lithostatic pressure cases, respectively ($L_F$ in Fig 4.3b, e).

#### 4.3.1.2 Complex and more realistic Case B scenario

We now consider the more complex and realistic scenario where permeability evolution during the interseismic period in the fault core is additionally controlled by deformation, via brittle or ductile failure in the OFC (Fig. 4.3c-d). We consider here the same two scenarios as before, for slightly supra-hydrostatic ($\lambda_v = 0.45$) and sub-lithostatic ($\lambda_v = 0.85$) initial pore pressure condition in the damage zone reservoir, again compared to initial hydrostatic ones ($\lambda_v = 0.4$) in the fault core (Fig. 4.4a, e).

Let us consider first the case of slightly supra-hydrostatic ($\lambda_v = 0.45$) initial pore pressure conditions in the damage zone reservoir and assume the same initial state of stress in the fault and extensional tectonic loading rate as in the previous scenario (Fig. 4.4i-j). Similarly, to the case with no deformation control, high time resolution simulations show that pore fluids start to rapidly diffuse within the OFC first and, then, into the IFC and PSZ. After 327 years, the stress level in the OFC is such that ductile failure sweeps rapidly across its full width (Fig. 4.4b), before sliding begins along the PSZ (Fig. 4.4i-j). Sliding along the PSZ occurs earlier than in Case A (Fig. 4.3b, 4.4c), at almost the same time as ductile failure in the OFC.

For sub-lithostatic ($\lambda_v = 0.85$) initial pore pressure conditions in the damage zone reservoir, pore fluids start to rapidly diffuse within the OFC, IFC and PSZ, and pore pressure reaches a quasi-steady state after 19 days, but with the attainment of higher pore pressure values than in Case A. This means that at the initial stage, the OFC Mohr circle is more translated to the left than in Case A and, hence, will intercept the brittle segment of the OFC failure envelope during loading (Fig. 4.4i-j). After 58 years, brittle failure begins in the OFC increasing its permeability by around 3 orders of magnitude (Fig. 4.4f). Sliding along the PSZ first occurs at 58.7 years ($L_F$ in Fig. 4.4g), while the brittle failure front extends away from the overpressure contact, towards the IFC. There is no ductile failure in this case due to the lower level of effective stress in the OFC (Fig. 4.4i-j). The onset of sliding along the PSZ occurs after about 59 years, earlier than in Case A when there is no-deformation in the OFC (Compare Fig. 4.3e, 4.4g).

### 4.3.2 Pore pressure evolution and earthquake nucleation

During the nucleation stage of an earthquake, stable sliding spreads out from an initial small fault patch ($L_F$) until it reaches a critical size, the nucleation length ($L_N$), at which unstable fast sliding begins causing the propagation of the rupture (Marone, 1998; Scholz, 1998). In the framework of rate and state friction theory, the critical patch size or nucleation length is inversely proportional to effective normal stress (Campillo et al., 2001; Scholz, 1998) and can defined as in Eq. 3.13.

Hence, modelling results of pore pressure evolution can be used to investigate the evolution in space and time of the nucleation length, during the nucleation stage.

During our simulations, we assume velocity weakening behaviour for the PSZ (***F*** in Table 1 and Eq. 4), which has been observed for anhydrite and dolomite-rich

gouge at high temperature and sub-seismic sliding velocity (Scuderi et al., 2013). When shear stress exceeds the fault shear strength, for a given pore pressure, sliding begins along the PSZ. This condition coincides with the beginning of the nucleation phase, which ends when the size of the sliding patch on the PSZ equals that of the nucleation length (i.e. $L_F = L_N$ in Figs. 4.3c, f and 4.4d, h); a condition leading to the dynamic fast propagation of the rupture. Hence, the computed nucleation length values can be used to estimate the duration of the nucleation stage.

For the simple Case A scenario, with no deformation control on permeability, our results show that the initial pore pressure within the damage zone reservoir controls the time at which sliding initiates along the PSZ (Fig. 4.3b, e). In fact, the nucleation phase initiates significantly earlier for sub-lithostatic ($\lambda_v = 0.85$; Fig. 4.3e) initial pore pressure conditions than for slightly supra-hydrostatic ones ($\lambda_v = 0.45$; Fig. 4.3b). During the nucleation phase of the earthquake, the failure patch grows along the PSZ until conditions for dynamic seismic rupture propagation are attained (e.g. $L_F = L_N$ in Fig. 4.3c, f).

Remarkably, the nucleation phase is one order of magnitude longer in the case of initial sub-lithostatic pore pressure conditions (10.1 years, Fig. 4.3b-c), than in the case of supra-hydrostatic ones (0.4 years, Fig. 4.3e-f). These results are due to the trade-off of two competing effects: the reduction of effective normal stress due to high pore pressures and the growth of the failure patch along the PSZ. The higher the pore pressure, the lower the effective normal stress so the sooner sliding can begin (Fig. 4.3b, e). However, the lower the effective normal stress, the higher the nucleation length so a larger sliding patch is needed for earthquake nucleation, resulting in a longer nucleation phase (Fig. 4.3c, f). It is worth noting that, during the nucleation phase, pore fluid pressure conditions do not vary from the steady state

conditions attained early during the interseismic period (Fig. 4.3b-c, 4.3e-f). Hence, it is the nucleation length inverse dependence on pore pressure that controls the duration of the nucleation phase.

In Case B scenario, the occurrence of ductile ($\lambda_v = 0.45$) and brittle ($\lambda_v = 0.85$) failure before and during the nucleation phase significantly increases permeability within the OFC (Figs. 4.4b-d, 4.4f-h). The permeability enhancement caused by ductile and brittle deformation in the OFC changes the pore pressure field within the OFC and IFC. These pore pressure variations reduce the length of the interseismic period, when compared to the case with no deformation (e.g. compare Figs. 5b, e with Fig. 4.4c, g). This effect is particularly significant for the initial sub-lithostatic pore pressure regime (Fig. 4.4e), when brittle deformation in the OFC can halve the length of the interseismic period, when compared to the case with no deformation (Fig. 4.3f, 4.4h).

The length of the nucleation phase is 0.1 years in the case of initial sub-lithostatic pore pressure conditions (Fig. 4.4g-h) and much quicker, below the resolution of the modelling, in the case of supra-hydrostatic ones (Fig. 4.4c-d).

Compared to the case with no deformation control on permeability, transient fluid pressures conditions occur during the nucleation phase (Fig. 4.4c-d, 4.4g-h), as opposed to steady state conditions attained in the simple Case A with no deformation (Fig. 4.3b-c, 4.3e-f).

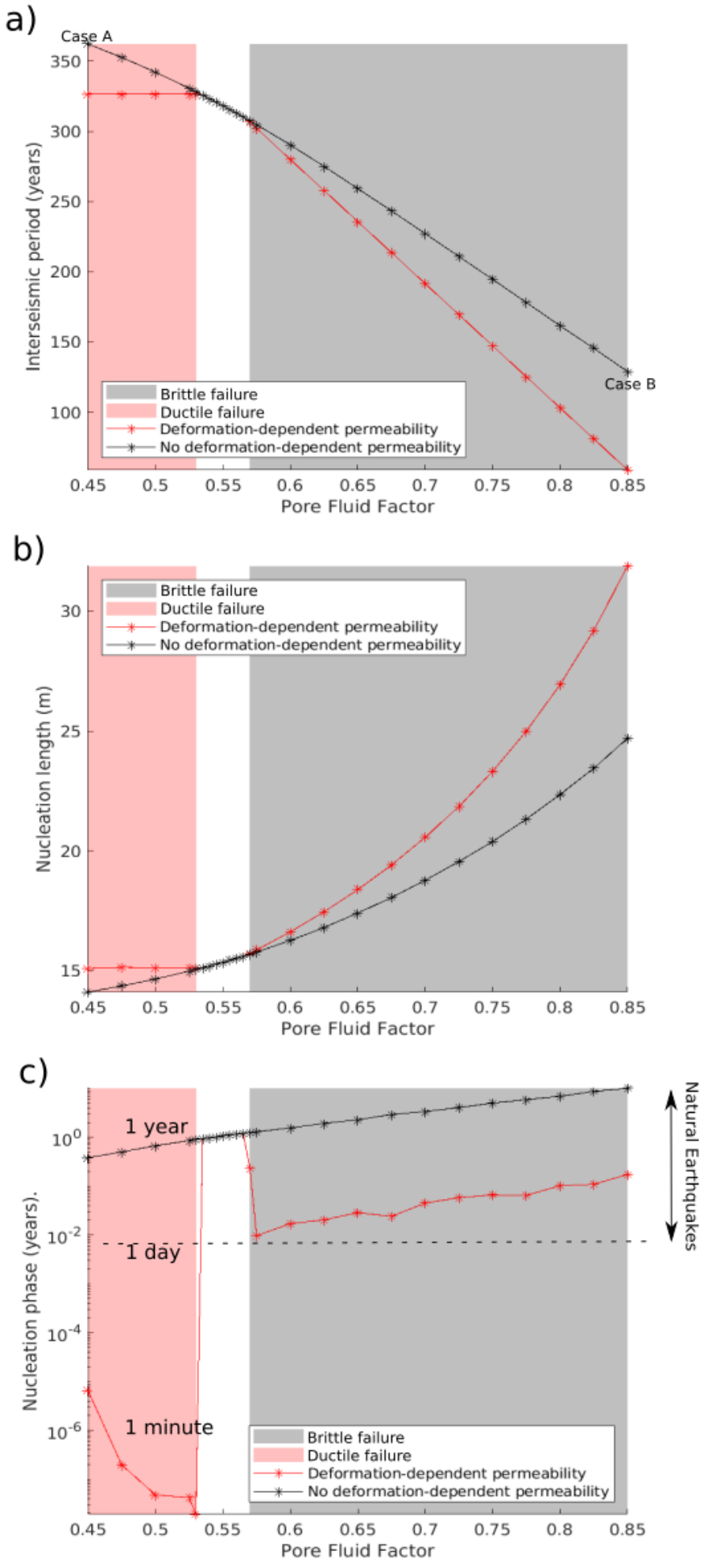

a)
Case A
Case B
Interseismic period (years)
Pore Fluid Factor
Brittle failure
Ductile failure
Deformation-dependent permeability
No deformation-dependent permeability
b)
Nucleation length (m)
Pore Fluid Factor
c)
1 year
1 day
1 minute
Nucleation phase (years).
Natural Earthquakes
Pore Fluid Factor

**Figure 4.5: Variations of earthquake parameters vs. pore fluid factor. Length of interseismic period (a), nucleation length (b) and duration of nucleation phase (c) are plotted against variation of the pore fluid factor across multiple simulations for Case A and Case B, respectively. For case B, the pressure fields of ductile and brittle deformation in the OFC are also shown in pink and grey, respectively. In panel 7c, a double-headed arrow also shows the range of lengths of the nucleation phase of some natural earthquakes, estimated by seismological observations (Fig. 4.7c; Mavrommatis et al., 2014; Kato et al., 2012; Socquet et al., 2017).**

### 4.3.3 Pore fluid factor control

We perform parameter studies of the same two case scenarios, with (Case A) and without (Case B) deformation-dependent permeability, at a range of different pore fluid factors ($0.45 < \lambda_v < 0.85$, in steps of 0.025) (Fig. 4.5a-c). Our results show that pore fluid factor and deformation-dependent permeability all exert primary control over the duration of the interseismic period and nucleation phase, and over the length of the failure patch at nucleation. Increased initial pore fluid factor in the pressurised reservoir acted in both scenarios to decrease the duration of the interseismic period (Fig. 4.5a), and to increase the nucleation length (Fig. 4.5b) and the duration of the nucleation phase (Fig. 4.5c).

The timing of brittle failure in the OFC is dependent on pore fluid factor, while the timing of ductile failure is constant due to the flat ductile region of the OFC failure envelope (Fig. 4.1a). In all cases, the inclusion of deformation-dependent permeability decreases both the interseismic period (to 60.1 years, Fig 4.5a) and the nucleation phase duration (to less than 1 year, Fig 4.5c), while increasing the size of the nucleation length (to > 30 m, Fig. 4.5b).

It is interesting to note that for pore fluid factors of 0.55 the results obtained for the two scenarios considered here converge (Figs. 4.5a-c). This is due to the specific tectonic loading rate chosen for our simulations, which causes fault sliding in the PSZ before any fracturing by brittle or ductile failure can occur within the OFC. This observation shows that tectonic loading rate also plays a role in controlling pore pressure diffusion in fault zones during the seismic cycle.

## 4.4. Discussion and Conclusions

### 4.4.1 Mode of failure controls pore pressure diffusion and earthquake recurrence interval

Small scale fracturing within fault zones acts as a primary control on pore pressure diffusion during the interseismic period (e.g. natural earthquakes) or when a stress perturbation is caused in a reservoir (e.g. induced seismicity). In general, fracturing can increase the average permeability of rocks in the zones of damage adjacent to the main slip zones by several orders of magnitude, and therefore driving pore pressure diffusion more effectively. In the specific case investigated here, at a given tectonic fault-loading rate, failure occurring within the outer fault core (OFC) reduces the frictional strength of the fault at a faster rate, and fault sliding (e.g. beginning of the nucleation phase) can initiate earlier than in the case where no fracturing occurs in the OFC (Fig. 4.5a).

More specifically, results from our case study show that initial high pore pressures can cause small scale brittle failure in the OFC during the interseismic period, which creates higher permeability than ductile failure occurring at lower initial pore pressures (e.g. Fig. 4.4b-c and 4.4f-g). Hence, in the case of brittle failure in the OFC, fault sliding occurs much earlier than in the case of failure by ductile deformation, reducing the duration of the interseismic period (Fig. 4.5a).

These results have implications in terms of seismic hazard estimates, as they show that local factors such as lithology and fault zone structure can significantly affect the length of the interseismic period and, hence, the recurrence interval of earthquakes. They also show that, during the coseismic and interseismic period, the evolution of the hydrogeological conditions of the fault zone and its connected reservoirs will affect the recurrence interval of events. In particular, hydrogeological monitoring of springs (e.g., Barberio et al., 2017) and boreholes in the epicentral area or in the surrounding areas of injection sites could potentially provide information on coseismic fluid discharge and on interseismic fluid recharge between the fault zone and the connected aquifers. These observations could be used to estimate the pore pressure evolution of the fault zone and its surroundings during the seismic cycle.

### 4.4.2 Implications for fluid induced earthquake nucleation

Our results have some implications for our understanding of the role and controls of aseismic slip during the nucleation phase preceding an earthquake. Although aseismic slip episodes have been relatively commonly observed over the last decade, until very recently the occurrence of aseismic slip as a precursor to major earthquakes was almost completely unknown. Large aseismic slip episodes have now been identified immediately preceding the recent $M_w$ 9, 2011 Tohoku earthquake (Kato et al., 2012; Mavrommatis et al., 2014) and the $M_w$ 8.1, 2014 Iquique earthquake (Ruiz et al., 2017; Socquet et al., 2017). It is argued that these aseismic slip events, lasting a few months, contribute to the triggering of earthquakes and are related to their preparatory nucleation phase (Guglielmi et al., 2015).

Overall, our results show that both the inclusion of realistic models of fault zone architecture and deformation-dependent permeability (brittle and ductile failure)

control the size of the sliding patch (Fig. 4.7b) during earthquake nucleation and the duration of the nucleation phase (Fig. 4.7c).

The size of the failure patch during the nucleation phase is always larger when realistic models of fault zone architecture and deformation-dependent permeability are considered (Fig. 4.7b). In particular, small scale fracturing by brittle failure, occurring for initially high pore pressures, provides the largest slipping patches (> 30 m in Fig. 4.7b). These results are of particular relevance when considering that technological improvements in signal/noise ratio and spatio-temporal resolution of geodetic data are lowering the detection thresholds for measurements of aseismic slip. In particular, the advent of new satellite radar missions now enables a systematic, global investigation of pre-seismic slip for the first time.

Our results show that the duration of the nucleation phase is significantly reduced from a few years to a few months at high values of initial pore pressure, when realistic models of fault zone architecture and deformation-dependent permeability are considered. Interestingly, a few months is also the time scale of aseismic slip measured during the nucleation phase of some recent large earthquakes (Fig. 4.7c; $M_w$ 9, 2011 Tohoku and $M_w$ 8.1, 2014 Iquique earthquakes).To conclude, estimates of the duration of the nucleation phase have implications for earthquake early warning systems. In fact, intermittent aseismic creep on fault patches > 30 m in diameter, over a period of few months, could be detectable well in advance of a significant seismic event, perhaps using geodetic data and new satellite remote sensing techniques.

Table 4.1: Physical constants and variables used in simulations

| Symbol | Definition | Value | Reference |
|---|---|---|---|
| $\beta$ | Compressibility | $10^{-10}$ $Pa^{-1}$ | (Burke, 2011) |
| $\eta$ | Viscosity | $10^{11}$ Pa s | (Burke, 2011) |
| $\phi$ | Porosity | 1% | (De Paola et al., 2009) |
| $\mu_s$ | Static friction of the fault plane (Primary slip zone) | 0.6 | (Byerlee, 1978) |
| G | Shear modulus | 45.7 Gpa | (Anderson and Ivsik, 1995) |
| $L_c$ | Critical slip distance | 63 µm | (Scuderi et al., 2013) |
| F | Difference of the rate and state parameters | 0.005 | (Scuderi et al., 2013) |
| $\xi$ | Failure patch geometry factor | 1 | - |
| $L_{OFC}$ | Outer fault core width | 2 m | - |
| $L_{IFC}$ | Inner fault core width | 1 m | - |
| $L_{PSZ}$ | Primary slip zone width | 2 mm | - |
| $C_{fault}$ | Fault cohesion (Primary slip zone) | 0 | - |
| $\mu_{OFC-static}$ | Internal friction of the OFC (Static) | 0.704 | Fitted with data from (De Paola et al., 2009) |
| $C_{OFC-static}$ | Cohesion of the OFC (Static) | 15.5 MPa | Fitted with data from (De Paola et al., 2009) |
| $\mu_{OFC-ductile}$ | Internal friction of the OFC (ductile) | 0 | Fitted with data from (De Paola et al., 2009) |
| $C_{OFC-ductile}$ | Cohesion of the OFC (ductile) | 38.14 MPa | Fitted with data from (De Paola et al., 2009) |
| $k_{OFC-prefailure-horizontal}$ | Unstressed prefailure permeability of the OFC | $8\times10^{-21}$ $m^2$ | (De Paola et al., 2009) |
| $k_{OFC-brittle-horizontal}$ | Unstressed permeability of the OFC after brittle failure | $1.1287\times10^{-18}$ $m^2$ | Fitted with data from (De Paola et al., 2009) |
| $k_{OFC-ductile-horizontal}$ | Unstressed permeability of the OFC after ductile failure | $2.497\times10^{-18}$ $m^2$ | Fitted with data from (De Paola et al., 2009) |
| $k_{IFC-horizontal}$ | IFC permeability | $10^{-18}$ $m^2$ | (Evans et al., 1997) |
| $k_{PSZ-horizontal}$ | PSZ permeability | $10^{-21}$ $m^2$ | (Wibberley and Shimamoto, 2002) |
| $\gamma_{OFC-prefailure-horizontal}$ | OFC prefailure pressure sensitivity | $-0.04$ $Mpa^{-1}$ | De Paola et al., 2009) |
| $\gamma_{OFC-brittle-horizontal}$ | OFC pressure sensitivity after brittle failure | 0 | -0.1136 $Mpa^{-1}$ Fitted with data from (De Paola et al., 2009) |
| $\gamma_{OFC-ductile-horizontal}$ | OFC pressure sensitivity after ductile failure | $-0.1136$ $Mpa^{-1}$ | Fitted with data from (De Paola et al., 2009) |
| $k_{OFC-prefailure-vertical}$ | Unstressed prefailure permeability of the OFC | $3.5\times10^{-20}$ $m^2$ | (De Paola et al., 2009) |
| $k_{OFC-brittle-vertical}$ | Unstressed permeability of the OFC after brittle failure | $1.394\times10^{-17}$ $m^2$ | Fitted with data from (De Paola et al., 2009) |
| $k_{OFC-ductile-vertical}$ | Unstressed permeability of the OFC after ductile failure | $3.681\times10^{-17}$ $m^2$ | Fitted with data from (De Paola et al., 2009) |
| $k_{IFC-vertical}$ | IFC permeability | $10^{-17}$ $m^2$ | (Evans et al., 1997) |
| $k_{PSZ-vertical}$ | PSZ permeability | $10^{-20}$ $m^2$ | (Wibberley and Shimamoto, 2002) |
| $\gamma_{OFC-prefailure-vertical}$ | OFC prefailure pressure sensitivity | $-0.133$ $Mpa^{-1}$ | (De Paola et al., 2009) |
| $\gamma_{OFC-brittle-vertical}$ | OFC pressure sensitivity after brittle failure | 0 | -0.07968 $Mpa^{-1}$ Fitted with data from (De Paola et al., 2009) |
| $\gamma_{OFC-ductile-vertical}$ | OFC pressure sensitivity after ductile failure | $-0.07968$ $Mpa^{-1}$ | Fitted with data from (De Paola et al., 2009) |
| $\dot{\tau}_R$ | Tectonic loading rate | 0.15 MPa $yr^{-1}$ | - |
| t | Time | - | - |
| p | Pressure | - | - |
| $p_e$ | Effective pressure | - | - |
| k | Permeability tensor | - | - |
| $\sigma_n$ | Normal stress | - | - |
| $\sigma_n'$ | Effective normal stress | - | - |
| $\sigma_1$ | Principal maximum stress | - | - |
| $\sigma_3$ | Principal minimum stress | - | - |
| $\tau$ | Shear stress | - | - |
| $\tau_f$ | Shear strength | - | - |
| $L_N$ | Nucleation length | - | - |

**OFC = Outer Fault Core**
**IFC = Inner Fault Core**
**PSZ = Primary Slip Zone**

NB: The values for compressibility and viscosity given in Burke, 2011, and subsequently, Table 3.1 are lower than other values produced from empirical relationships. With viscosity being higher by a factor of ten and compressibility by a factor of five (Mathias et al., 2009). These variables decrease the instantaneous rate of pore pressure change (Eq. 3.4) impact short term transient flow much more heavily than longer-term quasi-steady-state changes. Simulations were run to validate the impact of this variation on the results of Chapters 4 and 5. There were no changes to rupture patch dimensions and the length of the interseismic period, both

controlled by longer-term changes. The length of the nucleation phase increase by roughly an order of magnitude for short nucleation phases (seconds to minutes), for longer nucleation phases the value of nucleation phase is not significantly changed. Transient fluid flow triggered by discontinuous failure explains why this impact disproportionately affects shorter nucleation lengths.

# CHAPTER 5

*5. Fault zone architecture and dimensions control the evolution of the pore pressure field during the seismic cycle*

# 5. Fault zone architecture and dimensions control the evolution of the pore pressure field during the seismic cycle


**Abstract**

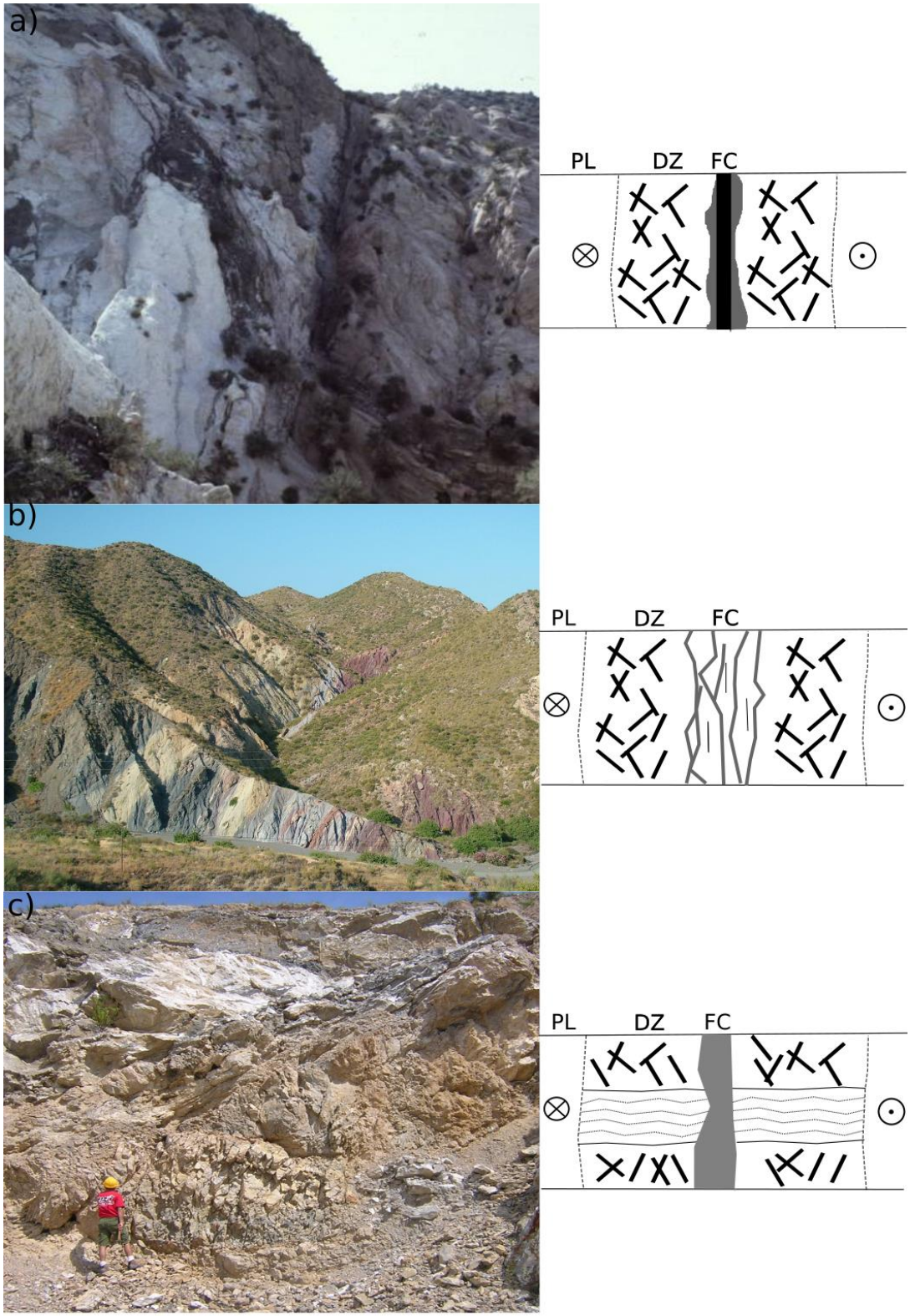
Natural subsurface fluid flow can perturb fault zone pore pressure environments, and in turn can drive large variations in fault frictional strength potentially triggering seismicity. The earthquake nucleation phase spans the period of stable sliding from initial infinitesimal development of the patch to critical nucleation length, at which fast, unstable sliding (an earthquake) begins rupture (Marone, 1998; Scholz, 1998). Previous numerical simulation studies have examined the role of fluid flow in faulting processes and associated seismicity, no simulations have examined the earthquake nucleation phase sensitivity to variation in the scale of fault zone architecture and neighbouring lithology.

Here, we perform parameter studies on pore pressure diffusion and earthquake nucleation, with realistic models of ductile failure, varying the dimension of components of fault zone architecture and neighbouring lithology, outer fault core width and the height of pressurised layers abutting the fault core. As a base case for the studies, we simulate the evolution of overpressured, supercritical $CO_2$ in natural, exhumed faults in evaporite sequences. The failure and transport properties of rocks are derived from laboratory measurements, and realistic models of fault zone architecture.

The results obtained show that for a given tectonic loading rate, a thinner fault core will result in a more effective fault weakening, as the fault frictional strength will reduce at a faster rate due to higher pore pressures. The impact of fluid flow on the fault being more significant for faults with a thinner rather than thicker

outer fault core. In the absence of fluids, the base mechanical strength of the slipping portion of the fault did not vary with thickness. Similarly, an increasing intersecting overpressured aquifer thickness in the damage zone produces a higher magnitude of pore pressure in the fault core, which weakens the principal slip zone located in the centre of the fault core.

Understanding the controls exerted on the duration of the nucleation phase of earthquakes has important implications for premonitory signal detection, as identifying extended nucleation phases of active faults would increase the likelihood of detection of early seismicity warnings. Our case study shows that, for a given fault, characteristic values of fault zone parameters (e.g. fault core width and intersecting overpressured aquifer thickness) govern the transition from relatively long – on the order of days to months – easily detectable nucleation phase to very short ones – on the order of seconds to minutes – difficult to detect. As such, realistic estimates of uncertainty in fault zone architecture dimension must inform hazard estimates, as small differences in scale can correspond to significant variation of the nucleation phase, from seconds to years.




DZ = Damage Zone
FC = Fault Core
PL = Protolith

**Figure 5.1: Examples of differing fault zone structures with simplified schematic diagrams. (a) The Punchbowl fault, San Andreas system, California. A fault with 40km of displacement, a 50cm thick ultracataclasite layer with 1mm thick primary slip surfaces and a damage zone extending 15m. This fault and faults with a similar ratio of damage zone to fault core act as distributed conduits with respect to fluid flow. (Chester and Chester, 1998; Chester and Logan, 1986) (b) The Carboneras fault, Spain. A fault with 40km of displacement, a 1km thick fault core of fault gouge bounding fractured lens and included blocks and a 100m thick damage zone. This fault and faults with a similar ratio of fault core to damage zone would act as a combined conduit/barrier(Faulkner et al., 2003). (c) The Roccastrada fault, Italy. A fault with 1km of displacement a thick cataclasite layer containing primary slip surfaces around 1mm thick, a well-developed damage zone in dolostone layers and an absent damage zone in the anhydrite layers. So far, no study has considered the fluid flow properties of this fault zone structure.**

## 5.1. Introduction

Natural (Di Luccio et al., 2010; Miller et al., 2004; Nur and Booker, 1972) and human induced (Ellsworth, 2013; McGarr et al., 2015; Sumy et al., 2014) pore pressure variations in the upper crust can cause fluid migration and trigger seismicity.

An increase in pore fluid pressure can weaken faults, by decreasing their frictional strength (Cox, 2010; Sibson, 1990, 1992), and make the interseismic period shorter by bringing pressurized fault patches closer to failure. At the same time, an increase in pore pressure can increase the duration of the nucleation phase of an earthquake, due to the inverse proportionality of the nucleation length fault

parameter to the effective normal stress (Marone, 1998; Scholz, 1998). The nucleation phase concept implies that slow, aseismic fault sliding precedes an earthquake. Then, sliding spreads out with an accelerating velocity until the sliding patch reaches a critical size – the nucleation length – at which fast seismic sliding will propagate (Marone, 1998; Scholz, 1998). Usually aseismic sliding initiates at small fault asperities, due to the local concentration of high shear stress or elevated pore fluid pressure reducing fault strength.

There is a relation between the evolution of the pore pressure field during the seismic cycle and fluid-induced seismicity (Collettini et al., 2009; Cox, 1995; De Paola et al., 2008; Hickman et al., 1995; Miller, 1996; Rice, 1992; Sibson, 2000). In particular, the integration of field observations and experimental datasets show that the transport properties of faults are closely related to their internal structure (Caine et al., 1996; Collettini et al., 2009; Lockner and Beeler, 1999; Seront et al., 1998). Nevertheless, the control exerted by fault zone architecture and lithological variations in the neighbouring zones on pore pressure evolution during the seismic cycle is still poorly investigated and understood. This is mainly due to the complexity and great variability shown by seismic fault zone architectures and to significant uncertainty in the estimation of the transport properties and size of their associated fault zone domains (Caine et al., 1996; Cox, 1995; De Paola et al., 2009; Fischer, 1992; Hangx et al., 2010; Mitchell and Faulkner, 2008; Morrow and Lockner, 1997, 1994; Paterson and Wong, 2005; Peach and Spiers, 1996; Wong et al., 1997; Zhu et al., 1997; Zoback and Byerlee, 1975).

Simulations of faulting and fluid flow have previously been conducted using coupled deformation and fluid flow modelling software (e.g. TOUGH-FLAC) (Cappa and Rutqvist, 2011a, 2012; Rutqvist et al., 2015, 2013a, 2002). However,

these studies consider simplistic fault zone architectures (e.g. Fig. 5.1a-b; Cappa, 2009; Cappa and Rutqvist, 2011; Hsiung et al., 2005; Leclère et al., 2015; Mazzoldi et al., 2012; Rinaldi et al., 2014; Rutqvist et al., 2013b, 2009), where permeability is a continuous function in porosity, volumetric strain, average effective stress (Davies et al., 2001) and in some cases shear strain on the fault (Rutqvist et al., 2007).

Here, we model fluid flow in exhumed faults in evaporite sequences with complex architecture, and pore pressure- and deformation-dependent permeability. These faults represent an analogue of the seismic sources at hypocentre depth of recent seismic events in the Northern Apennines of Italy (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.5 2016 Norcia earthquakes). Our results are then generalised to relate fault zone parameters (e.g. dimension, architecture and associated deformation mechanisms, lithological variations in the neighbouring rocks to the fault) to the duration of the interseismic period and the nucleation phase of an earthquake.

Low resolution of indirect measurement methods and generic inference from specific natural analogues make fault zone parameters poorly constrained. Hence, numerical simulations and multiphysics models of seismic faults are a useful tool to predict the distribution and evolution of pore fluid pressure during the seismic cycle. Predictions of the pore pressure field during the seismic cycle have important implications for earthquake forecasting, as they potentially allow the estimation of the length of the interseismic period. Furthermore, they have also implications for earthquake early warning systems, as they potentially allow estimating the duration of the aseismic nucleation phase preceding an earthquake, and the likelihood of detecting early premonitory signal.

### 5.2. Fault zone architecture controls fluid migration

Tectonic faults are zones of finite width, whose internal architecture can be described by three main fault zone domains (Chester et al., 1993; Faulkner et al., 2010) : the protolith, the damage zone and the fault core. The fault core is the domain where most of the fault slip is accommodated within narrow slip zones. Intense damage, but relatively little amount of slip, is accommodated within the damage zone domains often present on both sides of the fault core. In the damage zone deformation is mostly accommodated by network of connected fractures and veins. Finally, the protolith is the original source rock of those fault rocks found in the damage zone and fault core. There is no damage or faulting in the protolith related to the specific fault zone activity.

Caine et al. (1996) proposed that whether a fault acts as a conduit or a barrier to fluid migration will depend on the relative thickness between the fault core domain, dominated by low permeability multiple narrow slip zones and fine-grained fault rocks, and the surrounding damage zones, dominated by distributed and well-connected fracture patterns. Faults with a thin, low permeability fault core and a thick and well developed damage zone may act as conduits, favouring fluid migration through the damage zone fault rocks (Fig. 5.1a, e.g. Punchbowl fault with a 50 mm thick fault core and 15 m wide damage zone; Chester and Chester, 1998; Chester and Logan, 1986). Conversely, faults with a thick, low permeability fault core and a relatively thin, poorly developed damage zone may act as barriers to migrating fluids (Fig. 5.1b, e.g. Carboneras fault with 1 km thick fault core and less than a few tens of metres damage zone; Faulkner et al., 2003; Faulkner and Rutter, 2001). In between these end-members there are faults with a variety of architectures and range of fault core damage zone aspect ratios (Faulkner et al., 2010). Amongst

these, exhumed seismic faults with complex fault core structure observed in evaporite sequences can be attributed with a mixed conduit/barrier behaviour, controlled by transient pore pressure evolution and mode of failure (Fig. 5.1c e.g. Roccastrada fault; De Paola et al., 2008; Collettini et al., 2009).

The Roccastrada fault analogue from the Northern Apennines of Italy is chosen as a base case for fluid flow simulations. There, borehole measurements at 4 - 4.8 km in the Triassic evaporites encountered overpressured $CO_2$, which has been implicated in historical and recent seismicity in that area (Chiodini and Cioni, 1989; Collettini and Barchi, 2002; Miller et al., 2004).

We apply our simulations to the observed fault zone architecture, deformation patterns and protolith stratigraphic setting (De Paola et al., 2008; Collettini et la., 2009), where permeability is assumed to vary as a function of effective stress and ductile failure. To generalise our findings, the dimensions of fault zone domains and protolith stratigraphy are varied systematically from this base case, during parameter study simulations.

### 5.3. Numerical Method

We perform parameter studies composed of sets of simulations of nonlinear diffusion in fault zones with realistic, complex fault zone architecture. A well-studied and constrained base case for these studies is taken from an exhumed normal fault in evaporite sequences (Fig. 5.1c), which is an analogue of the seismic source in the hypocentre zone of the Northern Apennines seismic belt (e.g. Mirabella et al., 2008). We apply our simulations to the observed fault zone architecture, deformation patterns and protolith stratigraphic setting (De Paola et al., 2008; Collettini et al., 2009), where permeability is assumed to vary as a function of effective stress and ductile failure. To generalise our findings, the dimensions of fault zone domains and

protolith stratigraphy are varied systematically from this base case, during parameter study simulations.

### 5.3.1. Porous media flow and numerical solution

In simulations, the evaporites are treated as classically porous media, using Eq. 3.4 a relation between pore pressure $p$ and permeability $k$ to the rate of change of pressure with time $t$. We approximate the compressibility $\beta$ as being only the fluid compressibility, because the compressibility of supercritical $CO_2$ exceeds that of evaporite rocks by several orders of magnitude (Burke, 2011; Robertson et al., 1958). The variation of viscosity $\upsilon$ and compressibility with effective stress can be neglected for the conditions considered (Burke, 2011), where the effective stress is defined as the difference of principal minimum stress and pore pressure Eq. 2.1. Taking experimental relationships established for low porosity rocks in De Paola et al. (2009) and (Hangx et al., 2010), we express permeability as a function of effective stress both prefailure and after ductile deformation, Eq. 3.6 accommodated by microscale, distributed fracturing (Detournay and Cheng, 1993).

Large deep-seated $CO_2$ fluxes greater than 0.45 t day$^{-1}$ km$^{-2}$ have been measured in the epicentre areas of the Northern Apennines seismic belt (e.g. $M_w$ 6.0 1997-98 Colfiorito and $M_w$ 6.3 2009 L'Aquila extensional earthquakes) (Chiodini et al., 2004; Collettini et al., 2008). As such, it is reasonable to assume in simulations that the set of theoretical faults considered in the parameter study are saturated with a single phase of supercritical $CO_2$.

In the simulations presented here fault zone geometries, physical properties and mechanical behaviour, taken as a base case and re-used in part for parameter studies, are inferred and constrained from main seismic extensional fault zone in evaporite rocks (Collettini et al., 2009; De Paola et al., 2009, 2008). However, the

techniques and results are more generally applicable to any natural fault zone with partially constrained fault zone architecture and constrained physical and mechanical properties.

### 5.3.2. Model input parameters

#### *5.3.2.1. Fault zone architecture and base model setup*

The fault zones constraining the base simulation case occur within the Triassic Evaporite formation, which is a 1.5-2 km thick sequence of 6 – 19 m thick interbedded layers of anhydrite/gypsum and dolostones (Barchi, 2002; Trippetta et al., 2013). Moderately dipping (45°) normal faults are present at hypocentre depths in the Northern Apennines seismic belt, thought to favour fluid overpressure attainment and fault reactivation (Barchi, 2002; Miller et al., 2004; Mirabella et al., 2008).

For our base case simulation, we adopt a model of fault zone architecture proposed from field observations, which represent an analogue of those present at seismogenic depths (Fig. 5.1a; Collettini et al., 2009; De Paola et al., 2008). Modelled fault zone architecture comprises a complex fault core, made of a 1 m wide inner fault core (IFC), where most of the slip is localised within narrow principal slip zones, and a 2 m wide outer fault core (OFC) (Fig. 2.8). The distribution of damage in the damage zone is heterogeneous and controlled by the lithology of the protolith. In fact, when the protolith rocks are made of dolostones, a well-developed damage zone (DZ) of pervasively fractured dolostones extends at least 10 m in either direction from the fault core. On the other hand, the foliated anhydrite layers interbedded between the fractured dolostones, outside the fault core in the damage zone, contain no macroscopic fracturing, and hence no damage zone is

present here (Fig. 2.8) (De Paola et al., 2008). These fractured dolostone layers in the damage zone are pressurised with supercritical carbon dioxide at depth. We refer to the contact between this pressurised dolostone layer in the DZ and the OFC as the overpressure contact (OC).

A schematic, yet realistic, fault zone architecture is used within base case simulations. This fault zone architecture is formed of an IFC, containing fine-grained cohesive cataclasites, set within an OFC extending 2 m on both directions, containing cohesive foliated anhydrites (Figs. 2.8, 3.2) (De Paola et al., 2008). The IFC is made of fine-grained cohesive cataclasites, with an isotropic texture (De Paola et al., 2008). The most prominent structural feature in the IFC are straight and thin, millimetre scale, principal slip zone (PSZ), made of ultra-fine grained and dolomite-rich gouges (Fig. 2.8) (De Paola et al., 2008). Schematically we represent these as a single PSZ of zero thickness, made of incohesive fault gouges, in the centre of the IFC, which is assumed as made of cohesive cataclasite (Fig. 3.2).

During the seismic cycle, it is assumed that the modelled fault will behave according to the fault-valve behaviour described by Sibson (1990), when pore fluid overpressure is released upon seismic sliding. At the beginning of the seismic cycle, following a main seismic events, we then take pore pressure within the fault core to be uniform and hydrostatic (Miller et al., 2004). We model each of our set of theoretical faults at a hypocentre depth of 7 km, and subject to an extensional fault unloading rate of 0.15 MPa/year, based on the tectonic setting of our base case (Chiaraluce et al., 2003).

The fractured dolostone layer within the DZ represent our pore fluid reservoir due to the presence of well-connected and dense fracture patterns (Fig. 3.2). Pore fluid overpressure can be generated within the fracture dolostones of the DZ, due to

their high permeability compared to the low permeability of interbedded anhydrites, which may act as seal (Fig. 3.2). Pore pressures are taken to be hydrostatic in the anhydrites seal. The OFC and IFC are directly recreated in simulations where the DZ inform boundary conditions. Further, to reduce computational costs, we exploit the model's symmetry properties. The idealised fault section has two planes of symmetry, the fault parallel plane bisecting the fault, and the fault perpendicular plane bisecting the overpressure contacts. The gradient of pore pressure normal to theses symmetry boundaries is taken to be zero. We simulate an area of 2.5 by 1000 m, representing the upper left quadrant of the fault core (Fig. 3.2), subject to an extensional tectonic loading rate of 0.15MPa/year. The non-symmetry external boundaries are taken to be hydrostatic, except for the region of contact between the overpressured, fractured dolostone layers in the DZ and the fault core (Fig. 3.2; Trippetta et al., 2013). The top, and bottom boundary by symmetry, is taken to be sufficiently removed from the overpressure source that this boundary does not significantly alter fluid flow in the fault core. The length of the simulated area is selected ensuring that the pressure gradient at the upper perpendicular boundary is less than 1% of hydrostatic pressure per metre. All models are run from an initial stress state with minimum principal stress set at 85% of lithostatic load (Miller et al., 2004).

Fault frictional strength is assumed to be homogeneous, and without asperities, thus any variation in frictional strength is dependent only on effective normal stress. Effective normal stress on a sliding fault patch is held constant after sliding begins as the fault patch can no longer accumulate or dissipate stress energy locally, any change in energy acts instead to accelerate or decelerate sliding. A similar approach is taken in (Campillo et al., 2001) and using a similar

approximation to Uenishi and Rice (2003), we approximate the nucleation length ($L_N$) of a failure patch ($L_F$) as the nucleation length of the stiffest point in the patch (largest effective normal stress).

The simulations presented here consider the earthquake nucleation phase (Marone, 1998; Scholz, 1998). During this phase, stable sliding spreads out from an initial point ($L_F$) until it reaches a critical size, the nucleation length ($L_N$), at which unstable fast sliding begins causing the propagation of the rupture (Marone, 1998; Scholz, 1998). Taking rate and state friction as a theoretical framework, the critical patch size or nucleation length is inversely proportional to effective normal stress (Campillo et al., 2001; Scholz, 1998) and can be expressed by Eq. 3.13. Hence, our modelling results of pore pressure evolution can be extended to investigate the evolution in space and time of the nucleation length, during the nucleation stage.

During our simulations, we assume velocity weakening behaviour for the PSZ (parameter $\boldsymbol{F}$ in Eq. 4), which has been observed for anhydrite and dolomite-rich gouge at high temperature and sub-seismic sliding velocity (Scuderi et al., 2013). At the point when shear stress exceeds the fault shear strength, for a given pore pressure, sliding begins along the PSZ. We define the duration of the nucleation phase as the time interval between the beginning of sliding and the time when the size of the sliding patch on the PSZ equals that of the nucleation length (i.e. $L_F = L_N$); a condition leading to the dynamic, fast propagation of the rupture. Hence, the computed nucleation length values can be used to estimate the duration of the nucleation stage.

#### *5.3.2.2. Failure envelopes, mode of failure and transport properties*

The failure envelopes, mode of failure and transport properties of the OFC and IFC subdomains of the fault core are derived from triaxial deformation experiments with fluid flow, performed on borehole samples of Triassic Evaporites rocks (De Paola et al., 2009). Fabric anisotropy controls the strength of intact anhydrite rocks, being weakest when loaded along a sub-parallel to foliation direction. Mode of failure discontinuously transitions from brittle to ductile failure at effective stresses of 20 MPa, and appears independent of grain size or the presence of fabric anisotropy and its orientation (De Paola et al., 2009).

A simplified Mohr-Coulomb failure envelope is assigned to each of the components of fault zone architecture, i.e. the OFC, the IFC and the PSZ (Fig. 4.3, Table 4.1). For the OFC specifically, the failure envelope of the foliated anhydrite is taken from triaxial loading experiments performed on anhydrite borehole samples with foliation oriented sub-parallel to the loading direction (Fig. 4.3a; Table 4.1; De Paola et al., 2009). A sharp elbow is present in failure envelopes representing the sharp transition from brittle to ductile failure at an effective stress of 20 MPa (Fig. 4.3a). The failure envelope of the IFC is given by triaxial loading experiments performed on fine-grained, homogeneous anhydrite borehole samples, representative of cohesive, anhydrite bearing fine-grained cataclasites, with no fabric (Fig. 4.3a; Table 4.1; De Paola et al., 2009). The failure envelope of the PSZ is represented with a Byerlee's friction coefficient of 0.6 treating it as a cohesionless fault plane (Fig. 4.3b; Table 4.1; Scuderi et al., 2013).

The OFC permeability tensors are specified using data from triaxial deformation experiments with fluid flow in De Paola et al. (2009) (Fig. 4.4). Measurements of deformation and fluid flow both parallel and perpendicular to

fabric represent the fault parallel and perpendicular components of OFC permeability, respectively. These triaxial deformation measurements with fluid flow show that the permeability of anhydrite rocks is controlled by two factors (De Paola et al., 2009): 1) effective stress, as permeability decreases with increasing effective stress due to porosity reduction (Fig. 4.4a-b); and 2) distributed deformation, as permeability increases with increasing loading due to the creation of fractures within the rock (Fig. 4.4c-d). Sharp increases in permeability accompany failure of these anhydrite rocks and, for a given constant pore pressure, the magnitude of this increase is governed by the brittle or ductile mode of failure (Fig. 4.4c-d).

If effective stress is held constant while ductile failure occurs, permeability will rapidly increase before reaching a plateau value, when a percolation threshold state is attained due to the development of a fully connected network of microfractures (Fig. 4.4c-d; 5.6; De Paola et al., 2009). Anhydrite rocks deforming in a ductile manner are still sensitive to effective stress variations (Fig. 4.4c-d; 5.2; Table 4.1), which may alter sample porosity by opening or closing small fractures (De Paola et al., 2009).

Efficient hydrothermal healing processes, acting during the interseismic period, may seal micro- and macro-scale fractures within a few years of a slip event (Keulen et al., 2008; Nakatani and Scholz, 2004; Niemeijer et al., 2008; Scuderi et al., 2013; Yasuhara et al., 2005). As such, we assume that any fractures created during previous seismic events are effectively healed by the beginning of the interseismic period in our simulations.

The permeability of the fine-grained cataclasites in the IFC and gouges in the PSZ are assumed to be anisotropic in the fault-parallel and fault-orthogonal direction

(Evans et al., 1997; Wibberley and Shimamoto, 2002), excepting the OFC, they are not assumed to depend on pore pressure and deformation (Fig. 5.2; Table 4.1, Eq. 3).

### 5.4. Results

A series of parameter studies were performed at a depth of 7 km, for a fixed tectonic unloading rate of 0.15 MPa/year in the minimum principal stress direction. Fluid flow in the fault core during the interseismic period and the earthquake nucleation phase are modelled for parameter studies, where outer fault core (OFC) width and overpressure contact (OC) height are varied across realistic parameter ranges (Fig. 3.2). Further high-resolution studies are conducted in the vicinity of any sharp transitions in earthquake nucleation time, and the results compiled.

The pore fluid factor $\lambda_v$, defined as the ratio between pore pressure and lithostatic load $\sigma_v$, is used to represent the pore pressure conditions in the fault zone. Parameter studies are performed at a pore fluid factor of $\lambda_v = 0.45$, which correspond to the conditions that would favour ductile failure in the OFC.

For each parameter range, two separate studies are run using both deformation-independent and –dependent permeability. In the deformation-independent case, permeability evolves during the interseismic period in the OFC solely controlled by pore pressure variations and lithological factors (e.g. fabric presence and orientation; Fig. 4.4a-b). In the case of deformation-dependent permeability, a more complex and realistic scenario is assumed, where permeability evolution during the interseismic period is additionally controlled by deformation, via brittle or ductile failure in the OFC (Fig. 4.4c-d). In both cases, permeability is assumed to be constant in both the IFC and PSZ, but anisotropic along the fault-parallel and fault-orthogonal directions.

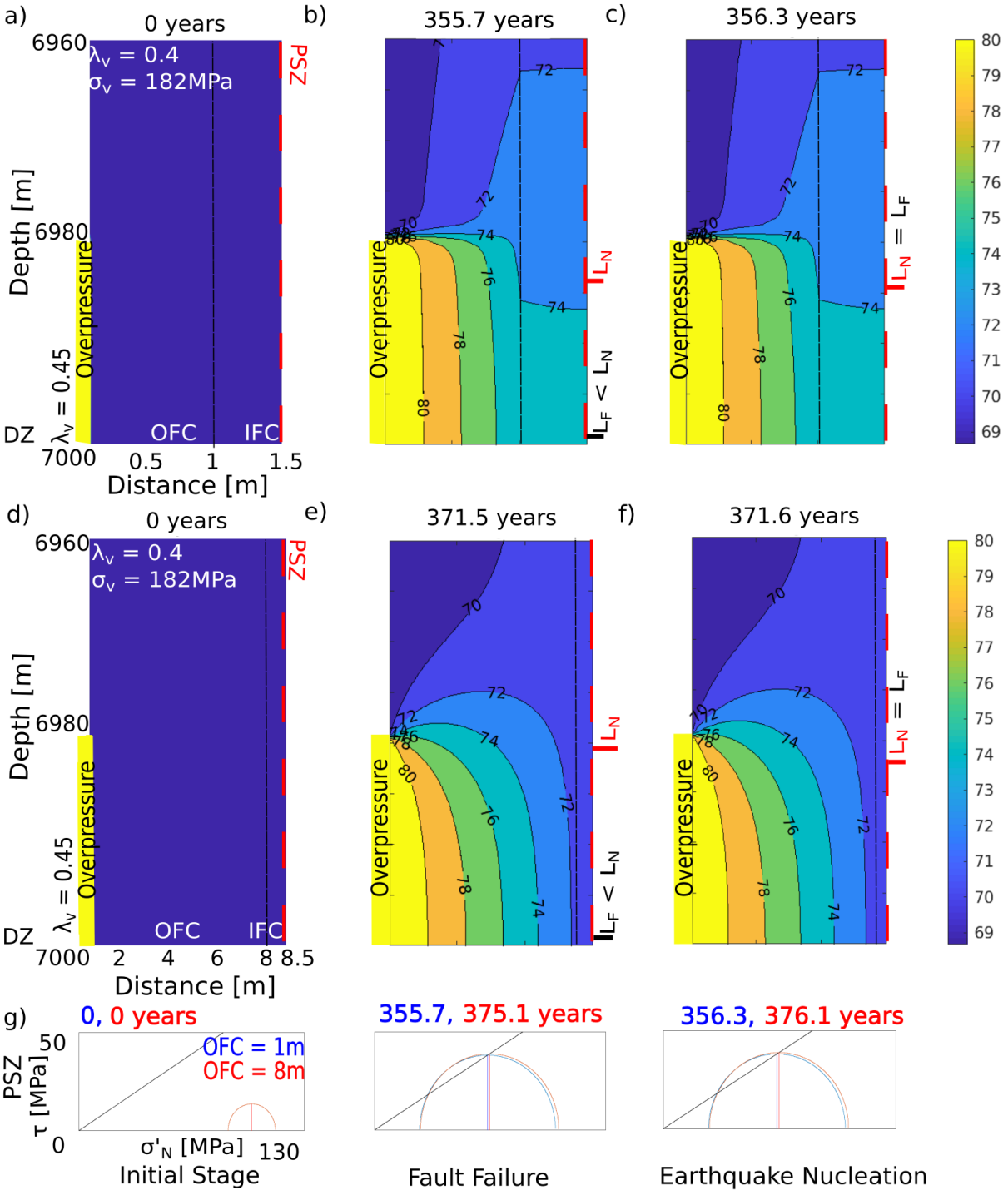



**Figure 5.2: Simulation results of the mode of failure independent pore pressure diffusion model. Plots are provided of pore pressure (with failure ($L_F$) and nucleation length ($L_N$)) and mode of failure (top row with an outer fault core 1m wide and middle row with an outer fault core 8m wide) and Mohr failure analysis (g) for the OFC and PSZ at (a, d) initial conditions, (b, e) the onset of**

**stable and (c, f) unstable sliding. Simulations presented were carried out at a depth of 7km, with a tectonic unloading rate of 0.15MPa/ year in the minimum principal stress direction. The intersecting overpressured aquifer thickness takes a base case value of 40m respectively. (40m of 1km simulated region shown vertically, fault core exaggerated horizontally.) During simulations a millimetre scale horizontal spatial grid was used, and vertically an initially millimetre scale logarithmic grid was used.**

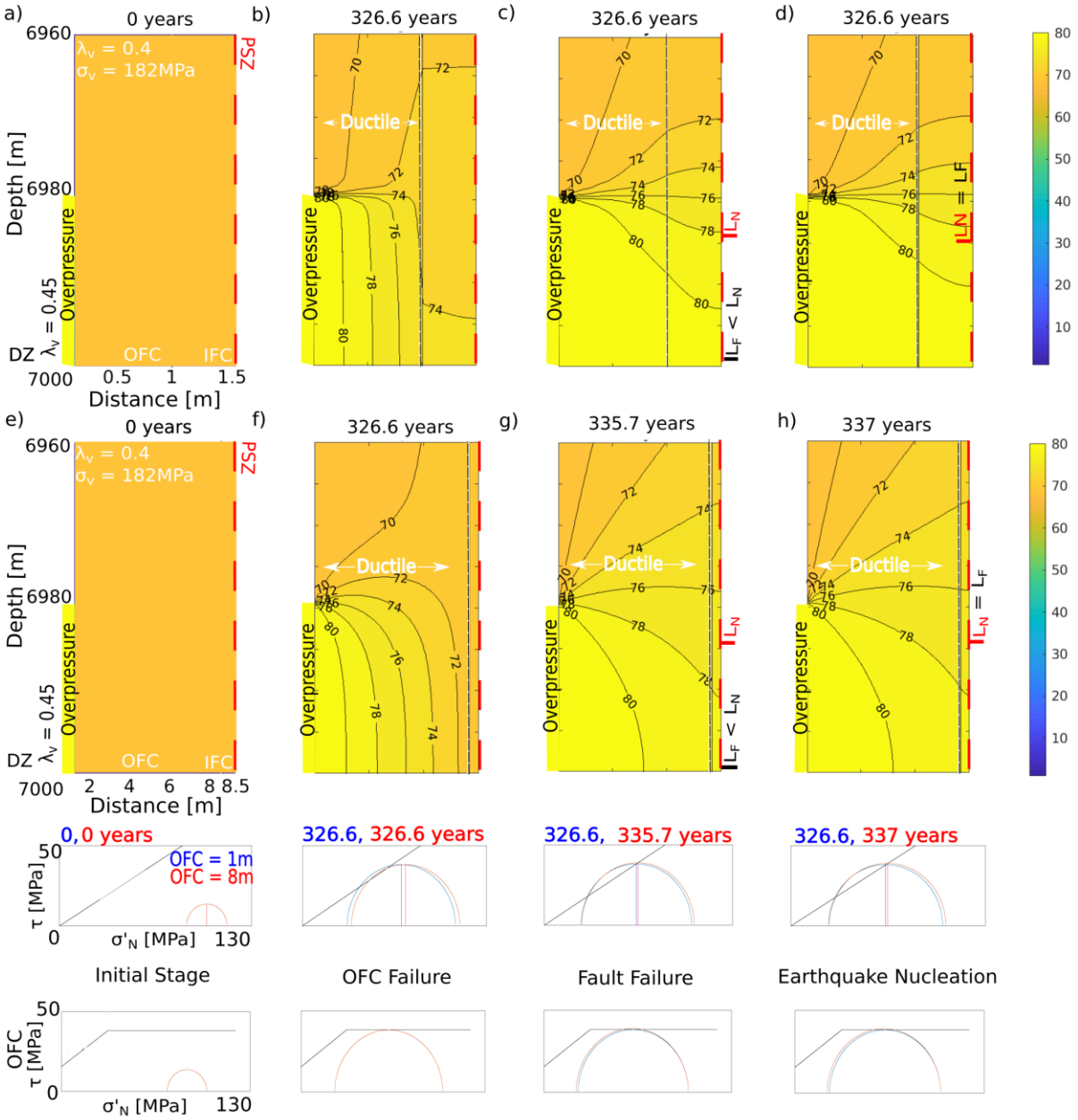
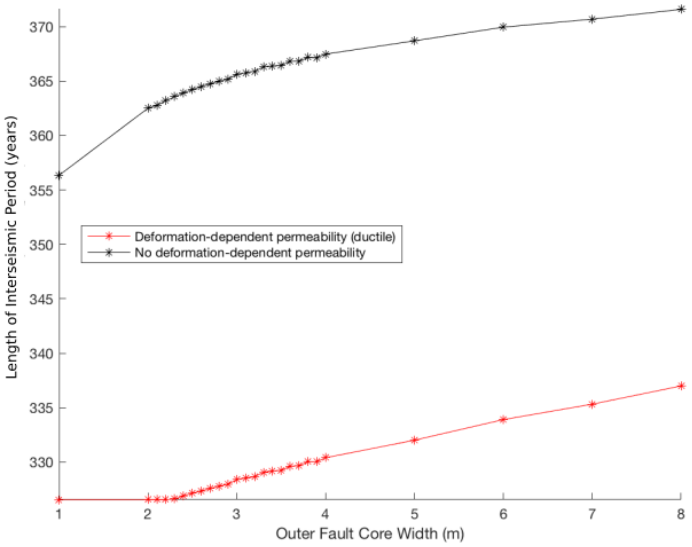


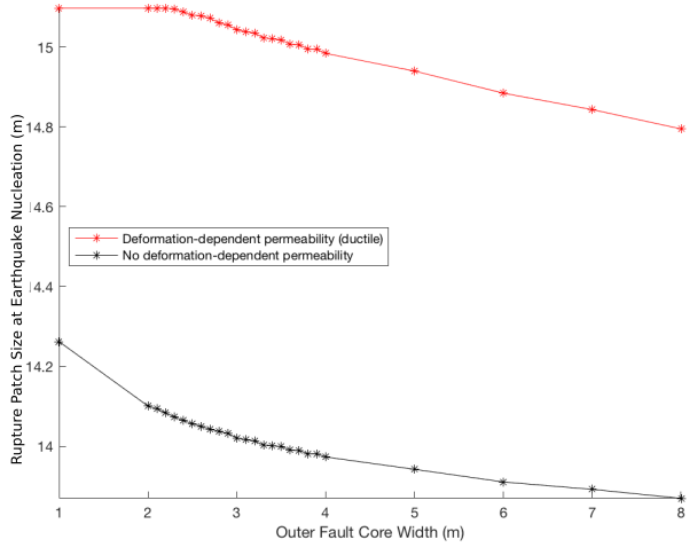

**Figure 5.3: Simulation results of the mode of failure-controlled pore pressure diffusion model. Plots are provided of pore pressure (with failure ($L_F$) and nucleation length ($L_N$)) and mode of failure (top row with an outer fault core 1m wide and middle row with an outer fault core 8m wide) and Mohr failure analysis(i, j) for the OFC and PSZ at (a, e) initial conditions, (b) the onset of ductile failure in the OFC, (f) the onset of brittle failure in the OFC, (c, g) the onset of stable and (d, h) unstable sliding. Simulations presented were carried**

**out at a depth of 7km, with a tectonic unloading rate of 0.15MPa/ year in the minimum principal stress direction. The intersecting overpressured aquifer thickness (overpressure contact height) takes a base case value of 40m respectively. (40m of 1km simulated region shown vertically, fault core exaggerated horizontally.) During simulations a millimetre scale horizontal spatial grid was used, and vertically an initially millimetre scale logarithmic grid was used.**

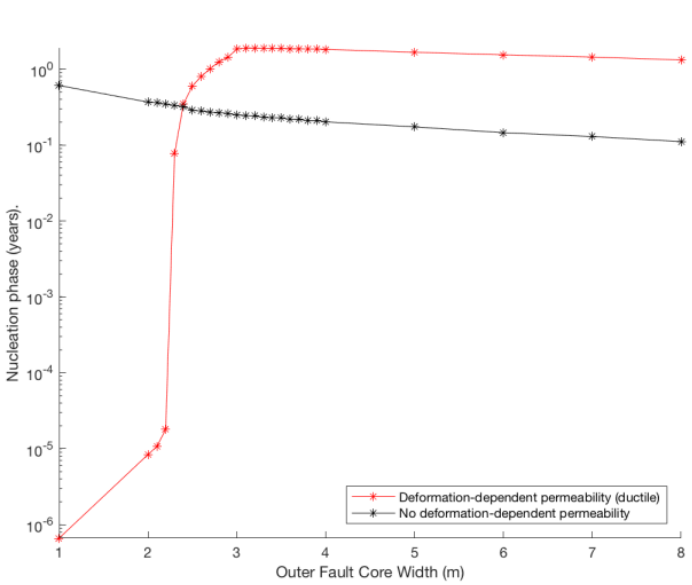
a)




b)




c)




| | | | |
|---|---|---|---|
| Nucleation Phase | 222 days<br>10 seconds | 80 days<br>5 minutes | 40 days<br>7.2 years |

**Figure 5.4: Earthquake nucleation parameters as controlled by the variation of the outer fault core width across multiple simulations, the fault is taken to be at a depth of 7km unloaded at a rate of 0.15MPa/year in the minimum principal stress direction from a critically stressed state. The intersecting overpressured aquifer thickness (overpressure contact height) takes a base case value of 40m respectively. a) Interseismic period. b) Nucleation length. c) Nucleation phase.**

### 5.4.1 Outer Fault Core Width

First, we consider a parameter study in which initially outer fault core width is varied over the range 1-8 m, in steps of 1 m. We ran fluid flow simulations to examine in more detail two end-member case study scenarios, for deformation-independent (Fig. 5.3) and deformation-dependent permeability (Fig. 5.4), respectively. In both case scenarios, the OFC width takes values between 1 m and 8 m. These cases reveal details about pore pressure evolution and failure distribution throughout the fault core, at the extremes of the considered range of the OFC width parameter (Figs. 5.3-4). A further high-resolution parameter study is conducted in the range 2 – 4 m, in steps of 0.1 m, in the vicinity of an observed sharp transition in earthquake nucleation length (Fig. 5.5).

Our results show that at the beginning of the interseismic period, soon after an earthquake event, pore pressure excess is concentrated in the vicinity of the overpressure contact at the DZ/OFC boundary (Fig. 5.3a, d and 5.4a, d). High time resolution simulations show that pore fluids start to rapidly diffuse within the OFC first and then into the IFC and PSZ, where pore pressure increases along the fault-parallel and -perpendicular direction. A quasi-steady state pore pressure regime is attained in the fault zone on the order of days.

Simulations with no deformation control on permeability show that failure by sliding along the PSZ will start at 356 and 372 years, for OFC = 1 m and 8 m, respectively (Fig. 5.3b, e). At this time, the shear stress level, which is controlled by the tectonic loading rate, matches the fault strength, which is dependent on pore fluid pressure. Failure patches begin to develop along the PSZ ($L_F$ in Fig 4.4b, e), a condition that coincides with the beginning of the nucleation phase of the earthquake. The nucleation phase ends when the size of the sliding patch on the PSZ equals that of the nucleation length (i.e. $L_F = L_N$ in Figs. 4.4c, f), a condition leading to the dynamic fast propagation of the rupture. Hence, the computed nucleation length values can be used to estimate the duration of the nucleation stage, which is 6 months and less than 1 month long for OFC = 1 m and 8 m, respectively (Fig. 5.3c, f).

Simulations with deformation control on permeability show that ductile failure in the OFC will start at 326.6 years (Fig. 5.4b, f). This occurs before failure along the PSZ, which will start at 327 years and 336 years, for OFC = 1 m and 8 m, respectively (Fig. 5.4c, g). The duration of the nucleation stage is much less than 1 minute and more than 1 year long for OFC = 1 m and 8 m, respectively (Fig. 5.5d, h).

Overall, simulations with no deformation control on permeability show that the width of the OFC controls the length of the interseismic period (Fig. 5.5a). In particular, the length of the interseismic period increases monotonically with OFC width, with an OFC width of 8 m entailing an interseismic period about 20 years longer than a 1 m wide OFC (356.3 years and 371.6 years respectively, Fig. 5.5a).

The occurrence of ductile failure in deformation-dependent permeability simulations leads to significant increases in the OFC permeability (Fig 4.4c-d).

Compared to the previous case when permeability was not dependent on deformation, the results show an increased pore pressure in the IFC and PSZ after the onset of ductile failure, which reduces the interseismic period for the considered values of OFC width (Fig. 5.5a). The interseismic period is up to 39.1 years shorter for an 8 m wide OFC than for a 1 m wide one (Fig. 5.5a). Further, the interseismic period takes a constant value of 326.6 years in the range of OFC widths 1 - 2.4 m (Fig. 5.5a).

For both cases considered here, we also observe a monotonic decrease in the nucleation length – the length of the rupture patch at the point of earthquake nucleation (i.e. $L_F = L_N$). For the deformation-independent case, an OFC width of 8 m exhibits a nucleation length of 13.87 m, which is 0.39 m smaller than 14.26 m obtained for a 1 m wide OFC (Fig. 5.5b). For the case of permeability controlled by deformation, larger nucleation lengths are observed, which can be up to 1.08 m larger than that obtained for the previous case for an OFC width of 8m (Fig 5.5b).

For the simplest case considered, when permeability was not dependent on deformation, the duration of the earthquake nucleation phase gradually decreases with increasing OFC width (Fig. 5.5c). For an OFC width of 8 m, the nucleation phase length is 0.11 years, which is 0.5 years shorter than 0.61 years obtained for a 1 m wide OFC (Fig. 5.5c).

In simulations considering deformation-dependent permeability, the onset of ductile deformation produces a more complex evolution of the earthquake nucleation phase (Fig. 5.5c). First, the nucleation phase duration increases with OFC width, from the order of 10 seconds to 1.84 years (Fig. 5.5c). However, the trend is not smooth, and a sharp increase by more than three orders of magnitude, from 9.4 minutes to 28.1 days, is observed between OFC widths of 2.2 - 2.3 m (Fig. 5.5c).

Then, for OFC widths > 2.4 m, the nucleation phase duration decreases monotonically from a value of 1.84 years (Fig. 5.5c).

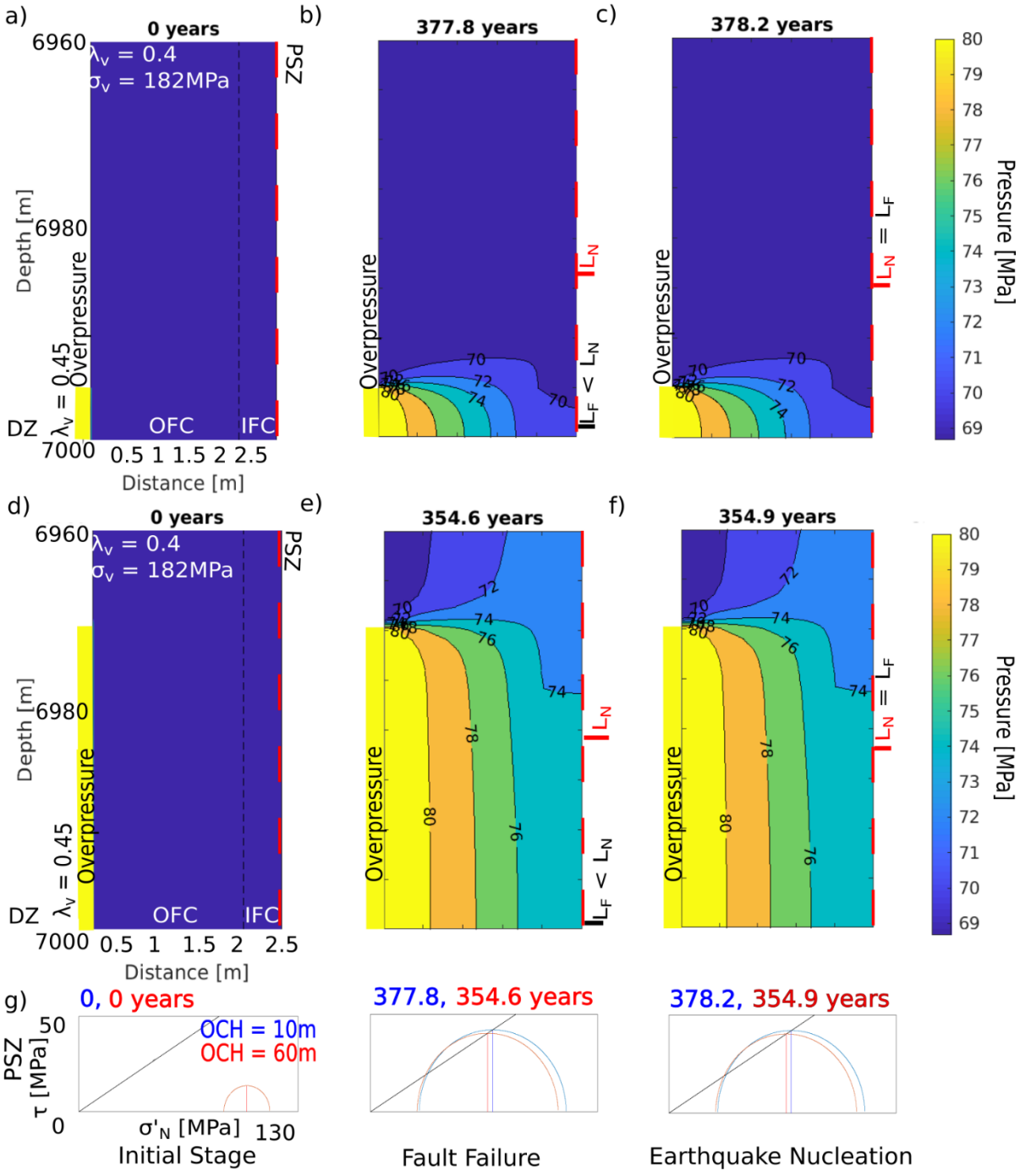



**Figure 5.5: Simulation results of the mode of failure independent pore pressure diffusion model. Plots are provided of pore pressure (with failure ($L_F$) and nucleation length ($L_N$)) and mode of failure (top row with an intersecting**

**overpressured aquifer thickness of 10m and middle row with an intersecting overpressured aquifer thickness of 60m) and Mohr failure analysis(g) for the OFC and PSZ at (a, d) initial conditions, (b, e) the onset of stable and (c, f) unstable sliding. Simulations presented were carried out at a depth of 7km, with a tectonic unloading rate of 0.15MPa/ year in the minimum principal stress direction. The outer fault core width takes a base case value of 2m. During simulations a millimetre scale horizontal spatial grid was used, and vertically a logarithmic grid was used with an initially millimetre scale spatial grid. (40m of 1km simulated region shown vertically, fault core exaggerated horizontally.)**

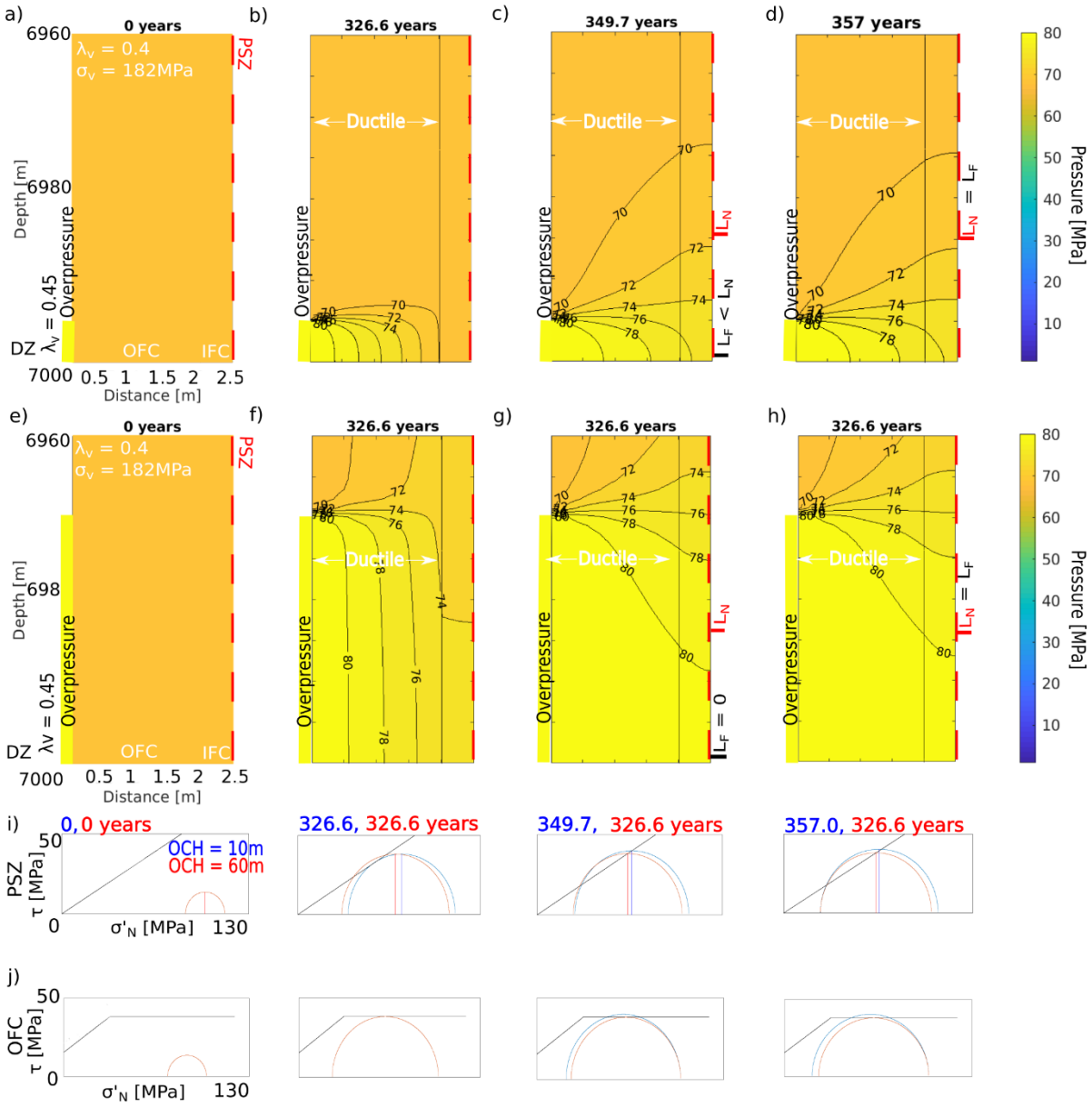
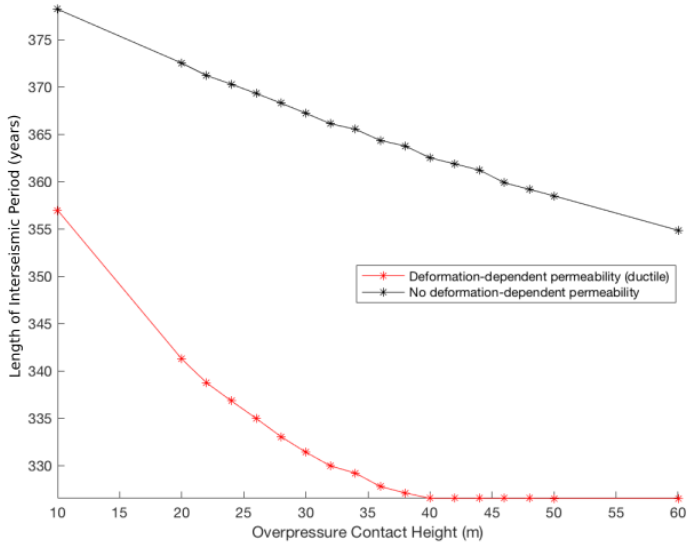


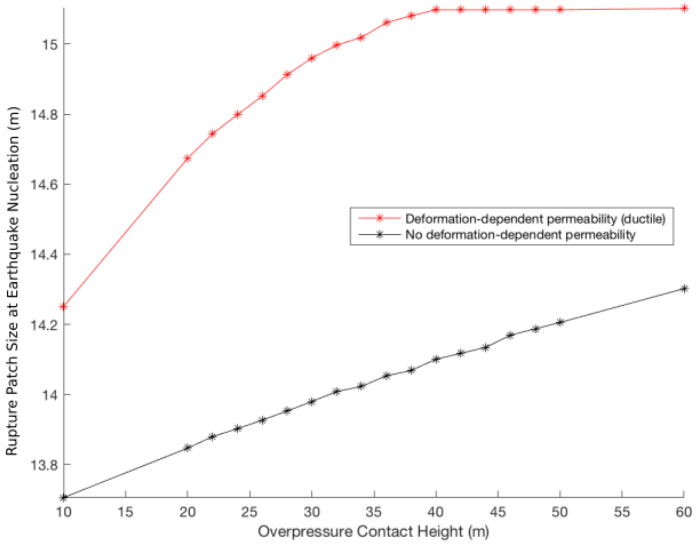

**Figure 5.6: Simulation results of the mode of failure-controlled pore pressure diffusion model. Plots are provided of pore pressure (with failure ($L_F$) and nucleation length ($L_N$)) and mode of failure (top row with an intersecting overpressured aquifer thickness of 10m and middle row with an intersecting overpressured aquifer thickness of 60m) and Mohr failure analysis(i, j) for the OFC and PSZ at (a, e) initial conditions, (b) the onset of ductile failure in the OFC, (f) the onset of brittle failure in the OFC, (c, g) the onset of stable and (d, h) unstable sliding. Simulations presented were carried out at a depth of 7km,**

**with a tectonic unloading rate of 0.15MPa/ year in the minimum principal stress direction. The outer fault core width takes a base case value of 2m. During simulations a millimetre scale horizontal spatial grid was used, and vertically a logarithmic grid was used with an initially millimetre scale spatial grid. (40m of 1km simulated region shown vertically, fault core exaggerated horizontally.)**

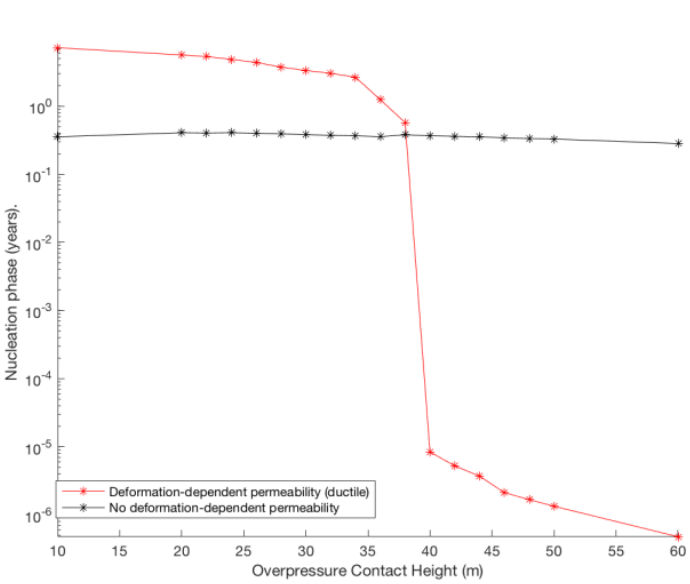
a)




b)



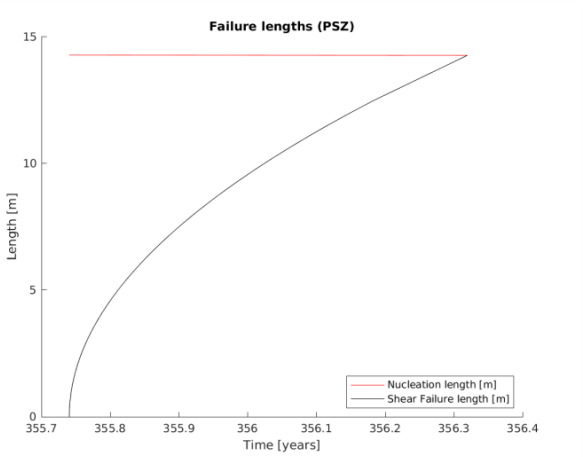

c)



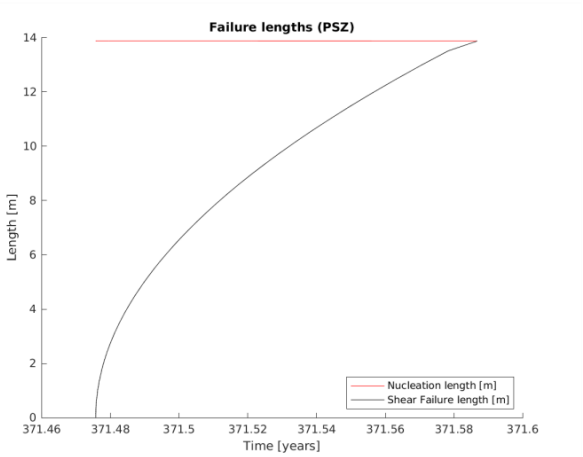

| | | | |
|---|---|---|---|
| Nucleation Phase | 130 days<br>2.62 years | 119 days<br>5 minutes | 103 days<br>15 seconds |

**Figure 5.7: Earthquake nucleation parameters as controlled by the variation of the intersecting overpressured aquifer thickness (overpressure contact height) across multiple simulations, the fault is taken to be at a depth of 7km unloaded at a rate of 0.15MPa/year in the minimum principal stress direction from a critically stressed state. The outer fault core width takes a base case value of 2m. a) Interseismic period. b) Nucleation length. c) Nucleation phase.**

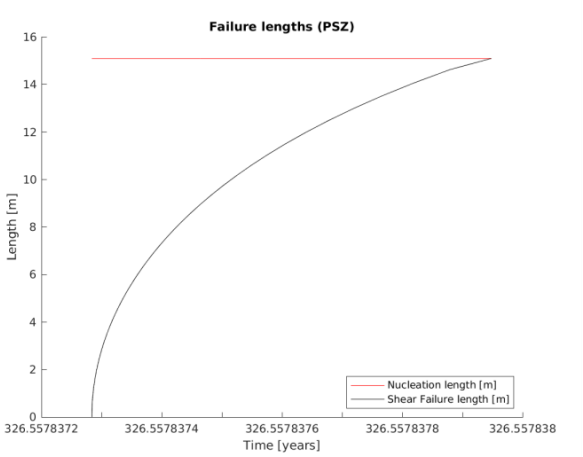
a)


Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

b)


Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

c)


Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

d)

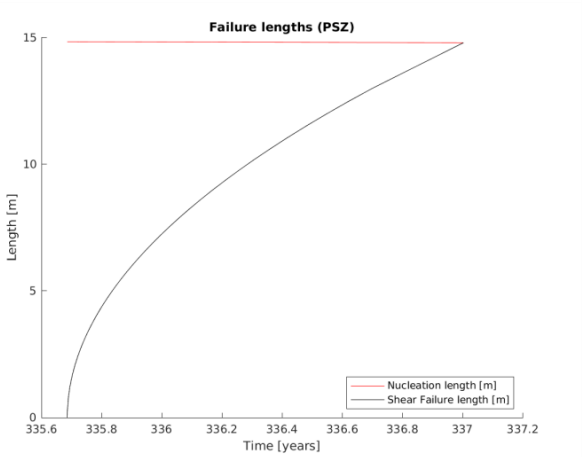
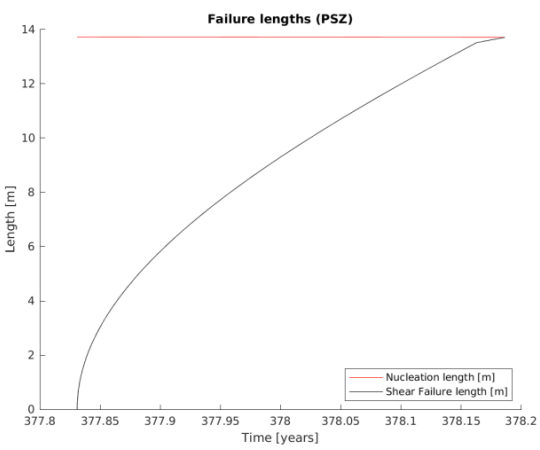
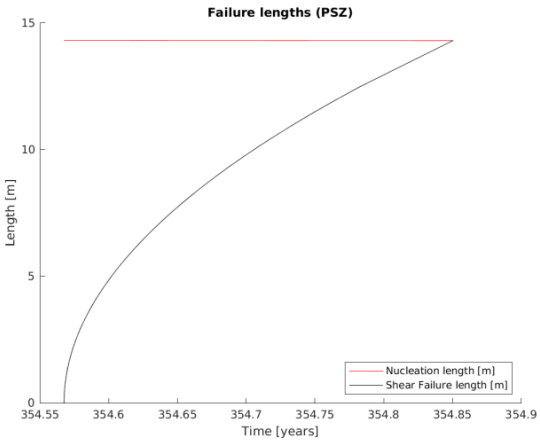

Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

**Figure 5.8: Earthquake failure and nucleation length evolution for the end member case studies in intersecting overpressured aquifer thickness, the fault is taken to be at a depth of 7km unloaded at a rate of 0.15MPa/year in the minimum principal stress direction from a critically stressed state. The intersecting overpressured aquifer thickness takes a base case value of 40m respectively. a) No deformation-dependent failure 1m outer fault core. b) No deformation-dependent failure 8m outer fault core. c) Deformation-dependent failure 1m outer fault core. d) Deformation-dependent failure 8m outer fault core.**

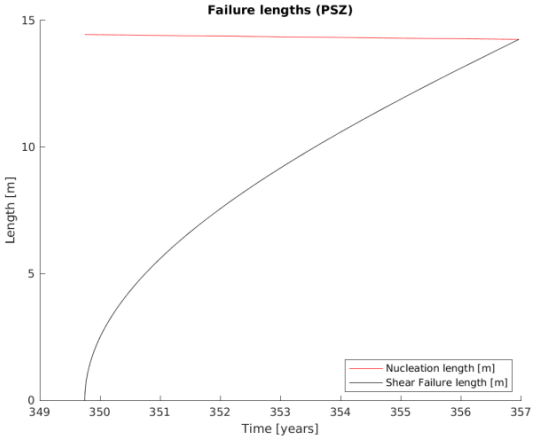
a)


Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

b)


Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

c)


Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

d)

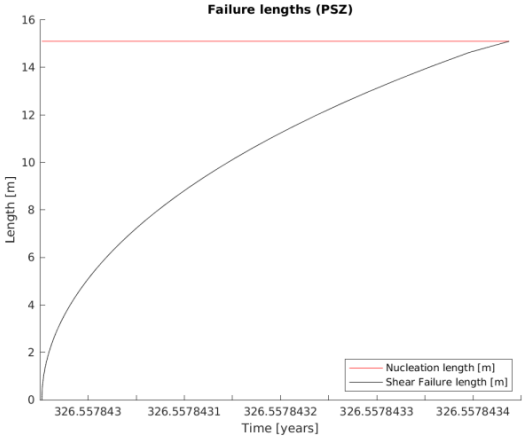

Failure lengths (PSZ)
Length [m]
Time [years]
Nucleation length [m]
Shear Failure length [m]

**Figure 5.9: Earthquake failure and nucleation length evolution for the end member case studies in intersecting overpressured aquifer thickness, the fault is taken to be at a depth of 7km unloaded at a rate of 0.15MPa/year in the minimum principal stress direction from a critically stressed state. The outer fault core width takes a base case value of 2m. a) No deformation-dependent failure 10m overpressure contact. b) No deformation-dependent failure 60m overpressure contact. c) Deformation-dependent failure 10m overpressure contact. d) Deformation-dependent failure 60m overpressure contact.**

### 5.4.2 Intersecting overpressured aquifer thickness

We consider another parameter study in which lithological anisotropy in the protolith control the size of the fractured dolostone layers and, hence, of the intersecting overpressured aquifer thickness in the damage zone reservoir (Fig. 3.2).

In our simulations, the thickness of dolostone layers and, hence, the overpressure height is varied from 10 to 60 m in steps of 10 m. We ran fluid flow simulations to examine in detail two case study scenarios, for deformation-independent (Fig. 5.6) and deformation-dependent permeability (Fig. 5.7), respectively. In both case scenarios, the OC height takes values of 10 m and 60 m. These cases reveal details about pore pressure evolution and failure distribution throughout the fault core, at the extremes of the considered range of the OC height parameter (Figs. 5.6-7). The other fault zone parameter of relevance, the OFC width, takes a base case value of 2 m.

Simulations of fluid flow with no deformation control on permeability show that failure by sliding along the PSZ will start at 378 and 355 years, for OC = 10 m and 60 m, respectively (Fig. 5.6b, e). The nucleation phase duration is 4 months and 3 months for OC = 10 m and 60 m, respectively (Fig. 5.6c, f).

Simulations of fluid flow with deformation control on permeability show that ductile failure in the OFC will start at 326.6 years (Fig. 5.7b, f). This occurs before failure along the PSZ, which will start at 349.7 years and 326.6 years, for OC = 10 m and 60 m, respectively (Fig. 5.7c, g). The duration of the nucleation stage is more than 7 years and much less than 1 minute for OC = 10 m and 60 m, respectively (Fig. 5.7d, h).

Overall, simulations with no deformation control on permeability show that the OC height controls the length of the interseismic period (Fig. 5.8a). In particular, the length of the interseismic period decreases monotonically with OC height, with an OC height of 60 m entailing an interseismic period about 23.4 years shorter than a 10 m thick OC height (378.2 years and 354.9 years respectively, Fig. 5.7a-h; Fig. 5.8a). Increasing OC height now increases the extent and magnitude of the pore pressure that develops in the IFC and, subsequently, decreases the shear strength of the PSZ.

When permeability is dependent on deformation, fluid flow simulations including ductile failure in the OFC show that the interseismic period reduces by up to 33.3 years for a 60 m OC height compared to 357 years for a 10 m height one (Fig. 5.8a). Further, the interseismic period takes a constant value of 326.6 years in the range of OC heights > 40 m (Fig. 5.8a).

For both cases considered here, we also observe a monotonic increase in the nucleation length (Fig. 5.8b). For the deformation-independent case, an OC height of 10 m exhibits a nucleation length of 13.7 m, which is 0.6 m smaller than 14.3 m obtained for a 60 m high OC (Fig. 5.8b). For the case of permeability controlled by deformation, larger nucleation lengths are observed, which can be up to 0.8 m larger than that obtained for the previous case for an OC height of 60 m (Fig 5.6b).

For the simplest case considered, when permeability was not dependent on deformation, the duration of the earthquake nucleation phase shows a subtle decrease from 4 and 3 months for OC height of 10 m and 60 m, respectively (Fig. 5.8c).

In simulations considering deformation-dependent permeability, the onset of ductile deformation produces a more complex evolution of the earthquake nucleation phase (Fig. 5.8c). First, for OC heights between 10 m and 35 m, the nucleation phase duration decreases monotonically from values of 2.62 to values of 1.3 years (Fig. 5.8c), respectively. Then, a sharp decrease by almost 5 orders of magnitude, from 209 days to 264 seconds, is observed between OC heights of 35 - 40 m (Fig. 5.8c). Finally, the nucleation phase duration decreases more gradually from 264 to 15 seconds, for OC height increasing from 40 m to 60 m (Fig. 5.8c).

## 5.5. Discussion and Conclusions

### 5.5.1 Dimensions of fault zone architecture and lithological variations in the protolith control pressure diffusion and earthquake recurrence interval

We model fault zone fluid flow considering the simplest case scenario where permeability of the fault core does not depend on deformation, but solely on pore pressure. Our results show that the thickness of the fault core domain and the intersecting overpressured aquifer thickness in a reservoir in the damage zone, abutting the fault core, act as controls on pore pressure diffusion.

For the specific fault zone architecture considered in our case study, an increased width of the OFC acts as a barrier to fluid flow. It reduces the extent and magnitude of the pore pressure that develops in the IFC and, subsequently, affects the shear strength of the PSZ (Fig 5.3 a-f). For a given tectonic loading rate, a thinner fault core will result in a more effective fault weakening, as the fault frictional strength will reduce at a faster rate due to higher pore pressures (Fig. 5.3b).

The impact of fluid flow on the fault being more significant for faults with a thinner rather than thicker outer fault core. In the absence of fluids, the base mechanical strength of the slipping portion of the fault did not vary with thickness. However, a thinner fault core shows a larger rupture patch at the point of earthquake nucleation (Fig. 5.5b) and a longer nucleation phase (Fig. 5.5c).

Similarly, an increasing intersecting overpressured aquifer thickness in the damage zone produces a higher magnitude of pore pressure in the fault core, which weakens the principal slip zone located in the centre of the fault core. A higher overpressure contact will result in a larger rupture patch at the point of earthquake nucleation (Fig. 5.8b) and in a longer nucleation phase (Fig. 5.8c).

These counterintuitive results are due to the heterogeneous distribution of pore fluid pressure within the IFC and, particularly, along the PSZ. Pore pressure values within the IFC and along the PSZ control the strength of the PSZ, which is reduced by lower effective normal stresses at higher pore pressures. On the other hand, the imposed tectonic loading controls the shear stress build-up along the PSZ, which is independent of pore fluid pressure and operates at same rate for any investigated case. Failure along the PSZ first occurs in patches where higher pore fluid pressures have reduced the PSZ strength, and then slowly spreads out along the PSZ due to shear stress increase by tectonic loading. For a wider OFC or lower intersecting overpressured aquifer thickness, the delayed onset of failure along the high pressure PSZ means that, outside the pressurised PSZ patch, shear stress is relatively high and close to the PSZ shear strength. This makes the nucleation phase relatively short as small amounts of tectonic loading, hence short times, are needed to grow the sliding patch to the size of the nucleation length (Fig. 5.9a-b; Fig. 5.10a, b). On the other hand, for thinner OFCs and higher intersecting overpressured

aquifer thickness, the anticipated onset of failure means that larger amount of tectonic loading, hence longer times, are needed to grow the sliding patch outside of the pressurised PSZ patch, where sliding first initiated (Fig. 5.9a-b; Fig. 5.8a-b).

It is well known that the scale of lithological variations in the protolith, controlling the size of overpressure reservoir in the damage zone, and fault zone architecture significantly affect the hydrogeological conditions of fault zones. Here, our results show that these parameters also control the evolution of fault strength during the seismic cycle and, hence, the length of the interseismic period and the duration of the nucleation phase of an earthquake. These findings have relevant implications for estimates of seismic hazards, such as the recurrence interval of earthquakes.

### 5.5.2 Ductile deformation in the fault core controls pore pressure diffusion during the seismic cycle

The occurrence of fluid driven ductile, distributed fracturing in the fault core acts as a primary control on pore pressure diffusion during the interseismic period. In fact, ductile failure can increase the permeability of fault core rocks by several orders of magnitude and enhances the diffusion of pore pressure toward the primary slip zone, located into the centre of the fault.

#### *5.5.2.1 Failure initiation and duration of the interseismic period*

The occurrence of ductile failure in the fault core causes fault weakening along the PSZ at a faster rate than when fluid flow is not affected by deformation in the fault core.

Our results show that, for a given constant tectonic loading rate and in the presence of ductile deformation in the fault core, failure along the PSZ of the fault

can initiate earlier for thinner fault cores (OFC) and higher contact pressure (OC) heights (Fig. 5.5a, 5.8a).

Similarly, to the previous case examined in Section 5.1, early fault initiation is due to the larger input of pore fluid pressure produced by a thinner OFC or a higher OC, following ductile failure in the fault core, which causes the weakening of the PSZ. For a thicker OFC and a smaller OC, a larger input of steady tectonic loading, hence a longer time, is required to trigger failure along the PSZ.

The implications of these results are that the duration of the interseismic period is also affected by the occurrence of ductile failure in the fault core, and by fault zone architecture and lithological variations in the protolith. In our specific case study, a critical fault core thickness (OFC < 2 m) and intersecting overpressured aquifer thickness (OC > 40 m) can be identified, beyond which the duration of the interseismic period does not vary. The attainment of a characteristic interseismic period duration, controlled by local fault zone factors, is due to the development of a failure patch whose dimension are comparable to those of the nucleation length, both being dependent on pore pressure.

These results again have implications for seismic hazards evaluation, such as the estimation of the recurrence interval of earthquakes.

#### *5.5.2.2 Earthquake nucleation phase and premonitory signal detection*

When pore pressure distribution during the interseismic is not affected by deformation in the fault core, our results show that the nucleation length monotonically increases and the duration of the nucleation phase is gradually longer for thin fault cores and greater intersecting overpressured aquifer thicknesss (Fig. 5.5b-c, 5.8b-c).

The increase in nucleation phase duration can be explained by the pore pressure distribution within the IFC, which controls the strength of the PSZ. A thinner OFC and a larger OC height will reach failure along the pressurised portion of the PSZ earlier, due to a high pore pressure in the fault core. However, they will still result in a longer nucleation phase, due to the lower shear stress level outside the pressurised portion of the PSZ, which requires a longer time of steady tectonic loading to grow the failure patch to the size of the nucleation length (Fig. 5.9a-b; 5.10a,b).

In simulations considering deformation-dependent permeability, the onset of ductile deformation produces a more complex evolution of the earthquake nucleation phase (Fig. 5.3c, 5.6c).

These results can be explained by rapid and transient fluctuations in pore pressure within the IFC and, particularly, along the PSZ, after ductile failure is activated in the OFC. For thicker OFC (> 3 m) and low values of OC height (< 35 m), steady state pore pressure conditions are attained in the fault core at the time of failure initiation within a pressurised patch along the PSZ (Fig. 5.4c-d, 5.7c-d), which is smaller than the nucleation length. This means that, after sliding initiation, the growth of the failure patch is relatively slow and solely controlled by steady shear stress increase by tectonic loading (Fig. 5.9d; 5.10c). For a thinner OFC, between 2.4 m and 3 m, and an increasing OC height, between 35 m and 55 m, a sharp first and then more gradual decrease in the duration of the nucleation length is observed (Fig. 5.5c, 5.7c). This can be explained by transient pore pressure conditions in the fault core, caused by ductile failure at the time of fault initiation in a pressurised patch along the PSZ (Fig. 5.5b-d, 5.7b-d). In fact, the subsequent growth of the failure patch is relatively fast and controlled by transient pore pressure

evolution along the PSZ and steady shear stress increase by tectonic loading. Finally, for thin OFC (< 2.4 m) and large OC heights of 60 m, failure will initiate along the PSZ with a several metre-long failure patch, of similar size compared to the nucleation length. The failure patch can very quickly grow to the size of the nucleation length, nucleating an earthquake on the order of seconds (Fig. 5.9c, 5.10d).

In simulations without ductile failure (Fig. 5.5c, 5.8c) or in simulations where ductile failure is significantly removed in time from the onset of stable sliding, the nucleation phase is on the order of days to years. On the other hand, simulations where ductile failure immediately precedes failure along the PSZ have nucleation phases on the order of seconds.

Our parameter studies often predict large aseismic slip episodes, which can precede the nucleation of an earthquake, and last for months to years (Fig. 5.5c, 5.8c). Evidence for aseismic slip episodes preceding major earthquake events are supported by independent geophysical observations for the recent $M_w$ 9, 2011 Tohoku earthquake (Kato et al., 2012; Mavrommatis et al., 2014) and the $M_w$ 8.1, 2014 Iquique earthquake (Ruiz et al., 2017; Socquet et al., 2017). It has been interpreted that these early slip events are related to the preparatory nucleation phase of the main events (Guglielmi et al., 2015).

Understanding the controls exerted on the duration of the nucleation phase of earthquakes has important implications for premonitory signal detection, as identifying extended nucleation phases of active faults would increase the likelihood of detection of early seismicity warnings. Our case study shows that, for a given fault, characteristic values of fault zone parameters (e.g. fault core width and intersecting overpressured aquifer thickness) govern the transition from relatively

long – on the order of days to months – easily detectable nucleation phase to very short ones – on the order of seconds to minutes – difficult to detect. As such, realistic estimates of uncertainty in fault zone architecture dimension must inform hazard estimates, as small differences in scale can correspond to significant variation of the nucleation phase, from seconds to years.

# CHAPTER 6

*Final discussions and conclusions*

# 6. Final discussion and conclusions

## 6.1 Introduction

In this thesis I have performed a series of numerical simulations and parameter studies of pore pressure diffusion and fluid flow in fault zones with a realistic complex architecture. In my simulations, fault zone permeability is controlled by pore pressure and deformation, as observed in laboratory experiments. The results of my simulations are then used in some specific case studies of seismically active fault zones, where I simulate pore pressure distribution during the seismic cycle. In particular, the role played by pore pressure evolution and distribution during the interseismic period and the nucleation phase preceding an earthquake is investigated.

The simulations are constrained by a simplified low-porosity fault zone model, which still encompasses the essential features of natural seismogenic fault zones with complex architecture. The numerical simulations are conducted using a multiphysics model of nonlinear diffusion in low porosity fault zones, which in turn incorporates: 1) models of fault zone with complex architecture; 2) brittle and ductile mode of failure within the fault core domain; 3) pore pressure- and deformation-dependent permeability constrained by triaxial deformation experiments with fluid flow; 4) basic assumptions about the physics of the earthquake nucleation processes.

Mathematically, I present a treatment of the fault zone and the fluids it contains as a non-smooth dynamical system. The results of these studies are then discussed in detail at the end of each of the research chapters. This final discussion chapter aims to highlight and contrast the main findings of the preceding chapters. I will also discuss the implications for fluid-induced earthquake nucleation,

earthquake forecasting and premonitory signal detection. Opportunities for future work will be signposted when discussing the results and their implications.

### 6.2 Summary and comparison of main findings

In Chapter 4, we present a parameter study of pore pressure diffusion from an overpressured reservoir, located in the fault damage zone abutting the fault core. Two end-member case studies at the extremes of the pore pressure range – e.g. supra-hydrostatic vs. sub-lithostatic conditions – were examined in further detail. The fault zones used to establish the base case, around which pore fluid factor was varied, are analogues of the seismic sources in the hypocentre zone of the Northern Apennines seismic belt (e.g. Mirabella et al., 2008; De Paola et al., 2008; Collettini et al., 2009). In simulations, we considered that faults were saturated with supercritical $CO_2$, at a hypocentre depth of 7 km. This was consistent with borehole measurements, which show the presence of overpressured $CO_2$ within the Triassic Evaporites host rocks at up to 80% of the lithostatic load (Chiodini and Cioni, 1989; Collettini and Barchi, 2002; Miller et al., 2004).

Fault zone architecture is taken from field observations of analogous evaporite faults reporting a 3 m wide fault core with a complex internal structure, where most of the seismic slip accommodated by the fault is localised (Collettini et al., 2009; De Paola et al., 2008). Outside the fault core, a well-developed damage zone (DZ) is observed within thick (a few meters to tens of meters) fractured dolostones interbedded with undeformed foliated anhydrite, extending at least 10 m in either direction from the fault core. At depth, high porosity fractured dolostones and low porosity foliated anhydrite layers in the DZ are believed to act as an efficient reservoir-seal system, where supercritical $CO_2$ over- pressurise can develop.

In Chapter 5, we extend this analysis to a more generalised parameter study, where fault zone parameters are systematically varied to account for the variability of fault zone structure and dimensions observed in nature. In particular, we retain the same base complex fault zone structure from analogues of the seismic sources in the hypocentre zone of the Northern Apennines seismic belt. What we vary instead are the dimensions of the lithological heterogeinity in the protolith and damage zone, and the dimensions of the fault core domains. The specific parameters varied are the width of the outer fault core (OFC) and the thickness of the pressurised reservoir in the DZ, while pore fluid factor is held constant.

### 6.2.1 Mode of failure controls pore pressure diffusion during the seismic cycle and earthquake recurrence interval

In previous studies, numerical simulation techniques have been used to analyse fault reactivation. These studies model pore pressure distribution and subsurface fluid flow within faults, which eventually cause fault reactivation (Cappa and Rutqvist, 2011a, 2012; Rutqvist et al., 2015, 2013a, 2002). Further, a number of metre- to kilometre-scale models have refined these results to include simplistic models of fault zone architecture and pore pressure dependent fault zone transport properties (Cappa et al., 2009; Cappa and Rutqvist, 2011; Hsiung et al., 2005; Mazzoldi et al., 2012; Rinaldi et al., 2014; Rutqvist et al., 2013, 2009).

The simulations within this thesis represent a novel implementation of previous studies, which is applied to a new case study of seismic faults from the Italian Northern Apennines seismic belt. These simulations also represent an extension of simulations in previous studies, as they include complex and realistic models of fault zone architecture and the role played by different model of failure, e.g. brittle vs. ductile, in the fault core. Rock failure and deformation-controlled

porosity/permeability are considered in these simulations, throughout the fault core and away from any primary slip zones. Both deformation and pore pressure dependency of rock transport properties observed in triaxial deformation experiments with fluid flow is retained in my simulations. Failure envelopes and permeability tensor of anhydrite rocks in the fault core domains are obtained from triaxial loading experiments performed on anhydrite borehole samples, with foliation oriented in a sub-parallel and sub-orthogonal direction to loading and fluid flow direction (De Paola et al., 2009).

Deformation by brittle and ductile mode of failure both acted as primary controls on pore pressure diffusion into fault cores. In my simulations, when fracture patterns were created by either brittle or ductile mode of failure the average fault core permeability increased by several orders of magnitude. Anisotropic permeability variations in the fault core affect both the fault parallel and fault normal pore pressure distribution. In all cases, the occurrence of failure during the interseismic period enhanced pore pressure diffusion into the fault core, compared to simulations where failure in the fault core was not accounted for. This resulted in an earlier fault failure, where slip initiated along the main slip zones with a larger failure patch. Further, the magnitude of permeability increase was greater in simulations where brittle failure occurred, so too was the development of pore pressure into the fault core. My results show that mode of failure occurring in the fault core does affect pore pressure distribution and fluid flow, which have an impact on the duration of the interseismic period and on the size of the initial rupture patch. Such effects are of the same order of magnitude as local factors, such as fault core thickness and the level of pressure and thickness of pressurised reservoirs in the damage zone. As such, the aforementioned simulations which consider simple fault

zone architecture and generic deformation, neglecting the role of specific mode of failure, did not consistently consider all primary factors at work (Rinaldi et al., 2014).

In our specific case study, a critical fault core thickness (OFC < 2 m) and intersecting overpressured aquifer thickness (OC > 40 m) can be identified beyond which the duration of the interseismic period does not vary. We believe that the attainment of a characteristic interseismic period duration, controlled by local fault zone factors, is due to the development of a failure patch whose dimension are comparable to those of the nucleation length, both being dependent on pore pressure.

### 6.2.2 The role of pore-fluid pressure during the earthquake nucleation phase

Simulations presented in Chapters 4 and 5 show that both the inclusion of realistic models of fault zone architecture and deformation-dependent permeability (brittle and ductile mode of failure) control both the size of the sliding patch during earthquake nucleation and the theoretical nucleation length, which is the predicted critical size required to start the propagation of fast seismic sliding. The conditions also affect the duration of the nucleation phase, which is the time interval between the initiation of slow failure and onset of fast seismic sliding along the main slip zone. Simulation of the earthquake nucleation phase has also given insight into the role that aseismic slip plays before a major earthquake is triggered. Further, it is evident from the parameter studies of Chapters 4 and 5 that the variation of each of these parameters acts to enhance or inhibit the development of pore pressure in the centre of the fault.

Increasing the pore fluid factor, thickness of the pressurised reservoir in the damage zone abutting the fault core and decreasing the thickness of the fault core all act to enhance the magnitude of pore pressure in the fault core. Subsequently, the

duration of the interseismic period is reduced, as an earthquake can nucleate on the fault plane sooner, with a larger rupture patch. However, my results show that the length of the nucleation phase is affected by the presence and timing of ductile deformation occurring in the fault core. In simulations without ductile failure or in simulations where ductile failure is significantly removed in time from the onset of sliding along the main slip zone, the nucleation phase duration is on the order of days to years. On the other hand, simulations where ductile failure immediately precedes failure along the main slip zone have nucleation phases lasting on the order of seconds.

In the simulations presented in this thesis, the lithostatic pressure and hydrogeological conditions considered determine that any fluid present in the fault would be in a single supercritical phase, and consistently several orders of magnitude below the criteria for non-Darcy flow (Thauvin and Mohanty, 1998). However, the same simulation techniques could be applied more broadly, and to shallower faults, if simulations of compressible, multiphase and/or multicomponent flow were considered, where the criteria for non-Darcy flow is met (e.g. Goudarzi, Mathias, & Gluyas, 2016).

## 6.3 Broader implications of main findings

### 6.3.1 Implications for fluid-induced seismicity and earthquake forecasting

The results of Chapter 4 suggest that the level of pore pressure in pressurised reservoirs in the damage zone acts as a primary control on the diffusion of pore pressure into the fault core. My results show that the resultant pore pressure distribution affects the length of the interseismic period and the size of the rupture patch. Increasing the level of pore pressure of supercritical $CO_2$ in the damage zone

reservoir acts to increase the length of the rupture patch and the duration of the nucleation phase, while decreasing the length of the interseismic period. Rupture patch dimensions have been related to rock transport properties in broadly similar studies in different locations on metre to kilometre scales. However, none of the previous studies considered the role played by local factors, and their effects on the duration of the interseismic period or nucleation phase (Cappa, 2009; Cappa and Rutqvist, 2011b; Mazzoldi et al., 2012; Rutqvist et al., 2007).

In Chapter 5 I show that the thickness of the pressurised reservoir in the damage zone and of the fault core also act as primary controls on the diffusion of pore pressure into the fault core. These fault zone parameters influence the length of the interseismic period and the size of the rupture patch. In this case, decreasing the thickness of the outer fault core and increasing the thickness of the pressurised reservoir in the damage zone act to increase the length of the rupture patch, while decreasing the length of the interseismic period. Also, both findings presented in Chapter 4 and 5 suggest that the inclusion of complex, realistic models of fault zone architecture alters interseismic period, nucleation length and the length of the nucleation phase. Hence, they are necessary to accurately simulate pore pressure diffusion and earthquake nucleation in low-porosity rocks.

Similar results have been recovered from TOUGH-FLAC simulations of carbon sequestration operations in interbedded shale and limestone (Rinaldi et al., 2014b). However, in these simulations much simpler fault zone architecture were considered, and rock the relations between rock physical parameters (e.g. structural porosity due to deformation and permeability) and pore pressure were more general and not constrained with triaxial deformation experiments with fluid flow. Further, the specific role played by mode of failure, e.g. brittle vs. ductile mode of failure,

and the effects acused on earthquake nucleation were not considered (Rinaldi et al., 2014) in these studies.

Simulations presented in both Chapters 4 and 5 show that small-scale fracturing acts to increase fault core permeability by several orders of magnitude and, therefore, to increase the development of pore pressure in the fault core. Fault zone parameters, such as fault core domains thickness, control the occurrence and relative timescale of small-scale fracturing in the fault core. Further control on the timescale of small-scale fracturing is exerted by the initial hydrogeological conditions of the fault zone (e.g. level of pressure and thickness of connected reservoirs in the damage zone) and their evolution during the interseismic period. Hence, information on coseismic fluid discharge and fluid recharge between the fault zone and the connected aquifers could thus be inferred from the hydrogeological monitoring of springs (e.g., Barberio et al., 2017) and boreholes in the epicentral area or in the surrounding areas of injection sites. These observations could then be used to estimate the pore pressure evolution of the fault zone and its surroundings during the seismic cycle.

To conclude, key fault zone parameters and lithological variations in the protolith control small-scale fracturing in the fault core, which can modulate the length of the interseismic period. Low resolution of indirect measurement methods and generic inference from specific natural analogues make fault zone parameters poorly constrained. Hence, numerical simulations and multiphysics models of seismic faults are a useful tool to predict the distribution and evolution of pore fluid pressure during the seismic cycle.

### 6.3.2 Implications for earthquake nucleation phase duration and premonitory signal detection

The parameter studies of pore pressure in a supercritical $CO_2$ reservoir in the damage zone in Chapter 4 indicate that it controls the length of the nucleation phase, thus increasing levels of pore pressure in the fault core and extending the duration of the nucleation phase by several years. This nucleation length control is much more pronounced when deformation dependent permeability is considered. The dependence of nucleation phase on pore pressure is mediated by the nonlinear trade-off as increasing pore pressure decreases fault strength but also decreases fault stiffness and therefore increases the critical patch size required for earthquake nucleation. A secondary layer of complexity is added when small scale fracturing in the fault core, accommodated by brittle or ductile mode of failure, is also considered. This fracturing acts to reduce the length of the nucleation phase to more realistic values, ranging from a few seconds to a few days, compared to several years required in simulations with no deformation in the fault core accounted for.

In Chapter 5, the thickness of the pressurised reservoir in the damage zone and the thickness of the fault core are both shown to control the length of the nucleation phase, when no deformation in the fault core is accounted for. In both cases, increasing thickness acts to reduce the length of the nucleation phase. However, the inclusion of small-scale fracturing, accommodated by brittle or ductile mode of failure in the fault core, reveals some rather complex evolution patterns of the nucleation length, which in some cases extends by several years and, in other, reduces to the order of seconds.

Our case studies in chapter 4 and 5 show that, for a given fault, characteristic values of fault zone parameters (e.g. fault core width and intersecting overpressured

aquifer thickness) govern the transition from relatively long (days to months) and easily detectable nucleation phase to very short ones (seconds to minutes) and difficult to detect. As such, realistic estimates of uncertainty in fault zone architecture dimension must inform hazard estimates. Small differences in scale can correspond to significant variation of the duration of the nucleation phase, from seconds to years. This hypersensitivity to local fault zone factors has significant implications for premonitory signal detection. In fact, any extension of the nucleation phase where the fault undergoes stable sliding may be more readily detected using remote sensing techniques which can resolve surface displacements down to 2cm (Guerrieri et al., 2010).

Further, this hypersensitivity to initial conditions supports the idea that the fault-fluid system can be treated as a dynamical system, and is consistent with the theoretical analysis of said system (Anghel et al., 2004; Kim, 2017; Sobolev, 2011). The consideration of fault zone fluids and small-scale fracturing in simulations mediates an effective nonlinear relationship in fault frictional strength. Similar chaotic behaviour emerges in both experimental (Johnson et al., 2012) and seismological (Shelly, 2010) observations of nonlinear friction in fault systems.

Both Chapters 4 and 5 include simulations which predict long periods of aseismic slip on the order of months to years. Episodes of aseismic slip on fault zones has been observed on numerous occasions throughout the last decade, although they were generally not linked to the triggering of seismicity. Recently, large aseismic slip episodes have been identified immediately preceding the recent $M_w$ 9, 2011 Tohoku earthquake (Kato et al., 2012; Mavrommatis et al., 2014) and the $M_w$ 8.1, 2014 Iquique earthquake (Ruiz et al., 2017; Socquet et al., 2017). It is argued that these aseismic slip events, each lasting a few months, contributed to the triggering of

earthquakes and were related to their preparatory nucleation phase (Guglielmi et al., 2015).

The control exerted by the local factors over the interseismic period has significant implications for earthquake forecasting. On the other hand, estimates of the duration of the nucleation phase have implications for earthquake early warning systems. We believe that the aseismic creep predicted by our simulations (creep on fault patches > 30m) could be detectable well in advance of a significant seismic event, particularly by using geodetic data and new satellite remote sensing techniques which can resolve surface displacements down to 2cm (Guerrieri et al., 2010).

### 6.4 Future modelling

Simulations of the earthquake nucleation phase were able to resolve the duration of this phase to the order of a few seconds in several hundred-year simulations consistently, despite high instantaneous stiffness in the Jacobian matrix for the fault-fluid system (eigenvalues differing by a factor of 1011). This efficiency is possible by considering the permeability transitions associated with small scale fracturing, brittle or ductile mode of failure, and stable sliding on the fault as the discontinuous transitions of a non-smooth dynamical system and conducting simulations with an explicit singly diagonal implicit Runge-Kutta (ESDIRK) solver (for A-,L-,S-stability) and event detection.

While simulations presented in Chapter 4 accounted for the occurrence of brittle mode of failure, those in Chapter 5 did not due to the level of pore pressure considered in the damage zone reservoir. This was due to the preliminary parameter studies, in the pressure ranges likely to produce brittle failure, being significantly

more computationally intensive. The simulation time was approximately two orders of magnitude larger due to the gradual, phased development of brittle mode of failure, as opposed to the instant onset of ductile failure.

Future work may consider running higher resolution parameter studies in these brittle failure ranges, although production of more performant code might be necessary. The current simulation code performs the majority of its short timestep calculations immediately after large changes in permeability following brittle or ductile mode of failure. The subsequent pressure front might be resolved more efficiently with existing computational techniques (e.g. Weighted Essentially Non-Oscillatory Schemes, Liu, Osher, & Chan, 1994). Beyond this, performance analysis of simulations indicates that a similar amount of execution time is spent in the MATLAB ODE solver library itself as in the MATLAB numerical simulation script I have written. The MATLAB execution engine calls C++ code using just-in-time compilation. This code itself is heavily optimised but also general enough to be versatile. Given the already high level of code optimization present in MATLAB's built-in libraries it is likely that the only possibility for performance improvements, beyond one order of magnitude, likely lies in memory optimisation with respect to processor caches. The most promising way forward might be hand optimised C++ code with existing solver libraries (e.g. CVODE; Hindmarsh et al., 2005) to potentially entirely avoid cache thrashing (e.g. (Jin et al., 1998)).

### 6.5 Conclusions

The work set out in this thesis illustrates a range of seismic processes present in natural low porosity fault zones. These processes are examined in numerical simulations of with transport and failure properties constrained using triaxial loading experiments with fluid flow. The results of these simulations have broad

implications for fluid flow and earthquake nucleation in low porosity fault zones and are described below:

- Several local factors of fault zones exert a primary control on the diffusion of pore pressure into the fault core of low porosity faults. Hence, these factors control the duration of the interseismic period, the size of the rupture patch and the duration of the nucleation phase of an earthquake. Specifically, fault core width and the thickness and level of pore pressure in pressurised reservoirs in the damage zone abutting the fault zone all mediate the aforementioned effect.
- Small-scale fracturing in the fault core, accommodated by either brittle or ductile mode of failure, acts as a primary control of the duration of the interseismic period, the size of the rupture patch, and the duration of the nucleation phase of an earthquake. The magnitude of this effect was significant in simulations that also included realistic, complex models of fault zone architecture. Therefore, both must be considered to produce realistic results.
- Earthquake rupture patch development and nucleation is governed by highly nonlinear processes. The fault-fluid system exhibits hypersensitivity to initial conditions, consistent with a chaotic dynamical system. This hypersensitivity to initial conditions has large implications for earthquake forecasting, and premonitory signal detection, as small centimetre scale uncertainty in fault zone parameters controlled by local factors can have outsized effects on the duration of the interseismic period, rupture patch dimension and length of the nucleation phase.

that these aseismic slip events, lasting a few months, contribute to the triggering of earthquakes and are related to their preparatory nucleation phase (Guglielmi et al., 2015).

Overall, our results show that both the inclusion of realistic models of fault zone architecture and deformation-dependent permeability (brittle and ductile failure) control the size of the sliding patch (Fig. 4.7b) during earthquake nucleation and the duration of the nucleation phase (Fig. 4.7c).

The size of the failure patch during the nucleation phase is always larger when realistic models of fault zone architecture and deformation-dependent permeability are considered (Fig. 4.7b). Small scale fracturing by brittle failure, occurring for initially high pore pressures, provides the largest slipping patches (> 30 m in Fig. 4.7b). These results are of relevance when considering that technological improvements in signal/noise ratio and spatio-temporal resolution of geodetic data are lowering the detection thresholds for measurements of aseismic slip. In particular, the advent of new satellite radar missions now enables a systematic, global investigation of pre-seismic slip for the first time.

Our results show that the duration of the nucleation phase is significantly reduced from a few years to a few months at high values of initial pore pressure, when realistic models of fault zone architecture and deformation-dependent permeability are considered. ınterestingly, a few months is also the time scale of aseismic slip measured during the nucleation phase of some recent large earthquakes (Fig. 4.7c; $M_w$ 9, 2011 Tohoku and $M_w$ 8.1, 2014 Iquique earthquakes).To conclude, estimates of the duration of the nucleation

phase have implications for earthquake early warning systems. In fact, intermittent aseismic creep on fault patches > 30 m in diameter, over a period of few months, could be detectable well in advance of a significant seismic event, perhaps using geodetic data and new satellite remote sensing techniques.

# BIBLIOGRAPHY

*Bibliography*

# Bibliography

Ake, J., Mahrer, K., O ’connell, D., Block, L., 2005. Deep-Injection and Closely Monitored Induced Seismicity at Paradox Valley, Colorado. Bull. Seismol. Soc. Am. 95, 664–683. https://doi.org/10.1785/0120040072

Anghel, M., Ben-Zion, Y., Rico-Martinez, R., 2004. Dynamical System Analysis and Forecasting of Deformation Produced by an Earthquake Fault. Pure Appl. Geophys. PAGEOPH 161, 2023–2051. https://doi.org/10.1007/s00024-004-2547-9

Atkinson, G., Assatourians, K., Cheadle, B., Greig, W., 2015. Ground Motions from Three Recent Earthquakes in Western Alberta and Northeastern British Columbia and Their Implications for Induced-Seismicity Hazard in Eastern Regions. Seismol. Res. Lett. 86, 1022–1031. https://doi.org/10.1785/0220140195

Atkinson, G.M., Eaton, D.W., Ghofrani, H., Walker, D., Cheadle, B., Schultz, R., Shcherbakov, R., Tiampo, K., Gu, J., Harrington, R.M., Liu, Y., Van Der Baan, M., Kao, H., Baan, M. van der, Kao, H., 2016. Hydraulic Fracturing and Seismicity in the Western Canada Sedimentary Basin. Seismol. Res. Lett. 87, 631–647. https://doi.org/10.1785/0220150263

Bagterp Jørgensen, J., Rode Kristensen, M., 2018. A FAMILY OF ESDIRK INTEGRATION METHODS.

Baisch, S., Voros, R., Weidler, R., Wyborn, D., 2009. Investigation of Fault Mechanisms during Geothermal Reservoir Stimulation Experiments in

the Cooper Basin, Australia. Bull. Seismol. Soc. Am. 99, 148–158. https://doi.org/10.1785/0120080055

Bao, X., Eaton, D.W., 2016. Fault activation by hydraulic fracturing in western Canada. Science 354, 1406–1409. https://doi.org/10.1126/science.aag2583

Barberio, M.D., Barbieri, M., Billi, A., Doglioni, C., Petitta, M., 2017. Hydrogeochemical changes before and during the 2016 Amatrice-Norcia seismic sequence (central Italy). Sci. Rep. 7, 11735. https://doi.org/10.1038/s41598-017-11990-8

Barchi, M., 2002. Lithological and structural controls on the seismogenesis of the Umbria region: observations from seismic reflection profiles. Boll. della Soc. Geol. Ital. 121, 855–864.

Berryman, J.G., 1992. Effective stress for transport properties of inhomogeneous porous rock. J. Geophys. Res. 97. https://doi.org/10.1029/92jb01593

Bullock, R.J., De Paola, N., Holdsworth, R.E., Trabucho-Alexandre, J., 2014. Lithological controls on the deformation mechanisms operating within carbonate-hosted faults during the seismic cycle. J. Struct. Geol. 58, 22–42. https://doi.org/10.1016/j.jsg.2013.10.008

Burke, L., 2011. Carbon dioxide fluid-flow modeling and injectivity calculations. U.S. Geol. Surv. Sci. Investig. Rep. 2011 5083, 16.

Byerlee, J., 1978. Friction of rocks. Pure Appl. Geophys. PAGEOPH 116, 615–626. https://doi.org/10.1007/BF00876528

Byerlee, J.D., 1968. Brittle-Ductile Transition in Rocks. https://doi.org/10.1029/JB073i014p04741

Caine, J.S., Evans, J.P., Forster, C.B., Saul, J., City, S.L., Caine, J.S., Evans, J.P., Forster, C.B., 1996. Fault zone architecture and permeability structure. Geology 24, 1025. https://doi.org/10.1130/0091-7613(1996)024<1025:FZAAPS>2.3.CO;2

Caine, J.S., Forster, C.B., 1999. Fault Zone Architecture and Fluid Flow : Insights From Field Data and Numerical Modeling.

Campillo, M., Favreau, P., Ionescu, I.R., Voisin, C., 2001. On the effective friction law of a heterogeneous fault. J. Geophys. Res. 106, 16307. https://doi.org/10.1029/2000JB900467

Cappa, F., 2009. Modelling fluid transfer and slip in a fault zone when integrating heterogeneous hydromechanical characteristics in its internal structure. Geophys. J. Int. 178, 1357–1362. https://doi.org/10.1111/j.1365-246X.2009.04291.x

Cappa, F., Rutqvist, J., 2011a. Impact of CO 2 geological sequestration on the nucleation of earthquakes 38, 2–7. https://doi.org/10.1029/2011GL048487

Cappa, F., Rutqvist, J., 2011b. Modeling of coupled deformation and permeability evolution during fault reactivation induced by deep underground injection of CO2. Int. J. Greenh. Gas Control 5, 336–346. https://doi.org/10.1016/j.ijggc.2010.08.005

Cappa, F., Rutqvist, J., Yamamoto, K., 2009. Modeling crustal deformation

and rupture processes related to upwelling of deep $CO_2$ -rich fluids during the 1965–1967 Matsushiro earthquake swarm in Japan. J. Geophys. Res. 114, B10304. https://doi.org/10.1029/2009JB006398

Cappa, F.F., Rutqvist, J., 2012. Seismic rupture and ground accelerations induced by CO 2 injection in the shallow crust. Geophys. J. Int. 190, 1784–1789. https://doi.org/10.1111/j.1365-246X.2012.05606.x

Chester, F.M., Chester, J.S., 1998. Ultracataclasite structure and friction processes of the Punchbowl fault, San Andreas system, California, Tectonophysics. Elsevier. https://doi.org/10.1016/S0040-1951(98)00121-8

Chester, F.M., Chester, J.S., Kirschner, D.L., Schulz, S.E., Evans, J.P., 2004. 8. Structure of Large-Displacement, Strike-Slip Fault Zones in the Brittle Continental Crust, in: Karner, G.D., Taylor, B., Driscoll, N.W., Kohlstedt, D.L. (Eds.), Rheology and Deformation of the Lithosphere at Continental Margins. Columbia University Press, New York Chichester, West Sussex. https://doi.org/10.7312/karn12738-009

Chester, F.M., Evans, J.P., Biegel, R.L., 1993. Internal structure and weakening mechanisms of the San Andreas Fault. J. Geophys. Res. Solid Earth 98, 771–786. https://doi.org/10.1029/92JB01866

Chester, F.M., Logan, J.M., 1986. Implications for mechanical properties of brittle faults from observations of the Punchbowl fault zone, California. Pure Appl. Geophys. PAGEOPH 124, 79–106. https://doi.org/10.1007/BF00875720

Chiaraluce, L., Ellsworth, W.L., Chiarabba, C., Cocco, M., 2003. Imaging the

complexity of an active normal fault system: The 1997 Colfiorito (central Italy) case study. J. Geophys. Res. 108, 2294. https://doi.org/10.1029/2002JB002166

Chiodini, G., Cardellini, C., Amato, A., Boschi, E., Caliro, S., Frondini, F., Ventura, G., 2004. Carbon dioxide Earth degassing and seismogenesis in central and southern Italy. Geophys. Res. Lett. 31, n/a-n/a. https://doi.org/10.1029/2004GL019480

Chiodini, G., Cioni, R., 1989. Gas geobarometry for hydrothermal systems and its application to some Italian geothermal areas. Appl. Geochemistry 4, 465–472. https://doi.org/10.1016/0883-2927(89)90004-8

Clarke, H., Eisner, L., Styles, P., Turner, P., 2014. Felt seismicity associated with shale gas hydraulic fracturing: The first documented example in Europe. Geophys. Res. Lett. 41, 8308–8314. https://doi.org/10.1002/2014GL062047

Collettini, C., Barchi, M.R., 2002. A low-angle normal fault in the Umbria region (Central Italy): a mechanical model for the related microseismicity. Tectonophysics 359, 97–115. https://doi.org/10.1016/S0040-1951(02)00441-9

Collettini, C., Cardellini, C., Chiodini, G., De Paola, N., Holdsworth, R.E., Smith, S. a. F., 2008. Fault weakening due to CO2 degassing in the Northern Apennines: short- and long-term processes. Geol. Soc. London, Spec. Publ. 299, 175–194. https://doi.org/10.1144/SP299.11

Collettini, C., De Paola, N., Faulkner, D.R., 2009. Insights on the geometry and mechanics of the Umbria–Marche earthquakes (Central Italy) from

the integration of field and laboratory data. Tectonophysics 476, 99–109. https://doi.org/10.1016/j.tecto.2008.08.013

Cox, S.F., 2010. The application of failure mode diagrams for exploring the roles of fluid pressure and stress states in controlling styles of fracture-controlled permeability enhancement in faults and shear zones. Geofluids 10, 217–233. https://doi.org/10.1111/j.1468-8123.2010.00281.x

Cox, S.F., 1995. Faulting processes at high fluid pressures: An example of fault valve behavior from the Wattle Gully Fault, Victoria, Australia. J. Geophys. Res. Solid Earth 100, 12841–12859. https://doi.org/10.1029/95JB00915

Cox, S.F., Etheridge, M.A., Wall, V.J., 1987. The role of fluids in syntectonic mass transport, and the localization of metamorphic vein-type ore deposits. Ore Geol. Rev. Elsevier Sci. Publ. B.V 2, 65–86. https://doi.org/10.1016/0169-1368(87)90024-2

Dahlquist, G.G., 1963. A special stability problem for linear multistep methods. BIT 3, 27–43. https://doi.org/10.1007/BF01963532

Davies, J.P.P., Davies, D.K.D.K., Davies, D.K.D.K., 2001. Stress-Dependent Permeability : Characterization and Modeling. SPE J. 6, 224–235. https://doi.org/10.2118/71750-PA

Davies, R., Foulger, G., Bindley, A., Styles, P., 2013. Induced seismicity and hydraulic fracturing for the recovery of hydrocarbons. Mar. Pet. Geol. 45, 171–185. https://doi.org/10.1016/j.marpetgeo.2013.03.016

De Paola, N., Collettini, C., Faulkner, D.R., Trippetta, F., 2008. Fault zone architecture and deformation processes within evaporitic rocks in the upper crust. Tectonics 27, 1–21. https://doi.org/10.1029/2007TC002230

De Paola, N., Faulkner, D.R., Collettini, C., 2009. Brittle versus ductile deformation as the main control on the transport properties of low-porosity anhydrite rocks. J. Geophys. Res. Solid Earth 114. https://doi.org/10.1029/2008JB005967

De Pater, C.J., Baisch, S., 2011. Geomechanical Study of Bowland Shale Seismicity Synthesis Report.

Detournay, E., Cheng, A.H.-D.A., 1993. Fundamentals of Poroelasticity. Compr. Rock Eng. Princ. Pract. Proj. II, 113–171. https://doi.org/10.1016/0148-9062(94)90606-8

Di Luccio, F., Ventura, G., Di Giovambattista, R., Piscini, A., Cinti, F.R., 2010. Normal faults and thrusts reactivated by deep fluids: The 6 April 2009 Mw 6.3 L'Aquila earthquake, central Italy. J. Geophys. Res. 115, B06315. https://doi.org/10.1029/2009JB007190

Dicelis, G., Assumpção, M., Prado, R.L., Agurto-Detzel, H., Barbosa, J.R., 2017. Improving the characterization of the seismic source in Bebedouro, Paraná Basin, Brazil: Further evidence of seismicity triggered by hydraulic stimulation in water wells. Geophys. J. Int. 210, 594–608. https://doi.org/10.1093/gji/ggx180

Dieterich, J.H., 1992. Earthquake nucleation on faults with rate- and statedependent friction. Tectonophysics.

Dieterich, J.H., 1979. Modeling of rock friction: 1. Experimental results and constitutive equations. J. Geophys. Res. 84, 2161. https://doi.org/10.1029/JB084iB05p02161

Ellsworth, W.L., 2013. Injection-Induced Earthquakes. Science (80-. ). 341, 1225942–1225942. https://doi.org/10.1126/science.1225942

Elsworth, D., Spiers, C.J., Niemeijer, A.R., 2016. Understanding induced seismicity. Science (80-. ). 354, 1380–1381. https://doi.org/10.1126/science.aal2584

Evans, J.P., Forster, C.B., Goddard, J. V., 1997. Permeability of fault-related rocks, and implications for hydraulic structure of fault zones. J. Struct. Geol. 19, 1393–1404. https://doi.org/10.1016/S0191-8141(97)00057-6

Farahbod, A.M., Kao, H., Cassidy, J.F., Walker, D., 2015a. How did hydraulic-fracturing operations in the Horn River Basin change seismicity patterns in northeastern British Columbia, Canada? Lead. Edge 34, 658–663. https://doi.org/10.1190/tle34060658.1

Farahbod, A.M., Kao, H., Walker, D.M., Cassidy, J.F., 2015b. Investigation of regional seismicity before and after hydraulic fracturing in the Horn River Basin, northeast British Columbia. Can. J. Earth Sci. 52, 112–122. https://doi.org/10.1139/cjes-2014-0162

Faulkner, D.., Lewis, A.., Rutter, E.., 2003. On the internal structure and mechanics of large strike-slip fault zones: field observations of the Carboneras fault in southeastern Spain. Tectonophysics 367, 235–251. https://doi.org/10.1016/S0040-1951(03)00134-3

Faulkner, D.R., 2004. A model for the variation in permeability of clay-bearing fault gouge with depth in the brittle crust. Geophys. Res. Lett. 31, L19611. https://doi.org/10.1029/2004GL020736

Faulkner, D.R., Rutter, E.H., 2003. The effect of temperature, the nature of the pore fluid, and subyield differential stress on the permeability of phyllosilicate-rich fault gouge. J. Geophys. Res. Solid Earth 108. https://doi.org/10.1029/2001JB001581

Faulkner, D.R., Rutter, E.H., 2001. Can the maintenance of overpressured fluids in large strike-slip fault zones explain their apparent weakness? Geology 29, 503–506. https://doi.org/10.1130/0091-7613(2001)029<0503:CTMOOF>2.0.CO;2

Faulkner, D.R., Rutter, E.H., 2000. Comparisons of water and argon permeability in natural clay-bearing fault gouge under high pressure at 20°C. J. Geophys. Res. Solid Earth 105, 16415–16426. https://doi.org/10.1029/2000JB900134

Faulkner, D.R.R., Jackson, C.A.L.A.L., Lunn, R.J.J., Schlische, R.W.W., Shipton, Z.K.K., Wibberley, C.A.J.A.J., Withjack, M.O.O., 2010. A review of recent developments concerning the structure, mechanics and fluid flow properties of fault zones. J. Struct. Geol. 32, 1557–1575. https://doi.org/10.1016/j.jsg.2010.06.009

Fischer, G.J., 1992. Chapter 8 The Determination of Permeability and Storage Capacity: Pore Pressure Oscillation Method. Academic Press. https://doi.org/10.1016/S0074-6142(08)62823-5

Folger, P., Tiemann, M., 2015. Human-Induced Earthquakes from Deep-Well

Injection: A Brief Overview 1–26.

Friberg, P.A., Besana-Ostman, G.M., Dricker, I., 2014. Characterization of an Earthquake Sequence Triggered by Hydraulic Fracturing in Harrison County, Ohio. Seismol. Res. Lett. 85, 1295–1307. https://doi.org/10.1785/0220140127

Frohlich, C., 2012. Two-year survey comparing earthquake activity and injection-well locations in the Barnett Shale, Texas. Proc. Natl. Acad. Sci. U. S. A. 109, 13934–8. https://doi.org/10.1073/pnas.1207728109

Frohlich, C., Brunt, M., 2013. Two-year survey of earthquakes and injection/production wells in the Eagle Ford Shale, Texas, prior to the MW4.8 20 October 2011 earthquake. Earth Planet. Sci. Lett. 379, 56–63. https://doi.org/10.1016/j.epsl.2013.07.025

Gan, W., Frohlich, C., Forsyth, D.W., 2013. Gas injection may have triggered earthquakes in the Cogdell oil field, Texas. https://doi.org/10.1073/pnas.1311316110

Goudarzi, S., Mathias, S.A., Gluyas, J.G., 2016. Simulation of Three-Component Two-Phase Flow in Porous Media Using Method of Lines. Transp. Porous Media 112, 1–19. https://doi.org/10.1007/s11242-016-0639-5

Griffith, A., 1924. The theory of rupture. Proc., Ist., Int., Congr., Appl., Mech. Biereno, C.B. Burgers, J.M(eds). Delft Tech. Boekhand. en Druk. J. Waltman Jr. 54–63.

Griffiths, D. V., 1990. Failure Criteria Interpretation Based on Mohr-

Coulomb Friction. J. Geotech. Eng. 116, 986–999. https://doi.org/10.1061/(ASCE)0733-9410(1990)116:6(986)

Guerrieri, L., Baer, G., Hamiel, Y., Amit, R., Blumetti, A.M., Comerci, V., Di Manna, P., Michetti, A.M., Salamon, A., Mushkin, A., Sileo, G., Vittori, E., 2010. InSAR data as a field guide for mapping minor earthquake surface ruptures: Ground displacements along the Paganica Fault during the 6 April 2009 L'Aquila earthquake. J. Geophys. Res. 115, B12331. https://doi.org/10.1029/2010JB007579

Guglielmi, Y., Cappa, F., Avouac, J., Henry, P., Elsworth, D., 2015. Seismicity triggered by fluid injection – induced aseismic slip. Science (80-. ). 348, 1224–1227. https://doi.org/10.1126/science.aab0476

Hairer, E., Wanner, G., 1996. Stability Analysis for Explicit RK Methods. pp. 15–39. https://doi.org/10.1007/978-3-642-05221-7_2

Hangx;, Spiers, C.J., Peach, C.J., 2010. Mechanical behavior of anhydrite caprock and implications for CO 2 sealing capacity. J. Geophys. Res. 115, B07402. https://doi.org/10.1029/2009JB006954

Hangx, S.J.T., Spiers, C.J., Peach, C.J., 2010. The effect of deformation on permeability development in anhydrite and implications for caprock integrity during geological storage of CO2. Geofluids 10, 369–387. https://doi.org/10.1111/j.1468-8123.2010.00299.x

Hickman, S., Sibson, R., Bruhn, R., 1995. Introduction to Special Section: Mechanical Involvement of Fluids in Faulting. J. Geophys. Res. Solid Earth 100, 12831–12840. https://doi.org/10.1029/95JB01121

Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E., Woodward, C.S., 2005. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers.

Holland, A.A., 2013. Earthquakes triggered by hydraulic fracturing in south-central Oklahoma. Bull. Seismol. Soc. Am. 103, 1784–1792. https://doi.org/10.1785/0120120109

Hornbach, M.J., Jones, M., Scales, M., DeShon, H.R., Magnani, M.B., Frohlich, C., Stump, B., Hayward, C., Layton, M., 2016. Ellenburger wastewater injection and seismicity in North Texas. Phys. Earth Planet. Inter. 261, 54–68. https://doi.org/10.1016/J.PEPI.2016.06.012

Hosea, M.E., Shampine, L.F., 1996. Analysis and implementation of TR-BDF2. Appl. Numer. Math. 20, 21–37. https://doi.org/10.1016/0168-9274(95)00115-8

Hsiung, S.M., Chowdhury, a H., Nataraja, M.S., 2005. Numerical simulation of thermal – mechanical processes observed at the Drift-Scale Heater Test at Yucca Mountain , Nevada , USA. Rock Mech. 42, 652–666. https://doi.org/10.1016/j.ijrmms.2005.03.006

Jin, G., Li, Z., Chen, F., 1998. An efficient solution to the cache thrashing problem caused by true data sharing. IEEE Trans. Comput. 47, 527–543. https://doi.org/10.1109/12.677228

Johnson, P.A., Carpenter, B., Knuth, M., Kaproth, B.M., Le Bas, P.-Y., Daub, E.G., Marone, C., 2012. Nonlinear dynamical triggering of slow slip on simulated earthquake faults with implications to Earth. J. Geophys. Res. Solid Earth 117, n/a-n/a. https://doi.org/10.1029/2011JB008594

Kato, A., Obara, K., Igarashi, T., Tsuruoka, H., Nakagawa, S., Hirata, N., 2012. Propagation of Slow Slip Leading Up to the 2011 W w 9.0 Tohoku-Oki Earthquake. Source Sci. New Ser. 335, 705–708. https://doi.org/10.1126/science.l213778

Keranen, K.M., Savage, H.M., Abers, G.A., Cochran, E.S., 2013. Potentially induced earthquakes in Oklahoma, USA: Links between wastewater injection and the 2011 Mw 5.7 earthquake sequence. Geology 41, 699–702. https://doi.org/10.1130/G34045.1

Keranen, K.M., Weingarten, M., Abers, G.A., Bekins, B.A., Ge, S., 2014. Sharp increase in central Oklahoma seismicity since 2008 induced by massive wastewater injection. Science (80-. ). 345, 448–451. https://doi.org/10.1126/SCIENCE.1255802

Keulen, N., Stünitz, H., Heilbronner, R., 2008. Healing microstructures of experimental and natural fault gouge. J. Geophys. Res. 113, B06205. https://doi.org/10.1029/2007JB005039

Kim, M.H., 2017. New dynamical systems modeling helps explain mega-earthquakes. Scilight 2017, 180004. https://doi.org/10.1063/1.5009894

Kim, W.-Y., 2013. Induced seismicity associated with fluid injection into a deep well in Youngstown, Ohio. J. Geophys. Res. Solid Earth 118, 3506–3518. https://doi.org/10.1002/jgrb.50247

Kristensen, M.R., Jørgensen, J.B., Thomsen, P.G., Jørgensen, S.B., 2004. An ESDIRK method with sensitivity analysis capabilities. Comput. Chem. Eng. 28, 2695–2707. https://doi.org/10.1016/J.COMPCHEMENG.2004.08.004

Lambert, J.D. (John D., D., J., 1991. Numerical methods for ordinary differential systems : the initial value problem. Wiley.

Leclère, H., Cappa, F., Faulkner, D., Fabbri, O., Armitage, P., Blake, O., 2015a. Development and maintenance of fluid overpressures in crustal fault zones by elastic compaction and implications for earthquake swarms. J. Geophys. Res. Solid Earth 120, 4450–4473. https://doi.org/10.1002/2014JB011759

Leclère, H., Cappa, F., Faulkner, D., Fabbri, O., Armitage, P., Blake, O., Zhang, L., Carpenter, B.M., Ikari, M.J., Marone, C., Zhang, L., Carpenter, B.M., Ikari, M.J., Marone, C., Leclère, H., Cappa, F., Faulkner, D., Fabbri, O., Armitage, P., Blake, O., 2015b. Journal of Geophysical Research : Solid Earth. J. Geophys. Res. Solid Earth 4450–4473. https://doi.org/10.1002/2014JB011759.Received

Leclère, H., Fabbri, O., Daniel, G., Cappa, F., 2012. Reactivation of a strike-slip fault by fluid overpressuring in the southwestern French-Italian Alps. Geophys. J. Int. 189, 29–37. https://doi.org/10.1111/j.1365-246X.2011.05345.x

Lei, X., Huang, D., Su, J., Jiang, G., Wang, X., Wang, H., Guo, X., Fu, H., 2017. Fault reactivation and earthquakes with magnitudes of up to Mw4.7 induced by shale-gas hydraulic fracturing in Sichuan Basin, China. Sci. Rep. 7, 7971. https://doi.org/10.1038/s41598-017-08557-y

Liu, X.-D., Osher, S., Chan, T., 1994. Weighted Essentially Non-oscillatory Schemes. J. Comput. Phys. 115, 200–212. https://doi.org/10.1006/JCPH.1994.1187

Lockner, D.A., Beeler, N.M., 1999. Premonitory slip and tidal triggering of earthquakes, JOURNAL OF GEOPHYSICAL RESEARCH. https://doi.org/10.1029/1999JB900205

Lucente, F., De Gori, P., Margheriti, L., Piccinini, D., Di Bona, M., Chiarabba, C., Piana Agostinetti, N., 2010a. Temporal variation of seismic velocity and anisotropy before the 2009 M W 6.3 L'Aquila earthquake, Italy. Geology 38, 1015–1018. https://doi.org/10.1130/G31463.1

Lucente, F., Margheriti, L., Chiarabba, C., Piana, N., Francesco, A., Lucente, P., De Gori, P., Piccinini, D., Di Bona, M., 2010b. Seismic Anisotropy beneath the Arabia-Eurasia collision zone: the major thrust-and-fold belts of Zagros and Alborz and the Iranian plateau View project ITALIAN SEISMIC BULLETIN View project. https://doi.org/10.1130/G31463.1

Mahesh, P., Gupta, S., Rai, S.S., Sarma, P.R., 2012. Fluid driven earthquakes in the Chamoli Region, Garhwal Himalaya: evidence from local earthquake tomography. Geophys. J. Int. 191, no-no. https://doi.org/10.1111/j.1365-246X.2012.05672.x

Marone, C., 1998. Laboratory-Derived Friction Laws and Their Application to Seismic Faulting. Annu. Rev. Earth Planet. Sci. 26, 643–696. https://doi.org/10.1146/annurev.earth.26.1.643

Mathias, S.A., Hardisty, P.E., Trudell, M.R., Zimmerman, R.W., 2009. Screening and selection of sites for CO2 sequestration based on pressure buildup. Int. J. Greenh. Gas Control 3, 577–585.

https://doi.org/10.1016/J.IJGGC.2009.05.002

Mavrommatis, A.P., Segall, P., Johnson, K.M., 2014. A decadal-scale deformation transient prior to the 2011 $M_w$ 9.0 Tohoku-oki earthquake. Geophys. Res. Lett. 41, 4486–4494. https://doi.org/10.1002/2014GL060139

Maxwell, S.C., Urbancic, T.I., Steinsberger, N., Zinno, R., 2002. Microseismic Imaging of Hydraulic Fracture Complexity in the Barnett Shale, in: SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers. https://doi.org/10.2118/77440-MS

Mazzoldi, A., Rinaldi, A.P., Borgia, A., Rutqvist, J., 2012. Induced seismicity within geological carbon sequestration projects: Maximum earthquake magnitude and leakage potential from undetected faults. Int. J. Greenh. Gas Control 10, 434–442. https://doi.org/10.1016/j.ijggc.2012.07.012

McGarr, A., 2014. Maximum magnitude earthquakes induced by fluid injection. J. Geophys. Res. Solid Earth 119, 1008–1019. https://doi.org/10.1002/2013JB010597

McGarr, A., Bekins, B., Burkardt, N., Dewey, J., Earle, P., Ellsworth, W., Ge, S., Hickman, S., Holland, A., Majer, E., Rubinstein, J., Sheehan, A., 2015. Coping with earthquakes induced by fluid injection. Science (80-. ). 347, 830–831. https://doi.org/10.1126/science.aaa0494

Miller, S. a, Collettini, C., Chiaraluce, L., Cocco, M., Barchi, M., Kaus, B.J.P., 2004. Aftershocks driven by a high-pressure CO2 source at depth. Nature 427, 724–727. https://doi.org/10.1038/nature02251

Miller, S.A., 1996. Fluid-mediated influence of adjacent thrusting on the seismic cycle at Parkfield. Nature 382, 799–802. https://doi.org/10.1038/382799a0

Mirabella, F., Barchi, M., Lupattelli, A., Stucchi, E., Ciaccio, M.G., 2008. Insights on the seismogenic layer thickness from the upper crust structure of the Umbria-Marche Apennines (central Italy). Tectonics 27, 1–15. https://doi.org/10.1029/2007TC002134

Mitchell, T.M., Faulkner, D.R., 2008. Experimental measurements of permeability evolution during triaxial compression of initially intact crystalline rocks and implications for fluid flow in fault zones. J. Geophys. Res. Solid Earth 113, 1–16. https://doi.org/10.1029/2008JB005588

Mizoguchi, K., Hirose, T., Shimamoto, T., Fukuyama, E., 2008. Internal structure and permeability of the Nojima fault, southwest Japan. J. Struct. Geol. 30, 513–524. https://doi.org/10.1016/j.jsg.2007.12.002

Morrow, C.A., Lockner, D.A., 1997. Permeability and porosity of the Illinois UPH 3 drillhole granite and a comparison with other deep drillhole rocks. J. Geophys. Res. Solid Earth 102, 3067–3075. https://doi.org/10.1029/96JB03178

Morrow, C.A., Lockner, D.A., 1994. Permeability differences between surface-derived and deep drillhole core samples. Geophys. Res. Lett. 21, 2151–2154. https://doi.org/10.1029/94GL01936

Morrow, C.A., Shi, L.Q., Byerlee, J.D., 1984. Permeability of fault gouge under confining pressure and shear stress. J. Geophys. Res. Solid Earth

89, 3193–3200. https://doi.org/10.1029/JB089iB05p03193

Nakatani, M., Scholz, C.H., 2004. Frictional healing of quartz gouge under hydrothermal conditions: 1. Experimental evidence for solution transfer healing mechanism. J. Geophys. Res. Solid Earth 109. https://doi.org/10.1029/2001JB001522

Niemeijer, A., Marone, C., Elsworth, D., 2008. Healing of simulated fault gouges aided by pressure solution: Results from rock analogue experiments 113, B04204. https://doi.org/10.1029/2007JB005376

Noir, J., Jacques, E., Békri, S., Adler, P.M., Tapponnier, P., King, G.C.P., 1997. Fluid flow triggered migration of events in the 1989 Dobi Earthquake sequence of central Afar. Geophys. Res. Lett. 24, 2335–2338. https://doi.org/10.1029/97GL02182

Nur, A., Booker, J.R., 1972. Aftershocks caused by pore fluid flow? Science 175, 885–7. https://doi.org/10.1126/science.175.4024.885

Parotidis, M., Rothert, E., Shapiro, S.A., 2003. Pore-pressure diffusion: A possible triggering mechanism for the earthquake swarms 2000 in Vogtland/NW-Bohemia, central Europe. Geophys. Res. Lett. 30, n/a-n/a. https://doi.org/10.1029/2003GL018110

Paterson, M.S., Wong, T.-F., 2005. Experimental Rock Deformation - The Brittle Field. Exp. Rock Deform. - Brittle Field, by M.S. Paterson T.-F. Wong. X, 348 p. 87 illus. 2nd Ed. 3-540-24023-3. Berlin Springer, 2005. 87.

Peach, C.J., Spiers, C.J., 1996. Influence of crystal plastic deformation on

dilatancy and permeability development in synthetic salt rock. Tectonophysics 256, 101–128. https://doi.org/10.1016/0040-1951(95)00170-0

Porreca, M., Minelli, G., Ercoli, M., Brobia, A., Mancinelli, P., Cruciani, F., Giorgetti, C., Carboni, F., Mirabella, F., Cavinato, G., Cannata, A., Pauselli, C., Barchi, M.R., 2018. Seismic Reflection Profiles and Subsurface Geology of the Area Interested by the 2016-2017 Earthquake Sequence (Central Italy). Tectonics 37, 1116–1137. https://doi.org/10.1002/2017TC004915

Rice, J.R., 1992. Fault Stress States, Pore Pressure Distributions, and the Weakness of the San Andreas Fault, Fault Mechanics and Transport Properties in Rocks. Academic Press. https://doi.org/10.1016/S0074-6142(08)62835-1

Rice, J.R., Ruina, a. L., 1983. Stability of Steady Frictional Slipping. J. Appl. Mech. 50, 343. https://doi.org/10.1115/1.3167042

Rinaldi, A.P., Jeanne, P., Rutqvist, J., Cappa, F., Guglielmi, Y., 2014a. Effects of fault-zone architecture on earthquake magnitude and gas leakage related to $CO_2$ injection in a multi-layered sedimentary system. Greenh. Gases Sci. Technol. 4, 99–120. https://doi.org/10.1002/ghg.1403

Rinaldi, A.P., Rutqvist, J., Cappa, F., 2014b. Geomechanical effects on CO2 leakage through fault zones during large-scale underground injection. Int. J. Greenh. Gas Control 20, 117–131. https://doi.org/10.1016/J.IJGGC.2013.11.001

Robertson, B.E.C., Robie, R.A., Books, K.G., 1958. Physical properties of

salt, anhydrite, and gypsum -- preliminary report. United States Dep. Inter. Trace Elem. Memo. Rep. 1048.

Rowland, J. V., Sibson, R.H., 2004. Structural controls on hydrothermal flow in a segmented rift system, Taupo Volcanic Zone, New Zealand. Geofluids 4, 259–283. https://doi.org/10.1111/j.1468-8123.2004.00091.x

Ruiz, S., Aden-Antoniow, F., Baez, J.C., Otarola, C., Potin, B., del Campo, F., Poli, P., Flores, C., Satriano, C., Leyton, F., Madariaga, R., Bernard, P., 2017. Nucleation Phase and Dynamic Inversion of the $M_w$ 6.9 Valparaíso 2017 Earthquake in Central Chile. Geophys. Res. Lett. 44, 10,290-10,297. https://doi.org/10.1002/2017GL075675

Rutledge, J.T., Phillips, W.S., 2003. Hydraulic stimulation of natural fractures as revealed by induced microearthquakes, Carthage Cotton Valley gas field, east Texas. GEOPHYSICS 68, 441–452. https://doi.org/10.1190/1.1567214

Rutledge, J.T., Phillips, W.S., Mayerhofer, M.J., 2004. Faulting Induced by Forced Fluid Injection and Fluid Flow Forced by Faulting: An Interpretation of Hydraulic-Fracture Microseismicity, Carthage Cotton Valley Gas Field, Texas. Bull. Seismol. Soc. Am. 94, 1817–1830.

Rutqvist, J., 2012. The Geomechanics of CO2 Storage in Deep Sedimentary Formations. Geotech. Geol. Eng. 30, 525–551. https://doi.org/10.1007/s10706-011-9491-0

Rutqvist, J., Birkholzer, J., Cappa, F., Tsang, C.F., 2007. Estimating maximum sustainable injection pressure during geological sequestration of CO2 using coupled fluid flow and geomechanical fault-slip analysis.

Energy Convers. Manag. 48, 1798–1807. https://doi.org/10.1016/j.enconman.2007.01.021

Rutqvist, J., Cappa, F., Mazzoldi, A., Rinaldi, A., 2013a. Geomechanical Modeling of Fault Responses and the Potential for Notable Seismic Events During Underground CO2 Injection. Energy Procedia 37, 4774–4784. https://doi.org/10.1016/j.egypro.2013.06.387

Rutqvist, J., Rinaldi, A.P., Cappa, F., Jeanne, P., Mazzoldi, A., Urpi, L., Guglielmi, Y., Vilarrasa, V., 2016. Fault activation and induced seismicity in geological carbon storage – Lessons learned from recent modeling studies. J. Rock Mech. Geotech. Eng. 8, 789–804. https://doi.org/10.1016/J.JRMGE.2016.09.001

Rutqvist, J., Rinaldi, A.P., Cappa, F., Moridis, G.J., 2013b. Modeling of fault reactivation and induced seismicity during hydraulic fracturing of shale-gas reservoirs. J. Pet. Sci. Eng. 107, 31–44. https://doi.org/10.1016/j.petrol.2013.04.023

Rutqvist, J., Rinaldi, A.P., Cappa, F.F., Moridis, G.J., 2015. Modeling of fault activation and seismicity by injection directly into a fault zone associated with hydraulic fracturing of shale-gas reservoirs. J. Pet. Sci. Eng. 127, 377–386. https://doi.org/10.1016/j.petrol.2015.01.019

Rutqvist, J., Vasco, D.W., Myer, L., 2009. Coupled reservoir-geomechanical analysis of CO2 injection at In Salah, Algeria. Energy Procedia 1, 1847–1854. https://doi.org/10.1016/j.egypro.2009.01.241

Rutqvist, J., Wu, Y.-S., Tsang, C.-F., Bodvarsson, G., 2002. A modeling approach for analysis of coupled multiphase fluid flow, heat transfer, and

deformation in fractured porous rock. Int. J. Rock Mech. Min. Sci. 39, 429–442. https://doi.org/10.1016/S1365-1609(02)00022-9

Rutter, E.H., 1972. The effects of strain-rate changes on the strength and ductility of Solenhofen limestone at low temperatures and confining pressures. Int. J. Rock Mech. Min. Sci. Geomech. Abstr. 9, 183–189. https://doi.org/10.1016/0148-9062(72)90020-4

Rutter, E.H., Maddock, R.H., Hall, S.H., White, S.H., 1986. Comparative microstructures of natural and experimentally produced clay-bearing fault gouges. Pure Appl. Geophys. PAGEOPH 124, 3–30. https://doi.org/10.1007/BF00875717

Scholz, C.H., 2019. The Mechanics of Earthquakes and Faulting. Cambridge University Press. https://doi.org/10.1017/9781316681473

Scholz, C.H., 1998. Earthquakes and friction laws. Nature 391, 37–42. https://doi.org/10.1038/34097

Scholz, C.H., 1988. The critical slip distance for seismic faulting. Nature 336, 761–763. https://doi.org/10.1038/336761a0

Schultz, R., Stern, V., Novakovic, M., Atkinson, G., Gu, Y.J., 2015. Hydraulic fracturing and the Crooked Lake Sequences: Insights gleaned from regional seismic networks. Geophys. Res. Lett. 42, 2750–2758. https://doi.org/10.1002/2015GL063455

Scuderi, M.M., Niemeijer, A.R., Collettini, C., Marone, C., 2013. Frictional properties and slip stability of active faults within carbonate-evaporite sequences: The role of dolomite and anhydrite. Earth Planet. Sci. Lett.

369–370, 220–232. https://doi.org/10.1016/j.epsl.2013.03.024

Seront, B., Wong, T.F., Caine, J.S., Forster, C.B., Bruhn, R.L., Fredrich, J.T., Seront, Bernard ; Wong, Teng-Fong ; Caine, Jonathan S. ; Forster, Craig B. ; Bruhn, Ronald L. ; Fredrich, J.T., Seront, B., Wong, T.F., Caine, J.S., Forster, C.B., Bruhn, R.L., Fredrich, J.T., 1998. Laboratory characterization of hydromechanical properties of a seismogenic normal fault system. J. Struct. Geol. 20, 865–881. https://doi.org/10.1016/S0191-8141(98)00023-6

Shampine, L.F., Reichelt, M.W., 1997. The MATLAB ODE Suite. SIAM J. Sci. Comput. 18, 1–22. https://doi.org/10.1137/S1064827594276424

Shapiro, S.A.A., Dinske, C., 2009. Fluid-induced seismicity: Pressure diffusion and hydraulic fracturing. Geophys. Prospect. 57, 301–310. https://doi.org/10.1111/j.1365-2478.2008.00770.x

Shelly, D.R., 2010. Periodic, Chaotic, and Doubled Earthquake Recurrence Intervals on the Deep San Andreas Fault. Science (80-. ). 328, 1385–1388. https://doi.org/10.1126/science.1189741

Sibson, R.H., 2000. Fluid involvement in normal faulting 29.

Sibson, R.H., 1996. Structural permeability of fluid-driven fault-fracture meshes. J. Struct. Geol. 18, 1031–1042. https://doi.org/10.1016/0191-8141(96)00032-6

Sibson, R.H., 1990. Conditions for fault-valve behaviour. Geol. Soc. London, Spec. Publ. 54, 15–28. https://doi.org/10.1144/GSL.SP.1990.054.01.02

Sibson, R.H., 1977. Fault rocks and fault mechanisms. J. Geol. Soc. London.

133, 191–213. https://doi.org/10.1144/gsjgs.133.3.0191

Sibson, R.H.H., 1992. Implications of fault-valve behaviour for rupture nucleation and recurrence. Tectonophysics 211, 283–293. https://doi.org/10.1016/0040-1951(92)90065-E

Silin, D., Korneev, V., Goloshubin, G., 2003. Pressure diffusion waves in porous media, Lawrence Berkeley National Laboratory.

Skoumal, R.J., Brudzinski, M.R., Currie, B.S., 2015. Earthquakes Induced by Hydraulic Fracturing in Poland Township, Ohio. Bull. Seismol. Soc. Am. 105, 189–197. https://doi.org/10.1785/0120140168

Sobolev, G.A., 2011. Natural Hazards and Earth System Sciences Seismicity dynamics and earthquake predictability. Hazards Earth Syst. Sci 11, 445–458. https://doi.org/10.5194/nhess-11-445-2011

Socquet, A., Valdes, J.P., Jara, J., Cotton, F., Walpersdorf, A., Cotte, N., Specht, S., Ortega-Culaciati, F., Carrizo, D., Norabuena, E., 2017. An 8 month slow slip event triggers progressive nucleation of the 2014 Chile megathrust. Geophys. Res. Lett. 44, 4046–4053. https://doi.org/10.1002/2017GL073023

Sumy, D.F., Cochran, E.S., Keranen, K.M., Wei, M., Abers, G. a., 2014. Observations of static Coulomb stress triggering of the November 2011 M 5.7 Oklahoma earthquake sequence. J. Geophys. Res. Solid Earth 119, 1904–1923. https://doi.org/10.1002/2013JB010612.Received

Terakawa, T., Hashimoto, C., Matsu'ura, M., 2013. Changes in seismic activity following the 2011 Tohoku-oki earthquake: Effects of pore fluid

pressure. Earth Planet. Sci. Lett. 365, 17–24. https://doi.org/10.1016/j.epsl.2013.01.017

Terzaghi, K., 1963. The Shearing Resistance of Saturated Soils. Proc. First Int. Conf. Soil Mech. 1, 54–56.

Thauvin, F., Mohanty, K.K., 1998. Network Modeling of Non-Darcy Flow Through Porous Media. Transp. Porous Media 31, 19–37. https://doi.org/10.1023/A:1006558926606

Townend, J., Zoback, M.D., 2000. How faulting keeps the crust strong. Geology 28, 399. https://doi.org/10.1130/0091-7613(2000)28<399:HFKTCS>2.0.CO;2

Trippetta, F., Collettini, C., Barchi, M.R., Lupattelli, A., Mirabella, F., 2013. A multidisciplinary study of a natural example of a CO2 geological reservoir in central Italy. Int. J. Greenh. Gas Control 12, 72–83. https://doi.org/10.1016/J.IJGGC.2012.11.010

Uenishi, K., Rice, J.R., 2003. Universal nucleation length for slip-weakening rupture instability under nonuniform fault loading. J. Geophys. Res. B Solid Earth 108. https://doi.org/10.1029/2001JB001681

Vermylen, J., Zoback, M.D., 2011. Hydraulic fracturing, microseismic magnitudes, and stress evolution in the Barnett Shale, Texas, USA. SPE Hydraul. Fract. Technol. … SPE 140507. https://doi.org/10.2118/140507-MS

Weeks, J.D., Tullis, T.E., 1985. Frictional sliding of dolomite: A variation in constitutive behavior. J. Geophys. Res. 90, 7821.

https://doi.org/10.1029/JB090iB09p07821

Weingarten, M., Ge, S., Godt, J.W., Bekins, B.A., Rubinstein, J.L., 2015. INDUCED SEISMICITY. High-rate injection is associated with the increase in U.S. mid-continent seismicity. Science 348, 1336–40. https://doi.org/10.1126/science.aab1345

Wibberley, C.A.J., Shimamoto, T., 2002. Internal structure and permeability of major strike-slip fault zones: The Median Tectonic Line in Mie Prefecture, Southwest Japan. J. Struct. Geol. 25, 59–78. https://doi.org/10.1016/S0191-8141(02)00014-7

Wong, T., Zhu, W., Wong, T., 1997. The transition from brittle faulting to cataclastic flow: Permeability evolution. J. Geophys. Res. Solid Earth 102, 3027–3041. https://doi.org/10.1029/96JB03282

Yasuhara, H., Marone, C., Elsworth, D., 2005. Fault zone restrengthening and frictional healing: The role of pressure solution. J. Geophys. Res 110. https://doi.org/10.1029/2004JB003327

Yoshida, K., Hasegawa, A., Yoshida, T., 2016. Temporal variation of frictional strength in an earthquake swarm in NE Japan caused by fluid migration. J. Geophys. Res. Solid Earth 121, 5953–5965. https://doi.org/10.1002/2016JB013022

Yoshida, N., Tsukahara, H., Okusawa, T., 2003. Andesitic Magmatic Water Which Generated Matsushiro Earthquake Swarm And S Wave Reflector. Am. Geophys. Union, Fall Meet. 2003, Abstr. #V52B-0437.

Zhang, S., Tullis, T.E., Scruggs, V.J., 1999. Permeability anisotropy and

pressure dependency of permeability in experimentally sheared gouge materials. J. Struct. Geol. 21, 795–806. https://doi.org/10.1016/S0191-8141(99)00080-2

Zhu, W., Montesi, L.G.J., Wong, T.-F., 1997. Shear-enhanced compaction and permeability reduction: Triaxial extension tests on porous sandstone. Mech. Mater. 25, 199–214. https://doi.org/10.1016/S0167-6636(97)00011-2

Zhu, W., Tivey, M.K., Gittings, H., Craddock, P.R., 2007. Permeability-porosity relationships in seafloor vent deposits: Dependence on pore evolution processes. J. Geophys. Res. Solid Earth 112, 1–15. https://doi.org/10.1029/2006JB004716

Zimmerman, R.W., 2018. Fluid flow in porous media.

Zoback, M.D., Byerlee, J.D., 1975. The effect of microcrack dilatancy on the permeability of westerly granite. J. Geophys. Res. 80, 752–755. https://doi.org/10.1029/JB080i005p00752

Zoback, M.D., Gorelick, S.M., 2012. Earthquake triggering and large-scale geologic storage of carbon dioxide. Proc. Natl. Acad. Sci. U. S. A. 109, 10164–8. https://doi.org/10.1073/pnas.1202473109

# APPENDIX

*Appendix: MATLAB Scripts*

```
classdef FaultFluidFlowClass < handle
    properties(Constant)
        SECONDS_PER_YEAR = 3.15569e7;
        SECONDS_PER_DAY = 86400;
        GRAVITATIONAL_CONSTANT = 9.81;
    end

    properties
        analyticalTime;
        maximumSimulationTime;
        timeVectorLength;
        time;
        timeOutput;
        timeVectorDensity;
        x;
        z;
        Delta;
        faultArchitectureList;
        simulatedFaultWidth;
        simulatedFaultHeight;
        horizontalArrayLength;
        verticalArrayLength;
        FaultArchitectureEnds;
        ModeOfFailureArchitectureFlag;
        SlidingFailureFlag;
        FineFeatureFlag;
        overpressureHeight;
        overpressureMap;
        pszWidth;
        blankingArray;
        EarthquakeLengthStore;
        EarthquakeLengthVector;
        CohesiveFlag;
        nucleationDetectionFactor = 5E5;

        faultPreset;
        rockDensity;
        faultAngle;
        FailureModeBoundary;
        FrictionCoefficient;
        porosity;
        porosityStates;
        compressiblity;
        Viscosity;
        UnstressedPermeability;
        PressureSensitivity;
        Cohesion;

        initialStressField;
        arrayoverpressureHeight;
        contactOverpressure;
        initialPressure;
        initialSolverVariable;
        pressure;
        Density;
        shearModulus;
        rateAndStateDifference;
        criticalSlipDistance;
        psi;
        slidingStress;
        FailureTime = struct(...
            'Brittle', NaN,...
            'Ductile', NaN,...
```

```
            'Stable', NaN,...
            'Unstable', NaN);
        SlidingLength = struct;
        cohesionLimit;
        failureStateList;
        FailureAngle;
        TwoCosFailureAngle;
        twoCosFaultAngle;

        ArrayFaultArchitectureEnds;
        ArrayFaultArchitectureMap;
        FailureModeBoundaryStress;
        hydrostaticStress;
        lithostaticStress;
        FailureMarker;
        slidingFailureMarker;
        failureExtent;
        maximumStress;
        minimumStress;
        Permeability;
        options;
        outputPressure;
        outputSolverVariable;
        FailureMarkerStore;
        slidingFailureMarkerStore;
        slidingStressStore;
        oldFailureMarker;
        newFailureMarker;
        PlotProperties = struct;
        twoCosAngle;
        twoSinAngle;
        internalFrictionArray;
        cohesion;
        TwoSinFailureAngle;
        twoSinFaultAngle;
        FailureEnvelope;
        poreFluidFactor;
        tectonicLoadingRate;
        faultDepth;
        confinementFactor;
        OFCwidth;
        IFCwidth;
        initialStress;
        modeOfFailureFlag = false;
        plotTimeScale;

        PlottingAngle;
    end

    methods
        function obj = initialise(...
                obj,...
                poreFluidFactor,...
                tectonicLoadingRate,...
                faultDepth,...
                confinementFactor,...
                overpressureHeight,...
                OFCwidth,...
                IFCwidth,...
                faultPreset,...
                varargin)

            % Initialise object for simulating fluid flow in a given fault
            % zone.
```

```
            FaultFluidFlowClass.printProgressString(...
                'Initialising problem parameters...')

            obj.poreFluidFactor = poreFluidFactor;
            obj.tectonicLoadingRate = tectonicLoadingRate;
            obj.faultDepth = faultDepth;
            obj.confinementFactor = confinementFactor;
            obj.overpressureHeight = overpressureHeight;
            obj.OFCwidth = OFCwidth;
            obj.IFCwidth = IFCwidth;

            obj.importFaultValues(faultPreset);

            obj.updateFaultWidthValues;

            obj.initialiseVarargin(varargin);

            obj.initialiseSpatialArray;

            obj.mapFaultArchitecture;

            obj.colfioritoOverpressureMap;

            obj.initialiseRockMatrixVariables;
            obj.initialiseIntensiveVariables(confinementFactor);

            obj.initialiseTimeVariables;
        end


        function initialiseIntensiveVariables(obj, confinementFactor)

            % Intialise intensive physical variables of fault zone.

            obj.initialiseNonSolverVariables(confinementFactor);
        end

        function initialiseVarargin(obj, varargin)

            %Process arguments in varargin input.

            if any(strcmp(varargin{:}, 'modeoffailure'))
                obj.modeOfFailureFlag = true;
            end
        end

        function initialiseNonSolverVariables(obj, confinementFactor)

            %Initialise physical variables not altered by solver.

            obj.hydrostaticStress = 1000 ...
                * FaultFluidFlowClass.GRAVITATIONAL_CONSTANT...
                * obj.faultDepth;
            obj.lithostaticStress = obj.rockDensity...
                * FaultFluidFlowClass.GRAVITATIONAL_CONSTANT...
                * (obj.faultDepth);

            obj.initialisePressure;

            obj.initialiseStress(confinementFactor);

            obj.initialPressure =...
                obj.pressureBCS(obj.initialPressure);

            [obj.Permeability,...
```

```
            obj.FailureMarker,...
            obj.slidingFailureMarker,...
            ~,...
            ~,...
            ~,...
            ~]...
            = obj.rockMatrixState(obj.initialPressure, 0);

    end

    function initialiseCommonRockMatrixVariables(obj)
        obj.slidingStress = NaN(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);

        obj.Permeability = obj.initialiseXZMidpointStruct;

        obj.FailureMarker = obj.initialiseXZCMidpointStruct;
        obj.slidingFailureMarker = false(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);

        obj.calculatePorosity(obj.FailureMarker);

        obj.initialiseFailureBlankingArray;

        obj.initialiseFailureAngles;
        obj.initialiseInternalFriction;
        obj.initialiseCohesion;
        obj.initialisePorosity;
    end

    function initialisePorosity(obj)
        obj.porosity = obj.initialiseArray;

        obj.porosity = obj.porosityStates.Prefailure;
    end

    function initialiseStress(obj, confinementFactor)

        % Calculate initial stress for simulation.

        obj.fixedInitialStress(confinementFactor);
        obj.calculateMaximumStress;
        obj.calculateMinimumStress(0);

        if isa(obj, 'SinglePhaseFluidFlowSolidVelocityClass')
            obj.initialStressField = cat(...
                3,...
                -obj.maximumStress,...
                -obj.minimumStress,...
                obj.maximumStress - obj.minimumStress);

            obj.initialStressField = repmat(...
                obj.initialStressField(1, 1, :),...
                obj.verticalArrayLength + 1,...
                obj.horizontalArrayLength + 1);
        end
    end

    function initialiseFailureBlankingArray(obj)

        %Returns an array true at array points where logical failure
        %can occur.
```

```
            obj.blankingArray = false(...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength);
            for loopCounter = 1:length(obj.faultArchitectureList)
                architectureComponent =...
                    obj.faultArchitectureList{loopCounter};

                if (obj.ModeOfFailureArchitectureFlag.(...
                        architectureComponent)...
                        && obj.modeOfFailureFlag)...
                        || obj.SlidingFailureFlag.(architectureComponent)
                    obj.blankingArray(...
                        obj.ArrayFaultArchitectureMap.(...
                        architectureComponent).Central) = true;
                end
            end
        end

        function initialisePressure(obj)

            %Set initial pressure for simulation.

            obj.contactOverpressure = obj.poreFluidFactor...
                * obj.lithostaticStress - obj.hydrostaticStress;

            obj.initialPressure = obj.hydrostaticStress...
                * ones(...
                obj.verticalArrayLength, obj.horizontalArrayLength);
        end

        function initialiseCommonSpatialArray(obj)

            % Initialise spatial arrays for simulation.
            obj.verticalSemiLogMesh;

            obj.arrayoverpressureHeight...
                = FaultFluidFlowClass.indexOfNearest(...
                obj.z,...
                obj.overpressureHeight);

            [obj.x, obj.z] = meshgrid(obj.x, obj.z);
            obj.Delta.X = diff(obj.x, 1, 2);
            obj.Delta.Z = diff(obj.z, 1, 1);
            obj.Delta.XX = (obj.x(:, 3:end) - obj.x(:, 1:end - 2)) / 2;
            obj.Delta.ZZ = (obj.z(3:end, :) - obj.z(1:end - 2, :)) / 2;
        end

        function verticalSemiLogMesh(obj)

            % Vertical semi-logarithmic spatial array.

            obj.x = (1:obj.horizontalArrayLength)...
                * obj.simulatedFaultWidth...
                / obj.horizontalArrayLength;

            obj.z = logspace(0, log10(obj.simulatedFaultHeight + 1),...
                obj.verticalArrayLength);
            obj.z = obj.z - 1;
        end

        function initialiseTimeVariables(obj)

            % Initialise time variables for solver.

            obj.calculateMaximumSimulationTime;
```

```
            obj.calculateAnalyticalTime;

            obj.timeVectorLength = round(...
                obj.timeVectorDensity...
                * obj.maximumSimulationTime...
            / obj.SECONDS_PER_YEAR);

            obj.time = 0:(obj.maximumSimulationTime...
                / obj.timeVectorLength):obj.maximumSimulationTime;
        end

        function updateFaultWidthValues(obj)

            % Update fault architecture component widths based on input.

            obj.FaultArchitectureEnds =...
                struct(...
                'OFC', obj.OFCwidth,...
                'IFC', obj.IFCwidth + obj.OFCwidth,...
                'PSZ', obj.IFCwidth + obj.OFCwidth - 1E-3);

            obj.simulatedFaultWidth = obj.OFCwidth + obj.IFCwidth;
        end

        function pressure = rockFluidCoupling(obj, pressure, time)

            % Physical coupling between rock and fluid.

            obj.oldFailureMarker = sum(obj.FailureMarker.X(:))...
                + sum(obj.FailureMarker.Z(:));

            obj.calculateMinimumStress(time);

            pressure = obj.pressureBCS(pressure);

            [obj.Permeability,...
                FailureMarkerLocal,...
                ~,...
                ~,...
                ~,...
                ~,...
                ~]...
                = obj.rockMatrixState(pressure, time);


            obj.newFailureMarker = sum(FailureMarkerLocal.X(:))...
                + sum(FailureMarkerLocal.Z(:));
        end

        function [value, isTerminal, direction] =  events(...
                obj,...
                time,...
                pressure)

            % ODES15S event trigger function.

            pressure = reshape(...
                pressure,...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength);

            pressure = obj.pressureBCS(pressure);

            [~,...
                ~,...
```

```
            ~,...
            distance,...
            failureLength,...
            nucleationLength,...
            ~]...
            = obj.rockMatrixState(pressure, time);

        value = min(distance(:));

        failureDiff = obj.nucleationDetectionFactor...
            * (nucleationLength - failureLength)...
            / nucleationLength;

        if abs(failureDiff) < abs(value) && value < 0
            value = failureDiff;
        end

        isTerminal = 1;
        direction = 0;

    end

    function obj = calculateMaximumStress(obj)

        % Calculate maximum stress at every spatial array point.

        obj.maximumStress = obj.lithostaticStress...
            * ones(obj.verticalArrayLength, obj.horizontalArrayLength);
    end

    function obj = calculateMinimumStress(obj, time)

        % Calculate minimum stress at every spatial array point.

        obj.extensionalMinimumStress(time);
    end

    function extensionalMinimumStress(obj, time)

        % If in extensionally defined minimum stress regime, calculate
        % minimum stress.

        obj.minimumStress = (obj.initialStress +...
            obj.tectonicLoadingRate * time / obj.SECONDS_PER_YEAR)...
            * ones(obj.verticalArrayLength, obj.horizontalArrayLength);
    end

    function pressureTimeDerivative = pressureTimeDerivative(...
            obj,...
            FluxDivergence)

        % Derivative of pressure with respect to time.

        pressureTimeDerivative = (...
            1 ./...
            (obj.compressiblity .* obj.porosity))...
            .* FluxDivergence.Central;
    end

    function FluxDivergence = fluxDivergenceBCS(...
            obj,...
            Flux,...
            FluxDivergence)

        % Enforce flux divergence boundary conditions.
```

```
            FluxDivergence.Z = [zeros(1, obj.horizontalArrayLength);...
                FluxDivergence.Z; zeros(1, obj.horizontalArrayLength)];
            FluxDivergence.X = [zeros(obj.verticalArrayLength, 1)...
                FluxDivergence.X zeros(obj.verticalArrayLength, 1)];

                FluxDivergence.Z(1, :) = 2 ...
                    .* ((Flux.Z(1, :))...
                    ./ obj.Delta.ZZ(1, :));

                FluxDivergence.X(:, end) = -2 ...
                    .* ((Flux.X(:, end))...
                    ./ obj.Delta.XX(:, end));

            FluxDivergence.Central = FluxDivergence.X + FluxDivergence.Z;
        end

        function Flux = pressureFlux(obj, pressure)

            % Struct of pressure fluxes in x and z directions and every
            % array point.

            PressureDerivative = obj.spatialDerivative(pressure);

            Flux = struct;
            Flux.X = obj.Permeability.X .* PressureDerivative.X...
                / obj.Viscosity.SinglePhase;
            Flux.Z = obj.Permeability.Z .* PressureDerivative.Z...
                / obj.Viscosity.SinglePhase;
        end

        function [Permeability,...
                FailureMarker,...
                slidingFailureMarker,...
                distance,...
                failureLength,...
                nucleationLength,...
                slidingStress]...
                = rockMatrixState(obj, pressure, time)

            % Evaluate physical conditions representing the state of the
            % rock matrix.

            obj.calculateMinimumStress(time);

            EffectivePressure = obj.initialiseXZCMidpointStruct;
            FailureMarker = EffectivePressure;
            EffectiveNormalStress = EffectivePressure;
            slidingFailureMarker = false(...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength);
            Permeability = obj.initialiseXZMidpointStruct;
            cohesionArray = obj.cohesion;
            internalFrictionLoop  = obj.internalFrictionArray;

            EffectivePressure = obj.effectivePressure(...
                EffectivePressure,...
                pressure);

            [twoCosFailureAngle, twoSinFailureAngle] = obj.calculateTwoFailureAngle(...
                EffectivePressure);

            for loopCounter = 1:length(obj.faultArchitectureList)
                architectureComponent =...
                    obj.faultArchitectureList{loopCounter};
```

```
            componentMap = obj.ArrayFaultArchitectureMap.(...
                architectureComponent);

            internalFrictionLoop = obj.internalFriction(...
                internalFrictionLoop,...
                EffectivePressure,...
                architectureComponent,...
                componentMap);

            cohesionArray = obj.calculateCohesion(...
                cohesionArray,...
                EffectivePressure,...
                architectureComponent,...
                componentMap);

            EffectiveNormalStress = obj.effectiveNormalStress(...
                pressure,...
                EffectiveNormalStress,...
                twoCosFailureAngle,...
                componentMap);
        end

        [logicalFailureMarker,...
            distance,...
            stressDifference,...
            slidingStress]...
            = obj.mohrAnalysis(...
            pressure,...
            internalFrictionLoop,...
            cohesionArray,...
            EffectiveNormalStress,...
            twoSinFailureAngle);

        obj.calculatePorosity(FailureMarker);

        for loopCounter = 1:length(obj.faultArchitectureList)
            architectureComponent =...
                obj.faultArchitectureList{loopCounter};
            componentMap = obj.ArrayFaultArchitectureMap.(...
                architectureComponent);

            Permeability = permeability(...
                obj,...
                EffectivePressure,...
                Permeability,...
                architectureComponent,...
                componentMap);

            if obj.modeOfFailureFlag
                FailureMarker = obj.markModeOfFailure(...
                    FailureMarker,...
                    architectureComponent,...
                    EffectivePressure,...
                    logicalFailureMarker,...
                    componentMap);
            end

            slidingFailureMarker = obj.markSlidingFailure(...
                slidingFailureMarker,...
                architectureComponent,...
                logicalFailureMarker,...
                componentMap);

        end
```

```
            failureLength = obj.failureLength(...
                stressDifference,...
                slidingFailureMarker);

            effectiveStressPatch = obj.calculateEffectiveStressPatch(...
                EffectiveNormalStress,...
                slidingFailureMarker,...
                failureLength);

            nucleationLength = obj.nucleationLength(...
                effectiveStressPatch);
        end

        function initialiseFailureAngles(obj)

            % Calcualte failure angles for each mode of failure.

            obj.twoCosFaultAngle = cosd(2 * obj.faultAngle);
            obj.twoSinFaultAngle = sind(2 * obj.faultAngle);

            obj.twoCosAngle = NaN(...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength);

            obj.twoSinAngle = NaN(...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength);

            for loopCounter = 1:length(obj.faultArchitectureList)
                architectureComponent =...
                    obj.faultArchitectureList{loopCounter};

                obj.FailureAngle.(architectureComponent).Brittle = 90 ...
                    + 0.5 * atand(-1 / obj.FrictionCoefficient.Brittle.(...
                    architectureComponent));

                obj.FailureAngle.(architectureComponent).Ductile = 90 ...
                    + 0.5 * atand(-1 / obj.FrictionCoefficient.Ductile.(...
                    architectureComponent));

                obj.TwoCosFailureAngle.(architectureComponent).Brittle...
                    = cosd(2 * obj.FailureAngle.(architectureComponent).Brittle);
                obj.TwoCosFailureAngle.(architectureComponent).Ductile...
                    = cosd(2 * obj.FailureAngle.(architectureComponent).Ductile);

                obj.TwoSinFailureAngle.(architectureComponent).Brittle...
                    = sind(2 * obj.FailureAngle.(architectureComponent).Brittle);
                obj.TwoSinFailureAngle.(architectureComponent).Ductile...
                    = sind(2 * obj.FailureAngle.(architectureComponent).Ductile);

                obj.FailureAngle.(architectureComponent).Brittle = [];

                obj.FailureAngle.(architectureComponent).Ductile = [];

                componentMap = obj.ArrayFaultArchitectureMap.(...
                    architectureComponent).Central;

                if obj.CohesiveFlag.(architectureComponent)

                    obj.twoCosAngle(componentMap)...
                        = obj.TwoCosFailureAngle.(...
                        architectureComponent).Brittle;

                    obj.twoSinAngle(componentMap)...
                        = obj.TwoSinFailureAngle.(...
```

```
                    architectureComponent).Brittle;

            else
                obj.twoCosAngle(componentMap)...
                    = obj.twoCosFaultAngle;

                obj.twoSinAngle(componentMap) = obj.twoSinFaultAngle;
            end

        end

    end

    function initialiseInternalFriction(obj)

        obj.internalFrictionArray = NaN(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);

        for loopCounter = 1:length(obj.faultArchitectureList)
            architectureComponent =...
                obj.faultArchitectureList{loopCounter};

            obj.internalFrictionArray(...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                = obj.FrictionCoefficient.Brittle.(...
                architectureComponent);
        end
    end

    function initialiseCohesion(obj)

        obj.cohesion = NaN(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);

        for loopCounter = 1:length(obj.faultArchitectureList)
            architectureComponent =...
                obj.faultArchitectureList{loopCounter};

            obj.cohesion(...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                = obj.Cohesion.Brittle.(...
                architectureComponent);
        end
    end

    function [twoCos, twoSin] = calculateTwoFailureAngle(...
            obj,...
            EffectivePressure)

        % Update failure angle based on failure state.

        twoCos = obj.twoCosAngle;
        twoSin = obj.twoSinAngle;

        for loopCounter = 1:length(obj.faultArchitectureList)

            architectureComponent =...
                obj.faultArchitectureList{loopCounter};
            if obj.CohesiveFlag.(architectureComponent)

                logicalMapDuctile = (EffectivePressure.Central...
```

```
                    > obj.FailureModeBoundary.(architectureComponent))...
                    & obj.ArrayFaultArchitectureMap.(...
                    architectureComponent).Central;

                twoCos(logicalMapDuctile)...
                    = obj.TwoCosFailureAngle.(architectureComponent).Ductile;

                twoSin(logicalMapDuctile)...
                    = obj.TwoSinFailureAngle.(architectureComponent).Ductile;
            end
        end
    end

    function failureLength = failureLength(...
            obj,...
            shearStrengthExcess,...
            failureMap)

        % Calculate failure length.

        if all(~failureMap)
            failureLength = NaN;
        else
                [failureLength, ~] = max(obj.z(failureMap));

                [failureEnd, ~] = find(obj.z == failureLength);
                failureEnd = failureEnd(end);

                if failureEnd == obj.verticalArrayLength
                    failureInterp = 0;
                else
                    failureInterp =...
                        -shearStrengthExcess(failureEnd, end)...
                        * (obj.z(failureEnd + 1, end)...
                        - failureLength)...
                        / (shearStrengthExcess(failureEnd + 1, end)...
                        - shearStrengthExcess(failureEnd, end));

                    %failureInterp(isnan(failureInterp)) = 0;
                end

                failureLength = 2 * (failureLength + failureInterp);
        end
    end

    function effectiveStressPatch = calculateEffectiveStressPatch(...
            obj,...
            EffectiveNormalStress,...
            failureMap,...
            failureLength)

        % Calculate effective stress of failure patch.

        if all(~failureMap)
            effectiveStressPatch = NaN;
        else
            effectiveStressPatch = obj.interpolateStressEdge(...
                EffectiveNormalStress,...
                failureMap,...
                failureLength);
        end
    end

    function effectiveStressPatch = interpolateStressEdge(...
            obj,...
```

```
            EffectiveNormalStress,...
            failureMap,...
            failureLength)

        % Interpolate stress to the edge of the failure patch.

        [row, ~] = find(max(obj.z(failureMap))...
            == obj.z);
        row = row(end);

        effectiveStress = EffectiveNormalStress.Central;

        logicalMap = ~isnan(obj.slidingStress);

        effectiveStress(logicalMap)...
            = obj.slidingStress(logicalMap);

        effectiveStressPatchVector = effectiveStress(...
            failureMap);


        [~, maxStressRow] = max(effectiveStressPatchVector);

        if row ~= maxStressRow
            effectiveStressPatch =...
                min(effectiveStressPatchVector);
        else
            if row ~= obj.verticalArrayLength
                stressInterp =...
                    (EffectiveNormalStress.Central(row + 1, end)...
                    - effectiveStressPatchVector(end))...
                    * (failureLength * 0.5 - obj.z(row, end));
            else
                stressInterp = 0;
            end

            effectiveStressPatch =...
                effectiveStressPatchVector(end)...
                + stressInterp;
        end
    end

    function EffectiveNormalStress = effectiveNormalStress(...
            obj,...
            pressure,...
            EffectiveNormalStress,...
            twoCosAngle,...
            componentMap)

        % Calculate effective normal stress.

        logicalMap = componentMap.Central;

        minStress = obj.minimumStress(logicalMap);

        maxStress = obj.maximumStress(logicalMap);

        EffectiveNormalStress.Central(...
            logicalMap) = 0.5...
            .* ((maxStress...
            + (minStress...
            - 2 .* pressure(...
            logicalMap)))...
            + (maxStress...
            - (minStress))...
```

```
            .* twoCosAngle(...
            logicalMap));

        EffectiveNormalStress.Central(obj.slidingFailureMarker)...
            = obj.slidingStress(obj.slidingFailureMarker);

        EffectiveNormalStress.X =...
            (EffectiveNormalStress.Central(:, 1:end-1)...
            + EffectiveNormalStress.Central(:, 2:end)) / 2;
        EffectiveNormalStress.Z =...
            (EffectiveNormalStress.Central(1:end-1, :)...
            + EffectiveNormalStress.Central(2:end, :)) / 2;
    end

    function nucleationLength = nucleationLength(...
            obj,...
            effectiveStressPatch)

        % Calculate nucleation length stability criterion.

        nucleationLength = obj.psi...
            * obj.shearModulus...
            * obj.criticalSlipDistance...
            ./ (effectiveStressPatch...
            * obj.rateAndStateDifference);
    end

    function [logicalFailureMarker,...
            distance,...
            stressDifference,...
            slidingStress]...
            = mohrAnalysis(...
            obj,...
            pressure,...
            internalFrictionArray,...
            cohesionArray,...
            EffectiveNormalStress,...
            twoSinFailureAngle)

        % Find spatial array points undergoing failure and distance
        % between failure envelope and Mohr circle and every point.

        slidingStress = NaN(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);
        meanStress = obj.meanStress - pressure;
        differentialStress = obj.calculateDifferentialStress;

        distance = obj.stressDifference(...
            EffectiveNormalStress.Central,...
            differentialStress,...
            internalFrictionArray,...
            cohesionArray,...
            twoSinFailureAngle);

        for loopCounter = 1:length(obj.faultArchitectureList)

            architectureComponent =...
                obj.faultArchitectureList{loopCounter};

            ComponentMap = obj.ArrayFaultArchitectureMap.(...
                architectureComponent);

            if obj.SlidingFailureFlag.(architectureComponent)
```

```
                faultPlaneDistance = obj.stressDifference(...
                    meanStress,...
                    differentialStress,...
                    internalFrictionArray,...
                    cohesionArray,...
                    twoSinFailureAngle);

                slidingStress = obj.slidingStress;

                logicalMap = ComponentMap.Central...
                    & isnan(obj.slidingStress);

                distance(...
                    logicalMap) = faultPlaneDistance(...
                    logicalMap);

                logicalMap = logicalMap...
                    & distance < 0;

                slidingStress(logicalMap) = meanStress(...
                    logicalMap);

            end
        end

        stressDifference = distance;

        distance(~obj.blankingArray) = NaN;

        logicalFailureMarker = obj.logicalFailureMarker(...
            distance);

        distance(obj.FailureMarker.Central ~= 0) = NaN;
        distance(obj.slidingFailureMarker) = NaN;
    end

    function stressDifference = stressDifference(...
            obj,...
            effNormStress,...
            differentialStress,...
            internalFrictionArray,...
            cohesionArray,...
            twoSin)

        % Stress difference between mohr circle and failure envelope.

        stressDifference...
            = obj.algebraicFailureEnvelope(...
            effNormStress,...
            internalFrictionArray,...
            cohesionArray)...
            - obj.algebraicMohrCircle(...
            differentialStress,...
            twoSin);

    end

    function Permeability = permeability(...
            obj,...
            EffectivePressure,...
            Permeability,...
            architectureComponent,...
            componentMap)

        % Return permeability struct for fault.
```

```
            Permeability =...
                FaultFluidFlowClass.architecturePermeability(...
                Permeability,...
                obj.FailureMarker,...
                EffectivePressure,...
                obj.UnstressedPermeability.(...
                architectureComponent),...
                obj.PressureSensitivity.(...
                architectureComponent),...
                componentMap);
        end

        function calculatePorosity(...
                obj,...
                FailureMarker)

            % Calculate porosity for a given set of physical and failure
            % conditions.

            if isfield(obj.porosityStates, 'Brittle')
                obj.porosity(FailureMarker.Central == 1)...
                    = obj.porosityStates.Brittle;
            end

            if isfield(obj.porosityStates, 'Ductile')
                obj.porosity(FailureMarker.Central == 2)...
                    = obj.porosityStates.Ductile;
            end
        end

        function meanStress = meanStress(obj)

            % Calculate the mean stress at every spatial array point.

            meanStress = 0.5 * (obj.maximumStress + obj.minimumStress);
        end

        function FailureMarker = markModeOfFailure(...
                obj,...
                FailureMarker,...
                architectureComponent,...
                EffectivePressure,...
                logicalFailureMarker,...
                ComponentMap)

            % Take logical failure marker and build failure marker struct
            % for every spatial array point.

            if obj.ModeOfFailureArchitectureFlag.(...
                    architectureComponent)
                FailureMarker.Central(...
                    obj.brittleFailureArrayCondition(...
                    EffectivePressure,...
                    logicalFailureMarker,...
                    architectureComponent, 'Central')) = 1;
                FailureMarker.Central(...
                    obj.ductileFailureArrayCondition(...
                    EffectivePressure, logicalFailureMarker,...
                    architectureComponent, 'Central')) = 2;

                FailureMarker.X(ComponentMap.X)...
                    = FailureMarker.Central(ComponentMap.X);
                FailureMarker.Z(ComponentMap.Z)...
                    = FailureMarker.Central(ComponentMap.Z);
```

```
        end
    end

    function slidingFailureMarker = markSlidingFailure(...
            obj,...
            slidingFailureMarker,...
            architectureComponent,...
            logicalFailureMarker,...
            ComponentMap)

        % Take logical failure marker and build failure marker struct
        % for every spatial array point.

        if obj.SlidingFailureFlag.(architectureComponent)

            slidingFailureMarker(...
                logicalFailureMarker...
                & ComponentMap.Central) = true;

        end
    end

    function cohesionArray =  calculateCohesion(...
            obj,...
            cohesionArray,...
            EffectivePressure,...
            architectureComponent,...
            componentMap)

        % Return array of cohesions at every spatial array point.

        cohesionArray(...
            (EffectivePressure.Central...
            > obj.FailureModeBoundary.(architectureComponent))...
            & componentMap.Central)...
            = obj.Cohesion.Ductile.(architectureComponent);
    end

    function internalFrictionArray = internalFriction(...
            obj,...
            internalFrictionArray,...
            EffectivePressure,...
            architectureComponent,...
            componentMap)

        % Calculate internal friction at each spatial array points.

        internalFrictionArray(...
            (EffectivePressure.Central...
            > obj.FailureModeBoundary.(architectureComponent))...
            & componentMap.Central)...
            = obj.FrictionCoefficient.Ductile.(...
            architectureComponent);
    end

    function condition = brittleFailureArrayCondition(...
            obj,...
            EffectivePressure,...
            logicalFailureMarker,...
            architectureComponent,...
            structString)

        % Condition for brittle failure at an array point.

        condition = (((EffectivePressure.Central...
```

```
            <= obj.FailureModeBoundary.(architectureComponent)...
            & obj.FailureMarker.(structString) == 0)...
            & logicalFailureMarker)...
            | obj.FailureMarker.(structString) == 1)...
            & obj.ArrayFaultArchitectureMap.(...
            architectureComponent).(structString);
    end

    function condition = ductileFailureArrayCondition(...
            obj,...
            EffectivePressure,...
            logicalFailureMarker,...
            architectureComponent,...
            structString)

        %Condition for ductile failure at an array point.

        condition = (((EffectivePressure.Central...
            > obj.FailureModeBoundary.(architectureComponent)...
            & obj.FailureMarker.(structString) == 0)...
            & logicalFailureMarker)...
            | obj.FailureMarker.(structString) == 2)...
            & obj.ArrayFaultArchitectureMap.(...
            architectureComponent).(structString);
    end

    function ShearStrength =...
            failureEnvelope(obj,...
            EffectiveNormalStress,...
            EffectivePressure,...
            ShearStrength,...
            architectureComponent)

        % Shear strength at each spatial array point.

        fields = fieldnames(teststruct);
        for loopCounter = 1:numel(fields)
            ShearStrength.(fields{loopCounter})(...
                obj.brittleFailureEnvelopeCondition(...
                EffectivePressure, architectureComponent,...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent), fields{loopCounter}))...
                = FaultFluidFlowClass.algebraicFailureEnvelope(...
                EffectiveNormalStress.(fields{loopCounter})(...
                obj.brittleFailureEnvelopeCondition(...
                architectureComponent,...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent),...
                fields{loopCounter})),...
                obj.FrictionCoefficient.Brittle.(...
                architectureComponent),...
                obj.Cohesion.Brittle.(architectureComponent));

            ShearStrength.(...
                fields{...
                loopCounter})(obj.ductileFailureEnvelopeCondition(...
                EffectivePressure, architectureComponent,...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent), fields{loopCounter}))...
                = FaultFluidFlowClass.algebraicFailureEnvelope(...
                EffectiveNormalStress.(fields{loopCounter})(...
                obj.ductileFailureEnvelopeCondition(...
                architectureComponent,...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent),...
```

```
                fields{loopCounter})),...
                obj.FrictionCoefficient.Ductile.(...
                architectureComponent),...
                obj.Cohesion.Ductile.(architectureComponent));
            ShearStrength.(...
                fields{...
                loopCounter})(...
                ShearStrength.(fields{loopCounter}) < 0 & ...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent).(fields{loopCounter})) = 0;
        end
    end

    function condition = brittleFailureEnvelopeCondition(...
            obj,...
            EffectivePressure,...
            architectureComponent,...
            structString)

        % Condition for array point to be susceptible to brittle
        % failure.

        condition = EffectivePressure.(structString)...
            <= obj.FailureModeBoundary.(architectureComponent)...
            & obj.ArrayFaultArchitectureMap.(...
            architectureComponent).(structString);
    end

    function condition = ductileFailureEnvelopeCondition(...
            obj,...
            EffectivePressure,...
            architectureComponent,...
            structString)

        % Condition for array point to be susceptible to ductile
        % failure.

        condition = EffectivePressure.(structString)...
            > obj.FailureModeBoundary.(architectureComponent)...
            & obj.ArrayFaultArchitectureMap.(...
            architectureComponent).(structString);
    end

    function ShearStress = shearStress(...
            obj,...
            differentialStress,...
            failureAngle,...
            ShearStress,...
            architectureComponent)

        % Shear stress at each central array point.

        fields = fieldnames(teststruct);
        for loopCounter = 1:numel(fields)
            ShearStress.Central(...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent).(fields{loopCounter})) =...
                FaultFluidFlowClass.shearStressRelation(...
                differentialStress,...
                failureAngle.(fields{loopCounter})(...
                obj.ArrayFaultArchitectureMap.(...
                architectureComponent).(fields{loopCounter})));
        end
    end
```

```
        function obj = calculateMaximumSimulationTime(obj)

            % Calculate maximum length of time for which simulation could
            % run.

            for loopCounter = 1:length(obj.faultArchitectureList)
                architectureComponent =...
                    obj.faultArchitectureList{loopCounter};
                if obj.SlidingFailureFlag.(architectureComponent)
                    obj.maximumSimulationTime = ...
                        1.5 * obj.analyticalPrediction(...
                        0.4,...
                        architectureComponent);
                end
            end
        end

        function obj = calculateAnalyticalTime(obj)

            % Calculate analytical prediction of earthquake timing.

            for loopCounter = 1:length(obj.faultArchitectureList)
                architectureComponent =...
                    obj.faultArchitectureList{loopCounter};
                if obj.SlidingFailureFlag.(architectureComponent)
                    obj.analyticalTime = ...
                        obj.analyticalPrediction(...
                        obj.poreFluidFactor,...
                        architectureComponent);
                end
            end
        end

        function analyticalTime = analyticalPrediction(...
                obj,...
                poreFluidFactor,...
                architectureComponent)

            % Analytical prediction of stable sliding on fault.

            analyticalTime = FaultFluidFlowClass.SECONDS_PER_YEAR * ...
                ((obj.lithostaticStress - obj.initialStress) *...
                (sind(2 * obj.faultAngle)) + 2 ...
                * obj.FrictionCoefficient.Brittle.(...
                architectureComponent)...
                * (poreFluidFactor...
                * obj.lithostaticStress -...
                obj.lithostaticStress * cosd(obj.faultAngle) ^ 2 -...
                obj.initialStress * sind(obj.faultAngle) ^ 2))...
                / (obj.tectonicLoadingRate * (sind(2 * obj.faultAngle)...
                + 2 * obj.FrictionCoefficient.Brittle.(...
                architectureComponent)...
                * sind(obj.faultAngle) ^ 2));
            analyticalTime(analyticalTime < 0) = 0;
        end

        function EffectivePressure = effectivePressure(....
                obj,...
                EffectivePressure,...
                pressure)

            % Effective pressure struct and midpoint spatial arrays.

            EffectivePressure.Central=...
                obj.minimumStress...
```

```
            - pressure;

        EffectivePressure.Central(EffectivePressure.Central < 0) = 0;

        EffectivePressure.X = (EffectivePressure.Central(:, 1:(end - 1))...
            + EffectivePressure.Central(:, 2:end)) / 2;
        EffectivePressure.Z = (EffectivePressure.Central(1:(end - 1), :)...
            + EffectivePressure.Central(2:end, :)) / 2;
    end

    function pressure = pressureBCS(obj, pressure)

        % Apply pressure boundary conditions.

                pressure(:, 1) = obj.hydrostaticStress;

                pressure(end, :) = obj.hydrostaticStress;

        pressure(obj.overpressureMap) =...
            obj.contactOverpressure + obj.hydrostaticStress;
    end

    function obj = fixedInitialStress(obj, confinementFactor)

        % Initial stress conditions for fault zones without coupling
        % between pore fluid and initial stress state.
        % (Typically no significant regional stress.)

        obj.initialStress =...
            confinementFactor...
            * obj.lithostaticStress;
    end

    function differentialStress = calculateDifferentialStress(obj)

        % Calculate differential stress array at each simulated spatial
        % point.

        differentialStress = obj.maximumStress - obj.minimumStress;
    end

    function obj = mapFaultArchitecture(obj)

        % Map fault zone architecture dimensions to array.

        obj.assignFaultArchitectureArrayEnds;

        cellLength = length(obj.faultArchitectureList);
        startEndVector = zeros(cellLength + 1, 1);
        startEndVector(1) = 0;

        for loopCounter = 1:cellLength
            startEndVector(loopCounter + 1) =...
                obj.ArrayFaultArchitectureEnds.(...
                obj.faultArchitectureList{loopCounter});

            obj.ArrayFaultArchitectureMap.(...
                obj.faultArchitectureList{loopCounter}) =...
                obj.mapFaultComponent(startEndVector(loopCounter),...
                startEndVector(loopCounter + 1));

            % Enforce PSZ
            if loopCounter ~= cellLength
                obj.ArrayFaultArchitectureMap.(...
                obj.faultArchitectureList{loopCounter}).Central(:, end) = false;
```

```
            obj.ArrayFaultArchitectureMap.(...
                obj.faultArchitectureList{loopCounter}).Z(:, end) = false;

            obj.ArrayFaultArchitectureMap.(...
                obj.faultArchitectureList{loopCounter}).X(:, end) = false;
            end
        end

    end

    function FaultComponentMap =...
            mapFaultComponent(obj,...
            faultComponentStart,...
            faultComponentEnd)

        % Map Fault component of fault zone architecture to array.

        FaultComponentMap =...
            obj.initialiseLogicalXZCMidpointStruct(false);

        FaultComponentMap = obj.mapParallelFaultComponent(...
            faultComponentStart, faultComponentEnd,...
            FaultComponentMap);
    end

    function FaultComponentMap = mapParallelFaultComponent(...
            obj,...
            faultComponentStart,...
            faultComponentEnd,...
            FaultComponentMap)

        % Map a fault component parallel to the fault plane.

        if faultComponentStart ~= obj.horizontalArrayLength
            faultComponentStart = faultComponentStart + 1;
        end

        if faultComponentStart > faultComponentEnd
            faultComponentEnd = faultComponentStart;
        end

        FaultComponentMap.Central(1:(obj.verticalArrayLength),...
            faultComponentStart:faultComponentEnd) = true;

        FaultComponentMap.Z(1:(obj.verticalArrayLength - 1),...
            faultComponentStart:faultComponentEnd) = true;

        if faultComponentEnd == obj.horizontalArrayLength
            faultComponentEnd = faultComponentEnd - 1;

            if faultComponentEnd - faultComponentStart < 0
                faultComponentStart = faultComponentEnd;
            end
        end
        FaultComponentMap.X(1:obj.verticalArrayLength,...
            faultComponentStart:faultComponentEnd)...
            = true;
    end

    function sameEffectiveStressTransitionAssumption(obj)

        % Enforce the assumption

        obj.FailureModeBoundaryStress.IFC...
```

```
            = obj.FailureModeBoundaryStress.OFC;
    end

    function obj = assignFaultArchitectureArrayEnds(obj)

        % Build struct to hold positions of ends of fault zone
        % architecture in array space.

        for loopCounter = 1:length(obj.faultArchitectureList)
            obj.faultArchitectureArrayEnd(...
                obj.faultArchitectureList{loopCounter});
        end
    end

    function obj = faultArchitectureArrayEnd(obj, componentName)

        % Calculate the ends of the components of fault zone
        % architecture in array space.

        obj.ArrayFaultArchitectureEnds.(componentName) =...
            ceil((obj.horizontalArrayLength)...
            * obj.FaultArchitectureEnds.(componentName)...
            / obj.simulatedFaultWidth);

        obj.ArrayFaultArchitectureEnds.(componentName)(...
            obj.ArrayFaultArchitectureEnds.(componentName)...
            > obj.horizontalArrayLength)...
            = obj.horizontalArrayLength;
    end

    function Derivative = spatialDerivative(obj, array)

        % Spatial first derivative of physical variable.

        Derivative = struct;
        Derivative.X = diff(array, 1, 2) ./ obj.Delta.X;
        Derivative.Z = diff(array, 1, 1) ./ obj.Delta.Z;
    end

    function Derivative =  spatialSecondDerivative(obj, InputStruct)
        Derivative = struct('X', [], 'Z', []);
        Derivative.X = diff(InputStruct.X, 1, 2) ./ obj.Delta.XX;
        Derivative.Z = diff(InputStruct.Z, 1, 1) ./ obj.Delta.ZZ;
    end

    function outputStruct = initialiseLogicalXZCMidpointStruct(...
            obj,...
            bool)

        % Initialise struct of boolean arrays based on spatial centres
        % and midpoints.

        if bool
            outputStruct = struct('X', true(obj.verticalArrayLength,...
                obj.horizontalArrayLength - 1), 'Z',...
                true(obj.verticalArrayLength - 1,...
                obj.horizontalArrayLength),...
                'Central', true(obj.verticalArrayLength,...
                obj.horizontalArrayLength));
        else
            outputStruct = struct('X', false(...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength - 1), 'Z',...
                false(obj.verticalArrayLength - 1,...
                obj.horizontalArrayLength),...
```

```
                'Central', false(obj.verticalArrayLength,...
                obj.horizontalArrayLength));
        end
    end

    function outputStruct = initialiseXZCMidpointStruct(obj)

        % Initialise a struct containing x-, z-midpoint and central
        % spatial arrays.

        outputStruct = struct(...
            'X',...
            zeros(obj.verticalArrayLength,...
            obj.horizontalArrayLength - 1),...
            'Z',...
            zeros(obj.verticalArrayLength - 1,...
            obj.horizontalArrayLength),...
            'Central',...
            zeros(obj.verticalArrayLength,...
            obj.horizontalArrayLength));
    end

    function outputStruct = initialiseXZMidpointStruct(obj)

        % Initialise a struct with x- and z-midpoint spatial arrays.

        outputStruct = struct('X', zeros(obj.verticalArrayLength,...
            obj.horizontalArrayLength - 1),...
            'Z', zeros(obj.verticalArrayLength - 1,...
            obj.horizontalArrayLength));
    end

    function outputStruct = initialiseXZStruct(obj)

        % Initialise x and z spatial arrays.

        outputStruct = struct(...
            'X', zeros(obj.verticalArrayLength,...
            obj.horizontalArrayLength),...
            'Z', zeros(obj.verticalArrayLength,...
            obj.horizontalArrayLength));
    end

    function outputArray = initialiseArray(obj)

        % Initialise and x and z array.

        outputArray = nan(obj.verticalArrayLength,...
            obj.horizontalArrayLength);
    end

    function initialiseSpatialArray(obj)

        % Initialise spatial arrays for dummy.

        obj.initialiseCommonSpatialArray;
    end

    function initialiseRockMatrixVariables(obj)

        % Initialise common rock matrix variables for dummy instance.

        obj.initialiseCommonRockMatrixVariables;
    end
```

```
        function importFaultValues(obj, faultPreset)

            % Import the fault specific properties for a given fault
            % preset.

            obj.faultPreset = faultPreset;

            switch faultPreset
                case 'Colfiorito'

                    obj.faultArchitectureList = {'OFC', 'IFC', 'PSZ'};
                    obj.failureStateList = {'Brittle', 'Ductile'};
                    obj.simulatedFaultHeight = 1000;
                    obj.horizontalArrayLength = 175;
                    obj.verticalArrayLength = 200;
                    obj.timeVectorDensity = 1;

                    obj.ModeOfFailureArchitectureFlag =...
                        struct('OFC', true, 'IFC', false, 'PSZ', false);
                    obj.SlidingFailureFlag =....
                        struct('OFC', false, 'IFC', false, 'PSZ', true);
                    obj.FineFeatureFlag =...
                        struct('OFC', false, 'IFC', false, 'PSZ', true);
                    obj.CohesiveFlag =...
                        struct('OFC', true, 'IFC', false, 'PSZ', false);
                    obj.colfioritoOverpressureMap;
                    obj.pszWidth = 1E-3;
                    obj.shearModulus = 45.7E9;
                    obj.rateAndStateDifference = 0.003;
                    obj.criticalSlipDistance = 0.000063;
                    obj.psi = 1;
                    obj.rockDensity = 2650;
                    obj.faultAngle = 45;
                    obj.FrictionCoefficient =...
                        struct('Brittle',...
                        struct(...
                        'OFC', 0.704,...
                        'IFC', 0.84,...
                        'PSZ', 0.6),...
                        'Ductile',...
                        struct('OFC', 0, 'IFC', 0, 'PSZ', NaN));
                    obj.FailureModeBoundary =...
                        struct(...
                        'OFC', 10E6,...
                        'IFC', 10E6,...
                        'PSZ' , 1E10);
                    obj.FailureModeBoundaryStress =...
                        struct(...
                        'OFC', 32E6,...
                        'IFC', 32E6,...
                        'PSZ' , 1E10);
                    obj.Cohesion = struct(...
                        'Brittle',  struct(...
                        'OFC', 15.5E6,...
                        'IFC', 26.4E6,...
                        'PSZ', 0),...
                        'Ductile', struct(...
                        'OFC', 38.14E6,...
                        'IFC', 53.28E6,...
                        'PSZ', NaN));
                    obj.cohesionLimit = obj.Cohesion.Brittle.OFC;

                    obj.compressiblity = 1E-10;
                    obj.Viscosity = struct(...
                        'SinglePhase', 1E-5);
```

```
                obj.porosityStates = struct(...
                    'Prefailure', 0.01,...
                    'Brittle', 0.0175,...
                    'Ductile', 0.015);
                obj.UnstressedPermeability =...
                    struct(...
                    'OFC', struct(...
                    'Prefailure', struct('X', 8E-21, 'Z', 3E-19),...
                    'Brittle',...
                    struct('X', 1.1287E-18, 'Z', 1.39E-17),...
                    'Ductile',...
                    struct('X', 2.407E-18, 'Z', 3.681E-18)),...
                    'IFC',...
                    struct(...
                    'Prefailure', struct('X', 1E-19, 'Z', 1E-17)),...
                    'PSZ',...
                    struct('Prefailure',...
                    struct('X', 1E-21, 'Z', 1E-19)));
                obj.PressureSensitivity = struct(...
                    'OFC', struct(...
                    'Prefailure', struct('X', -4E-8, 'Z', -1.3E-7),...
                    'Brittle', struct('X', 0, 'Z', 0),...
                    'Ductile',...
                    struct('X', -1.136E-7, 'Z', -7.968E-8)),...
                    'IFC',...
                    struct('Prefailure', struct('X', 0, 'Z', 0)),...
                    'PSZ',...
                    struct('Prefailure', struct('X', 0, 'Z', 0)));

                obj.PlotProperties.OFC...
                    = {'Color', 'red', 'LineWidth', 2.5};
                obj.PlotProperties.IFC...
                    = {'Color', 'cyan', 'LineWidth', 1.5};
                obj.PlotProperties.PSZ...
                    = {'Color', 'black'};
                obj.plotTimeScale = 'years';

                obj.PlottingAngle.Brittle.OFC = 60;
                obj.PlottingAngle.Ductile.OFC = 60;

            otherwise
                error('Fault preset not recognised.')
        end
    end

    function colfioritoOverpressureMap(obj)

        % Implement map of overpressure particular to Colfiorito
        % example.

        obj.overpressureMap...
            = false(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);
        obj.overpressureMap(...
            1:obj.arrayoverpressureHeight, 1)...
            = 1;
    end
end

methods(Static)
    function PermeabilityArrayStruct = architecturePermeability(...
            PermeabilityArrayStruct,...
            FailureMarker,...
```

```
            EffectivePressure,...
            UnstressedPermeabilityStruct,...
            PressureSensitivityStruct,...
            ArchitectureMap)

        % Calculate permeability for all components of fault zone
        % architecture.

        fields = fieldnames(PermeabilityArrayStruct);
        for loopCounter = 1:numel(fields)

            architectureComponent = fields{loopCounter};

            logicalMap = ...
                FailureMarker.(architectureComponent) == 0 ...
                & ArchitectureMap.(architectureComponent);

            PermeabilityArrayStruct.(architectureComponent)(...
                logicalMap) =...
                FaultFluidFlowClass.algebraicPermeability(...
                UnstressedPermeabilityStruct.Prefailure.(...
                architectureComponent),...
                EffectivePressure.(...
                architectureComponent)(...
                logicalMap),...
                PressureSensitivityStruct.Prefailure.(...
                architectureComponent));

            if any(FailureMarker.(...
                    fields{...
                    loopCounter})(...
                    ArchitectureMap.(architectureComponent)) == 1)

                logicalMap = ...
                FailureMarker.(architectureComponent) == 1 ...
                & ArchitectureMap.(architectureComponent);

                PermeabilityArrayStruct.(...
                    fields{...
                    loopCounter})(...
                    logicalMap) =...
                    FaultFluidFlowClass.algebraicPermeability(...
                    UnstressedPermeabilityStruct.Brittle.(...
                    architectureComponent),...
                    EffectivePressure.(...
                    architectureComponent)(...
                    logicalMap),...
                    PressureSensitivityStruct.Brittle.(...
                    architectureComponent));
            end

            if any(FailureMarker.(...
                    fields{...
                    loopCounter})(...
                    ArchitectureMap.(architectureComponent)) == 2)

                logicalMap = ...
                FailureMarker.(architectureComponent) == 2 ...
                & ArchitectureMap.(architectureComponent);

                PermeabilityArrayStruct.(...
                    architectureComponent)(...
                    logicalMap) =...
                    FaultFluidFlowClass.algebraicPermeability(...
                    UnstressedPermeabilityStruct.Ductile.(...
```

```
                    architectureComponent),...
                    EffectivePressure.(...
                    architectureComponent)(...
                    logicalMap),...
                    PressureSensitivityStruct.Ductile.(...
                    architectureComponent));
            end
        end
    end

    function logicalFailureMarker...
            = logicalFailureMarker(distance)

        % Calculate failure marker array as boolean.

        tolerance = 0;

        logicalFailureMarker = false(size(distance));
        logicalFailureMarker(distance <= tolerance) = true;
    end

    function mohr = algebraicMohrCircle(...
            differentialStress,...
            twoSin)

        % Calculate value of mohr circle as analytical relationship.

        mohr = (differentialStress / 2) .* twoSin;
    end

    function permeability = algebraicPermeability(...
            unstressedPermeability,...
            effectivePressure,...
            pressureSensitivity)

        % Calculate value of permeability as analytical relationship.

        permeability = unstressedPermeability .*...
            exp(pressureSensitivity...
            .* effectivePressure);

    end

    function inputStruct = assignToXZCMidpointStruct(...
            inputStruct,...
            value)
        inputStruct.Central = value;

        % Set all arrays in spatial position struct to a given value.

        inputStruct...
            = FaultFluidFlowClass.interpolateStructArray(...
            inputStruct);
    end

    function inputStruct = interpolateStructArray(inputStruct)

        % Interpolate central spatial struct to X and Z midpoints.

        inputStruct.X = (inputStruct.Central(:, 2:end)...
            + inputStruct.Central(:, 1: end - 1)) / 2;
        inputStruct.Z = (inputStruct.Central(2:end, :)...
            + inputStruct.Central(1:end -1, :)) / 2;
    end
```

```
        function index = indexOfNearest(input, value)

            % Find index of point in array nearest to a given value.

            temp = abs(input - value);
            [~ , index] = min(temp);
        end

        function output = copyToXZStruct(input)

            %Copy an input to both X and Z array midpoints.

            output = struct;
            output.X = input;
            output.Z = input;
        end

        function shearStress = shearStressRelation(...
                differentialStress,...
                failureAngle)

            % Shear stress relationship.

            shearStress = differentialStress...
                .* sind(2 .* failureAngle)...
                / 2;
        end

        function height = semiCircleHeight(x, MohrRadius, MohrCentre)

            % Find height of Mohr semi-circle.

            height = sqrt(MohrRadius .^ 2 - (MohrCentre - x) .^ 2);
        end

        function printProgressString(string)

            % Output current solver progress to console.

            disp(string)
            fprintf('\n')
        end

                function shearStrength = algebraicFailureEnvelope(...
                effectiveNormalStress,...
                frictionCoefficient,...
                cohesion)

            % Return failure envelope given effective normal stress,
            % cohesion and friction coefficient.

            shearStrength...
                = cohesion...
                + frictionCoefficient...
                .* effectiveNormalStress;
        end
    end
end
```

```
ans =

  FaultFluidFlowClass with properties:

                SECONDS_PER_YEAR: 31556900
```

```
                SECONDS_PER_DAY: 86400
           GRAVITATIONAL_CONSTANT: 9.8100
                   analyticalTime: []
            maximumSimulationTime: []
                 timeVectorLength: []
                             time: []
                       timeOutput: []
                timeVectorDensity: []
                                x: []
                                z: []
                            Delta: []
            faultArchitectureList: []
              simulatedFaultWidth: []
             simulatedFaultHeight: []
            horizontalArrayLength: []
              verticalArrayLength: []
            FaultArchitectureEnds: []
ModeOfFailureArchitectureFlag: []
               SlidingFailureFlag: []
                  FineFeatureFlag: []
               overpressureHeight: []
                  overpressureMap: []
                          pszWidth: []
                    blankingArray: []
           EarthquakeLengthStore: []
          EarthquakeLengthVector: []
                     CohesiveFlag: []
    nucleationDetectionFactor: 500000
                   faultPreset: []
                   rockDensity: []
                    faultAngle: []
              FailureModeBoundary: []
              FrictionCoefficient: []
                         porosity: []
                 porosityStates: []
                 compressiblity: []
                      Viscosity: []
          UnstressedPermeability: []
             PressureSensitivity: []
                         Cohesion: []
              initialStressField: []
          arrayoverpressureHeight: []
              contactOverpressure: []
                  initialPressure: []
           initialSolverVariable: []
                       pressure: []
                        Density: []
                  shearModulus: []
          rateAndStateDifference: []
            criticalSlipDistance: []
                             psi: []
                   slidingStress: []
                     FailureTime: [1×1 struct]
                   SlidingLength: [1×1 struct]
                   cohesionLimit: []
                failureStateList: []
                    FailureAngle: []
              TwoCosFailureAngle: []
                twoCosFaultAngle: []
     ArrayFaultArchitectureEnds: []
      ArrayFaultArchitectureMap: []
      FailureModeBoundaryStress: []
              hydrostaticStress: []
              lithostaticStress: []
                  FailureMarker: []
```

```
                slidingFailureMarker: []
                       failureExtent: []
                       maximumStress: []
                       minimumStress: []
                        Permeability: []
                             options: []
                      outputPressure: []
                outputSolverVariable: []
                  FailureMarkerStore: []
           slidingFailureMarkerStore: []
                  slidingStressStore: []
                    oldFailureMarker: []
                    newFailureMarker: []
                      PlotProperties: [1×1 struct]
                         twoCosAngle: []
                         twoSinAngle: []
              internalFrictionArray: []
                             cohesion: []
                 TwoSinFailureAngle: []
                    twoSinFaultAngle: []
                     FailureEnvelope: []
                     poreFluidFactor: []
                  tectonicLoadingRate: []
                           faultDepth: []
                 confinementFactor: []
                           OFCwidth: []
                           IFCwidth: []
                       initialStress: []
                   modeOfFailureFlag: 0
                       plotTimeScale: []
                       PlottingAngle: []
```

---

*Published with MATLAB® R2018b*

```
function faultFluidFlowScript(varargin)

% OOP based script to simulate fluid flow for a given fault zone
% architecture.
% INSTRUCTIONS: Pass the following input variables when running
% pressureDiffusionScript.
% Pore fluid factor lambda. Can be vector. (REQUIRED)
% Tectonic loading rate, extension. Can be vector. (REQUIRED)
% Fault depth. Can be vector. (REQUIRED)
% Confinement factor (of lithostatic stress).(REQUIRED).
% Overpressure half height. (REQUIRED)
% OFC width. (REQUIRED)
% IFC half width. (REQUIRED)
% Fault Preset. ('Colfiorito'). (REQUIRED.)
% Fail test pass string 'modeoffailure'. Turn on/off mode of failure
% behaviour. (REQUIRED FOR CASE STUDY)
% (Omit in case of paramter study.)
%Include analytical predictions pass string 'analytical'.
% (OPTIONAL, ONLY USED IN PARAMETER STUDY.)
% Side-by-side plot data side by side for specific paper diagrams.
% Include string 'sidebyside' (OPTIONAL).
% Examples: Case Study:
% pressureDiffusionScript(0.45, -1.5E5, 7000, 0.7, 'Colfiorito', 'modeoffailure')

% Parameter Study:
% pressureDiffusionScript(0.4:0.05:0.7, -1.5E5, 7000:250:7500, 0.7, 'Colfiorito', 'analytical')

% Side By Side:
% pressureDiffusionScript([0.45, 0.7], -1.5E5, 7000, 0.7,'Colfiorito', 'sidebyside')

clc
format long eng
tic

[analyticalFlag, sideBySideFlag, resultPlotFlag, fileName]...
    = processVarargin(varargin);

parallelIndex = indicesOfVectorInputs(varargin{:});

if ~any(strcmp(varargin, 'plotonly'))
    folderName = initialiseFolder(parallelIndex);

    saveInputsToFile(folderName, varargin{:});

else
    folderName = [];

end

if all(~parallelIndex)

    if strcmp(resultPlotFlag, 'plotonly')
        FaultFluidFlowMat = load(fileName);

        FaultFluidFlow = FaultFluidFlowMat.FaultFluidFlow;

    else
        FaultFluidFlow = solveFaultFluidFlow(varargin{:});

    end

    toc;
```

```
    if ~strcmp(resultPlotFlag, 'resultonly')
        plotResults(FaultFluidFlow, folderName, varargin{:})

    end

    if ~strcmp(resultPlotFlag, 'plotonly')
          save([folderName '/FaultFluidFlowResults.mat'],...
           'FaultFluidFlow')

    end

    close all;
else
    FaultFluidFlowClass.printProgressString(...
        'Initialise parameter study...')

    [gridVariables, parameterStudyFailureVector,...
        parameterStudyNoFailureVector]...
        = initialiseParameterStudyVariables(parallelIndex, varargin);

    SideBySideCellFailure = cell(2, 1);
    SideBySideCellNoFailure = cell(2, 1);

    parfor parallelLoopCounter = 1:length(gridVariables{1}(:))
        loopVector = varargin;

        loopVector(parallelIndex) = cellfun(...
            @(x)x(parallelLoopCounter),...
            gridVariables, 'un', 0);

        if sideBySideFlag
            if ~strcmp(resultPlotFlag, 'plotonly')
                [SideBySideCellFailure{parallelLoopCounter},...
                    SideBySideCellNoFailure{parallelLoopCounter}]...
                    = sideBySideStudy(...
                    loopVector,...
                    folderName,...
                    parallelLoopCounter);

            end
        else

            if ~strcmp(resultPlotFlag, 'plotonly')

                [parameterStudyFailureVector(parallelLoopCounter, :),...
                    parameterStudyNoFailureVector(...
                    parallelLoopCounter,...
                    :)]...
                    = parameterStudy(loopVector,  folderName);

            end
        end
    end

    if sideBySideFlag
            postProcessSideBySideResults(...
                folderName,...
                SideBySideCellFailure,...
                SideBySideCellNoFailure,...
                gridVariables,...
                parallelIndex,...
                resultPlotFlag,...
                fileName,...
                varargin);
```

```
    else
            postProcessParameterStudyResults(...
                parameterStudyFailureVector,...
                parameterStudyNoFailureVector,...
                folderName,...
                gridVariables,...
                parallelIndex,...
                analyticalFlag,...
                resultPlotFlag,...
                fileName,...
                varargin{:});

    end
    close all;
end

toc
end

function [SideBySideCellFailureOutput, SideBySideCellNoFailureOutput]...
    = sideBySideStudy(loopVector, folderName, parallelLoopCounter)

% Perform case studies necessary for side by side plotting.

[FaultFluidFlowFailure, FaultFluidFlowNoFailure]...
    = parameterStudySimulation(loopVector);

SideBySidePlottingFailure = processSideBySideResults(...
    FaultFluidFlowFailure,...
    folderName,...
    [num2str(parallelLoopCounter), '_1']);
SideBySidePlottingNoFailure = processSideBySideResults(...
    FaultFluidFlowNoFailure,...
    folderName,...
    [num2str(parallelLoopCounter), '_2']);

%Struct output necessary for parallelisation.
[SideBySideCellFailureOutput,...
    SideBySideCellNoFailureOutput]...
    = storeToSideBySideCell(...
    SideBySidePlottingFailure,...
    SideBySidePlottingNoFailure);
end

function postProcessSideBySideResults(...
    folderName,...
    SideBySideCellFailure,...
    SideBySideCellNoFailure,...
    gridVariables,...
    parallelIndex,...
    resultPlotFlag,...
    fileName,...
    argCell)

% Postprocess case study results for side by side plotting.

DummyFaultFluidFlow = dummyFaultFluidFlowClass(...
    gridVariables,...
    parallelIndex,...
    argCell);

if strcmp(resultPlotFlag, 'plotonly')
    SideBySidePlottingMat = load(fileName);
    SideBySidePlotting = SideBySidePlottingMat.SideBySidePlotting;
```

```
else
    SideBySidePlotting = SideBySidePlottingClass;

    SideBySidePlotting.initialise(...
        DummyFaultFluidFlow,...
        folderName);

    SideBySidePlotting.SideBySideResultStruct = struct(...
        'Failure',...
        SideBySideCellFailure,...
        'NoFailure',...
        SideBySideCellNoFailure);
end

if ~strcmp(resultPlotFlag, 'resultonly')
    SideBySidePlotting.sideBySidePlot(...
        DummyFaultFluidFlow);

end

if ~strcmp(resultPlotFlag, 'plotonly')
    save(...
        [folderName '/SideBySideStudyResults.mat'],...
        'SideBySidePlotting')

end
end

function SideBySidePlotting = processSideBySideResults(...
    FaultFluidFlow,...
    folderName,...
    fileNumber)

% Process side by side results

SideBySidePlotting = SideBySidePlottingClass;
SideBySidePlotting.initialise(FaultFluidFlow, folderName);
SideBySidePlotting.resultProcessing(FaultFluidFlow);
SideBySidePlotting.faultPlaneFailurePlot(FaultFluidFlow, fileNumber);
end

function postProcessParameterStudyResults(...
    parameterStudyFailureVector,...
    parameterStudyNoFailureVector,...
    folderName,...
    gridVariables,...
    parallelIndex,...
    analyticalFlag,...
    resultPlotFlag,...
    fileName,...
    varargin)

% Postprocess parameter study results.

if strcmp(resultPlotFlag, 'plotonly')
    ParameterStudyPlottingMat = load(fileName);
    ParameterStudyPlotting...
        = ParameterStudyPlottingMat.ParameterStudyPlotting;

else
    ParameterStudyPlotting = ParameterStudyPlottingClass;
    ParameterStudyPlotting.initialiseParameterStudy(...
        folderName,...
        gridVariables,...
        varargin{:},...
```

```
        analyticalFlag);

end

if ~strcmp(resultPlotFlag, 'plotonly')
      ParameterStudyPlotting.postProcessParameterStudyResults(...
                parameterStudyFailureVector,...
                parameterStudyNoFailureVector);

end

if ~strcmp(resultPlotFlag, 'resultonly')
    ParameterStudyPlotting.parameterStudyPlot(...
        parallelIndex,...
        gridVariables,...
        analyticalFlag);

end

if ~strcmp(resultPlotFlag, 'plotonly')
    save([folderName '/ParameterStudyResults.mat'],...
        'ParameterStudyPlotting')

end
end

function [gridVariables,...
    parameterStudyVector,...
    parameterStudyNoFailureVector]...
    = initialiseParameterStudyVariables(parallelIndex, argCell)

% Initialise variables necessary to perform parameter study.

parallelVariable = argCell(parallelIndex);
gridVariables = cell(1, numel(parallelVariable));
[gridVariables{:}] = ndgrid(parallelVariable{:});
gridLength = length(gridVariables{1}(:));

parameterStudyVector = NaN(gridLength, 11);
parameterStudyNoFailureVector = NaN(gridLength, 11);
end

function [FaultFluidFlowFailure, FaultFluidFlowNoFailure]...
    = parameterStudySimulation(loopVector)

% Peform set of simulations required for parameter study.

FaultFluidFlowFailure = solveFaultFluidFlow(...
    loopVector{1:8},...
    'modeoffailure');
FaultFluidFlowNoFailure = solveFaultFluidFlow(...
    loopVector{1:8});
end

function [parameterStudyFailureVectorOutput,...
    parameterStudyNoFailureVectorOutput]...
    = parameterStudy(loopVector,  folderName)

% Perform parameter study.

[FaultFluidFlowFailure, FaultFluidFlowNoFailure]...
    = parameterStudySimulation(loopVector);

ParameterStudyPlottingFailure = ParameterStudyPlottingClass;
ParameterStudyPlottingFailure.initialise(...
```

```
    FaultFluidFlowFailure,...
    folderName);

ParameterStudyPlottingNoFailure...
    = ParameterStudyPlottingClass;
ParameterStudyPlottingNoFailure.initialise(...
    FaultFluidFlowNoFailure,...
    folderName);

%Vector output necessary for parallelisation.
parameterStudyFailureVectorOutput =...
    ParameterStudyPlottingFailure.resultProcessing(...
    FaultFluidFlowFailure);

parameterStudyNoFailureVectorOutput =...
    ParameterStudyPlottingNoFailure.resultProcessing(...
    FaultFluidFlowNoFailure);
end

function FaultFluidFlow = solveFaultFluidFlow(varargin)

% Solve fault fluid flow problem for a given fault zone.
FaultFluidFlow = SinglePhaseFluidFlowClass;

FaultFluidFlow.initialise(varargin{:});

FaultFluidFlow.faultFluidFlowSolver();
end

function plotResults(FaultFluidFlow, folderName, varargin)

% Plot case study results.

ResultPlotting = ResultPlottingClass;

ResultPlotting.initialise(FaultFluidFlow, folderName);

ResultPlotting.caseStudyFigures(FaultFluidFlow);
end

function [analyticalFlag, sideBySideFlag, resultPlotFlag, fileName]...
    = processVarargin(varargin)

% Process variable input arguments.

fileName = '';

if any(strcmp(varargin{:}, 'analytical'))
    analyticalFlag = 'analytical';

else
    analyticalFlag = '';

end

if any(strcmp(varargin{:}, 'sidebyside'))
    sideBySideFlag = true;

else
    sideBySideFlag = false;

end

if any(strcmp(varargin{:}, 'plotonly'))
    resultPlotFlag = 'plotonly';
```

```
    idx = find(strcmp(varargin{:}, 'plotonly')) + 1;

    fileName = varargin{1}{idx};

elseif any(strcmp(varargin{:}, 'resultonly'))
    resultPlotFlag = 'resultonly';

else
    resultPlotFlag = 'both';

end
end

function [SideBySideCellFailure, SideBySideCellNoFailure]...
    = storeToSideBySideCell(...
    SideBySidePlotting,...
    SideBySidePlottingNoFailure)

% Create cell for storing side by side case studies simulation results.

SideBySideCellFailure =...
    SideBySidePlotting.SideBySideStruct;

SideBySideCellNoFailure =...
    SideBySidePlottingNoFailure.SideBySideStruct(...
    1:(length(SideBySidePlotting.SideBySideStruct)-1));

end

function DummyFaultFluidFlow = dummyFaultFluidFlowClass(...
    gridVariables,...
    parallelIndex,...
    argCell)

% Create a dummy fault fluid flow class instance to enable side by side
% plotting.


DummyFaultFluidFlow = FaultFluidFlowClass;

dummyVariables = argCell;

dummyVariables(parallelIndex) = cellfun(@(x)x(1), gridVariables, 'un', 0);

DummyFaultFluidFlow.initialise(dummyVariables{:});
end

function s = convertNum(n)

    % Convert number to string.

   s = [];
   while n > 0
      d = mod(n,10);
      s = [char(48+d), s];
      n = (n-d)/10;
   end
end

function saveInputsToFile(folderName, varargin)

% Store simulation inputs to file.

fileId = fopen([folderName '/input.txt'], 'w');
```

```
filevec = ['Pore Fluid Factor ', convertNum(varargin{1}),...
    ' Tectonic Loading Rate ', convertNum(varargin{2}),...
    ' Fault Depth ', convertNum(varargin{3}), ...
    ' Confinement Factor ' , convertNum(varargin{4}),...
    ' Overpressure Contact Height ' , convertNum(varargin{5}),...
    ' OFC Width ' , convertNum(varargin{6}),...
    ' IFC Width ' , convertNum(varargin{7}),...
    ' Fault Preset ' , convertNum(varargin{8}),...
    ' ' varargin{9:end}];
fprintf(fileId, '%s', filevec);
end

function folderName = initialiseFolder(parallelIndex)

% Initialise folder for storing inputs and results.

if sum(parallelIndex) ~= 0
    folderName = ['ParameterStudy' datestr(datetime('now'),...
        'ddmmyyHHMMSS')];
else
    folderName = [ 'FaultFluidFlow' datestr(datetime('now'),...
        'ddmmyyHHMMSS')];
end

ResultPlottingClass.makeDirectory(folderName);
end

function parallelIndex = indicesOfVectorInputs(varargin)

% Identify vectorised inputs for parallel case studies
% (parameter study or side by side plotting.)

parallelIndex = cellfun(@(e) length(e) ~= 1, varargin(1:7));
end
```

```
Index exceeds the number of array elements (0).

Error in faultFluidFlowScript>indicesOfVectorInputs (line 486)
parallelIndex = cellfun(@(e) length(e) ~= 1, varargin(1:7));

Error in faultFluidFlowScript (line 38)
parallelIndex = indicesOfVectorInputs(varargin{:});
```

*Published with MATLAB® R2018b*

```
classdef ParameterStudyPlottingClass < ResultPlottingClass
    properties
        FailureTime = struct(...
            'Brittle', NaN,...
            'Ductile', NaN,...
            'Stable', NaN,...
            'Unstable', NaN);

        FailureExtent = struct'
        ParameterStudyFailureTime = struct(...
            'Brittle', struct, 'Ductile', struct,...
            'Stable', struct('General', struct),...
            'Unstable', struct('General', struct));
        ParameterStudyFailureExtent = struct(...
            'Stable', struct,...
            'Unstable', struct);
        PatchSize
        analyticalTime;
        SlidingLength;

        poreFluidFactorList;
        tectonicLoadingRate;
        faultDepth;
        confinementFactor;
        overpressureHeight;
        OFCwidth;
        IFCwidth;

        failureStringCell = {'Failure', 'NoFailure'};

        options;
        simulatedFaultWidth;
        simulatedFaultHeight;
    end

    methods
        function obj = initialiseParameterStudy(...
                obj,...
                folderName,...
                grid,...
                poreFluidFactorList,...
                tectonicLoadingRate,...
                faultDepth,...
                confinementFactor,...
                overpressureHeight,...
                OFCwidth,...
                IFCwidth,...
                varargin)
            obj.ParameterStudyFailureTime.Stable.Failure = NaN(...
                length(grid{1}(:)));
            obj.ParameterStudyFailureTime.Unstable.Failure = NaN(...
                length(grid{1}(:) ));
            obj.ParameterStudyFailureTime.Stable.NoFailure = NaN(...
                length(grid{1}(:)));
            obj.ParameterStudyFailureTime.Unstable.NoFailure = NaN(...
            length(grid{1}(:)));
            obj.ParameterStudyFailureTime.Brittle = NaN(...
                length(grid{1}(:)));
            obj.ParameterStudyFailureTime.Ductile = NaN(...
                length(grid{1}(:)));
            obj.ParameterStudyFailureExtent.Stable = NaN(...
                length(grid{1}(:)));
            obj.ParameterStudyFailureExtent.Unstable = NaN(...
```

```
            length(grid{1}(:)));

        obj.poreFluidFactorList = poreFluidFactorList;
        obj.tectonicLoadingRate = tectonicLoadingRate;
        obj.faultDepth = faultDepth;
        obj.confinementFactor = confinementFactor;
        obj.overpressureHeight = overpressureHeight;
        obj.OFCwidth = OFCwidth;
        obj.IFCwidth = IFCwidth;

        obj.folderName = folderName;
    end

    function obj = assignEarthquakeLengthsToStruct(...
            obj,...
            parallelVector,...
            noFailureParallelVector)

        % Transfer results from parallel loop output vector to object
        % instance.

        obj.SlidingLength.Failure.Failure.Stable...
            = parallelVector(:, 6);
        obj.SlidingLength.Failure.Failure.Unstable...
            = parallelVector(:, 7);
        obj.SlidingLength.Failure.Nucleation.Stable =...
            parallelVector(:, 8);
        obj.SlidingLength.Failure.Nucleation.Unstable =...
            parallelVector(:, 9);
        obj.FailureExtent.Stable.Failure =...
            parallelVector(:, 10);
        obj.FailureExtent.Unstable.Failure =...
            parallelVector(:, 11);

        obj.SlidingLength.NoFailure.Failure.Stable =...
            noFailureParallelVector(:, 6);
        obj.SlidingLength.NoFailure.Failure.Unstable =...
            noFailureParallelVector(:, 7);
        obj.SlidingLength.NoFailure.Nucleation.Stable =...
            noFailureParallelVector(:, 8);
        obj.SlidingLength.NoFailure.Nucleation.Unstable =...
            noFailureParallelVector(:, 9);
        obj.FailureExtent.Stable.NoFailure =...
            noFailureParallelVector(:, 10);
        obj.FailureExtent.Unstable.NoFailure =...
            noFailureParallelVector(:, 11);
    end

    function obj = postProcessParameterStudyResults(...
            obj,...
            parallelVector,...
            noFailureParallelVector)

        % Apply post processing to parameter study results.

        obj.readDataFromParallelVector(...
            parallelVector,...
            noFailureParallelVector);

        obj.assignEarthquakeLengthsToStruct(...
            parallelVector,...
            noFailureParallelVector);

    end
```

```
        function obj = parameterStudyPlot(...
                obj,...
                parallelIndex,...
                gridVariable,...
                analyticalFlag)

            switch sum(parallelIndex)
                case 1
                    obj.oneVectorInputPlot(parallelIndex, analyticalFlag)
                case 2
                    obj.twoVectorInputPlot(gridVariable, parallelIndex)
                case 3
                    obj.threeVectorInputPlot(gridVariable, parallelIndex)
            end

        end

        function readDataFromParallelVector(...
                obj,...
                parallelVector,...
                noFailureParallelVector)

        obj.ParameterStudyFailureTime.Stable.Failure...
            = parallelVector(:, 3);
            obj.ParameterStudyFailureTime.Unstable.Failure...
                = parallelVector(:, 4);

            obj.ParameterStudyFailureTime.Stable.NoFailure...
                = noFailureParallelVector(:, 3);
            obj.ParameterStudyFailureTime.Unstable.NoFailure...
                = noFailureParallelVector(:, 4);

            obj.ParameterStudyFailureTime.Brittle = parallelVector(:, 1);
            obj.ParameterStudyFailureTime.Ductile = parallelVector(:, 2);
            obj.analyticalTime = parallelVector(:, 5);


        end

        function oneVectorInputPlot(obj, parallelIndex, analyticalFlag)
            if parallelIndex(1) == 1
                PlotStruct = struct(...
                    'plotVariable1',...
                    obj.poreFluidFactorList,...
                    'PlotString1',...
                    'Pore Fluid Factor');

            elseif parallelIndex(2) == 1
                PlotStruct = struct(...
                    'plotVariable1',...
                    obj.tectonicLoadingRate,...
                    'PlotString1',...
                    'Tectonic Loading Rate');

            elseif parallelIndex(3) == 1
                PlotStruct = struct(...
                    'plotVariable1',...
                    obj.faultDepth, ...
                    'PlotString1',...
                    'Fault Depth (m)');

            elseif parallelIndex(4) == 1
                PlotStruct = struct(...
                    'plotVariable1',...
                    obj.confinementFactor, ...
```

```
                'PlotString1',...
                'Confinement Factor');

        elseif parallelIndex(5) == 1
            PlotStruct = struct(...
                'plotVariable1',...
                2 * obj.overpressureHeight, ...
                'PlotString1',...
                'Overpressure Contact Height (m)');

        elseif parallelIndex(6) == 1
            PlotStruct = struct(...
                'plotVariable1',...
                obj.OFCwidth, ...
                'PlotString1',...
                'Outer Fault Core Width (m)');

        elseif parallelIndex(7) == 1
            PlotStruct = struct(...
                'plotVariable1',...
                2 * obj.IFCwidth, ...
                'PlotString1',...
                'Inner Fault Core Width (m)');

        end
        obj.onePlot(PlotStruct, analyticalFlag);
    end

    function twoVectorInputPlot(obj, gridVariable, parallelIndex)
        if parallelIndex(1) == 1 && parallelIndex(3) == 1
            obj.poreFluidFactorList = gridVariable{1};
            obj.faultDepth = gridVariable{2};

            PlotStruct = struct(...
                'PlotString1',...
                'Pore Fluid Factor',...
                'PlotString2',...
                'Fault Depth (m)');

            obj.twoPlot(gridVariable, PlotStruct);
        elseif parallelIndex(3) == 1 && parallelIndex(4) == 1
            obj.faultDepth = gridVariable{1};
            obj.confinementFactor = gridVariable{2};

            PlotStruct = struct(...
                'PlotString1',...
                'Fault Depth (m)',...
                'PlotString2',...
                'Initial Confinement (Pa)');

            obj.twoPlot(gridVariable, PlotStruct);
        elseif parallelIndex(1) == 1 && parallelIndex(4) == 1
            obj.poreFluidFactorList = gridVariable{1};
            obj.confinementFactor = gridVariable{2};

            PlotStruct = struct(...
                'PlotString1',...
                'Pore Fluid Factor',...
                'PlotString2',...
                'Initial Confinement (Pa)');

            obj.twoPlot(gridVariable, PlotStruct);
        end
    end
```

```
        function threeVectorInputPlot(obj, gridVariable, parallelIndex)
            if parallelIndex(1) == 1 ...
                && parallelIndex(2) == 1 ...
                && parallelIndex(3) == 1
            PlotStruct = struct(...
                    'plotVariable1', obj.faultDepth,...
                    'plotVariable2', obj.poreFluidFactorList,...
                    'plotVariable3', obj.tectonicLoadingRate,...
                    'PlotString1', 'Fault Depth (m)', ...
                    'PlotString2', 'Pore Fluid Factor',...
                    'plotString3', 'Tectonic Loading Rate (Pa/year)');

                obj.threePlot(gridVariable, PlotStruct);
            end
        end

        function onePlot(obj, PlotStruct, analyticalFlag)

            % Plot parameter study for one vector input.

            obj.plotInterseismicPeriod(PlotStruct, analyticalFlag);
            obj.plotNucleationLength(PlotStruct);
            obj.plotNucleationPhase(PlotStruct);
            obj.plotFailureExtent(PlotStruct);
        end

        function outputImage = plotInterseismicPeriod(...
                obj,...
                PlotStruct,...
                analyticalFlag)

            % Plot interseismic period.

            figure;
            hold on

            [p3, p4] = obj.addFailureRectanglesToPlot(...
                PlotStruct,...
                obj.ParameterStudyFailureTime.Unstable.Failure,...
                obj.ParameterStudyFailureTime.Unstable.NoFailure);

            p1 = plot(...
                PlotStruct.plotVariable1, ...
                obj.ParameterStudyFailureTime.Unstable.Failure,...
                'r*-');
            p2 = plot(...
                PlotStruct.plotVariable1,...
                obj.ParameterStudyFailureTime.Unstable.NoFailure,...
                'k*-');

            axis tight;
            xlabel(PlotStruct.PlotString1)
            ylabel('Interseismic period (years)')

            legcell = {...
                'Deformation-dependent permeability',...
                 'No deformation-dependent permeability'};

            if analyticalFlag
                plot(...
                    PlotStruct.plotVariable1,...
                    obj.analyticalTime,...
                    'r*');
                legcell = {...
                    legcell{:}, ...
```

```
                'Analytical Prediction of Stable Sliding'};
        end

        if ~isempty(p3)
            legcell = {...
                'Brittle failure',...
                legcell{:}};
        end

        if ~isempty(p4)
            legcell = {...
                'Ductile failure',...
                legcell{:}};
        end

        legend(...
            [p3(~isempty(p3)),...
            p4(~isempty(p4)),...
            p1(~isempty(p1)),...
            p2(~isempty(p2))],...
            legcell{:},...
            'Location',...
            'southwest');
        hold off;
        drawnow

        outputImage = ResultPlottingClass.copyPlotToImage(gcf);
        obj.saveFigure('FailureEventTiming');
    end

    function outputImage = plotNucleationPhase(obj, PlotStruct)

        % Plot nucleation time.

        nucleationTime = obj.nucleationTime('Failure');

        nucleationTimeNoFailure = obj.nucleationTime('NoFailure');

        figure;
        hold on;
        [p3, p4] = obj.addFailureRectanglesToPlot(...
            PlotStruct,...
            nucleationTime,...
            nucleationTimeNoFailure);

        p1 = plot(PlotStruct.plotVariable1,...
            nucleationTime,...
            'r*-');
        p2 = plot(...
            PlotStruct.plotVariable1,...
            nucleationTimeNoFailure,...
            'k*-');

        legcell = {...
            'Deformation-dependent permeability',...
             'No deformation-dependent permeability'};

        if ~isempty(p3)
            legcell = {...
                'Brittle failure',...
                legcell{:}};
        end

        if ~isempty(p4)
            legcell = {...
```

```
                'Ductile failure',...
                legcell{:}};
        end

        legend(...
            [p3(~isempty(p3)),...
            p4(~isempty(p4)),...
            p1(~isempty(p1)),...
            p2(~isempty(p2))],...
            legcell{:},...
            'Location',...
            'southeast');
        axis tight;
        hold off
        xlabel(PlotStruct.PlotString1)
        ylabel(['Nucleation phase (years).'])
        drawnow
        set(gca, 'YScale', 'log');

        outputImage = ResultPlottingClass.copyPlotToImage(gcf);
        obj.saveFigure('NucleationPhaseTiming');
    end

    function nucleationTime = nucleationTime(obj, failureString)

        % Calculate nucleation time.

        nucleationTime = (obj.ParameterStudyFailureTime.Unstable.(...
            failureString)...
            - obj.ParameterStudyFailureTime.Stable.(failureString));
    end

    function outputImage = plotNucleationLength(obj, PlotStruct)

        % Plot nucleation length.

        figure;
        hold on;
        [p3, p4] = obj.addFailureRectanglesToPlot(PlotStruct,...
            obj.SlidingLength.Failure.Failure.Unstable,...
            obj.SlidingLength.NoFailure.Failure.Unstable,...
            obj.SlidingLength.Failure.Failure.Unstable,...
            obj.SlidingLength.NoFailure.Failure.Unstable);

        p1 = plot(PlotStruct.plotVariable1,...
            obj.SlidingLength.Failure.Failure.Unstable, 'r*-');
        p2 = plot(PlotStruct.plotVariable1,...
            obj.SlidingLength.NoFailure.Failure.Unstable, 'k*-');

        legcell = {...
            'Deformation-dependent permeability',...
             'No deformation-dependent permeability'};

        if ~isempty(p3)
            legcell = {...
                'Brittle failure',...
                legcell{:}};
        end

        if ~isempty(p4)
            legcell = {...
                'Ductile failure',...
                legcell{:}};
        end
```

```
            axis tight;
            hold off;
            xlabel(PlotStruct.PlotString1);
            ylabel('Nucleation length (m)')
            legend(...
                [p3(~isempty(p3)),...
                p4(~isempty(p4)),...
                p1(~isempty(p1)),...
                p2(~isempty(p2))],...
                legcell{:},...
                'Location',...
                'northwest');
            drawnow

            outputImage = ResultPlottingClass.copyPlotToImage(gcf);
            obj.saveFigure('FailureLengths');
        end

        function outputImage = plotFailureExtent(obj, PlotStruct)

            % Plot extent of failure in OFC for each value of parameter.

            figure;
            hold on;
            [p3, p4] = obj.addFailureRectanglesToPlot(...
                PlotStruct,...
                obj.FailureExtent.Stable.Failure,...
                obj.FailureExtent.Unstable.Failure,...
                obj.FailureExtent.Stable.Failure,...
                obj.FailureExtent.Unstable.Failure);

            p1 = plot(...
                PlotStruct.plotVariable1,...
                obj.FailureExtent.Stable.Failure,...
                'r*-');
            p2 = plot(...
                PlotStruct.plotVariable1,...
               obj.FailureExtent.Unstable.Failure,...
                'k*-');

            legcell = {...
                'Deformation-dependent permeability',...
                 'No deformation-dependent permeability'};

            if ~isempty(p3)
                legcell = {...
                    'Brittle',...
                    legcell{:}};
            end

            if ~isempty(p4)
                legcell = {...
                    'Ductile',...
                    legcell{:}};
            end

            axis tight;
            hold off;
            xlabel(PlotStruct.PlotString1);
            ylabel('OFC Failure Extent Ratio')
            legend(...
                [p3(~isempty(p3)),...
                p4(~isempty(p4)),...
                p1(~isempty(p1)),...
                p2(~isempty(p2))],...
```

```
            legcell{:},...
            'Location',...
            'southeast');
        drawnow

        outputImage = ResultPlottingClass.copyPlotToImage(gcf);
        obj.saveFigure('FailureExtent');
    end

    function obj = twoPlot(obj, gridVariable, PlotStruct)

        % Plotting in two dimensions.

        obj.reshapeInputsToGridDimensions(gridVariable);
        obj.parameterPcolor(...
            gridVariable,...
            PlotStruct,...
            'BrittleFailure',...
            obj.ParameterStudyFailureTime.Brittle,...
            'Time of Brittle Failure');
        obj.parameterPcolor(...
            gridVariable,...
            PlotStruct,...
            'DuctileFailure',...
            obj.ParameterStudyFailureTime.Unstable.Failure,...
            'Time of Ductile Failure');

        obj.modeOfFailureContour(gridVariable, PlotStruct);

        obj.parameterPcolor(...
            gridVariable,...
            PlotStruct,...
            'StableSlidingWithFailure',...
            obj.ParameterStudyFailureTime.Stable.Failure,...
            'Time of Stable Sliding');

        obj.parameterPcolor(...
            gridVariable,...
            PlotStruct,...
            'UnstableSlidingWithFailure',...
            obj.ParameterStudyFailureTime.Unstable.Failure,...
            'Time of Unstable Sliding');

        obj.parameterPcolor(...
            gridVariable,...
            PlotStruct,...
            'StableSlidingNoFailure',...
            obj.ParameterStudyFailureTime.Stable.Failure,...
            'Time of Stable Sliding');

        obj.parameterPcolor(...
            gridVariable,...
            PlotStruct,...
            'UnstableSlidingNoFailure',...
            obj.ParameterStudyFailureTime.Stable.Failure,...
            'Time of /Unstable Sliding');

        nucleationTime = (...
            obj.ParameterStudyFailureTime.Stable.Failure -...
            obj.ParameterStudyFailureTime.Unstable.Failure);
        nucleationTimeNoFailure = (...
            obj.ParameterStudyFailureTime.Stable.Failure -...
            obj.ParameterStudyFailureTime.Unstable.Failure);

        obj.parameterPcolor(...
```

```
            gridVariable,...
            PlotStruct,...
            'NucleationPhaseWithFailure',...
            nucleationTime,...
            ['Length of Nucleation Phase (years).']),...
        obj.parameterPcolor(...
            gridVariable, PlotStruct,...
            'NucleationPhaseNoFailure',...
            nucleationTimeNoFailure,...
            ['Length of Nucleation Phase (years).'])
    end

    function ParameterStudyFailureTimePcolor(...
            obj,...
            gridVariable,...
            PlotStruct,...
            failureFlagString,...
            stabilityString)

        % Two dimensional failure time plot.

        figure;
        hold on
        pcolor(...
            gridVariable{1},...
            gridVariable{2},...
            obj.ParameterStudyFailureTime.(...
            stabilityString).(failureFlagString));
        shading interp
        axis tight;
        hold off
        xlabel(PlotStruct.PlotString1)
        ylabel(PlotStruct.PlotString2)
        title(...
            ['Timing of PSZ ' stabilityString ' sliding'])
        drawnow
        xlabel(colorbar, ...
            ['Timing of PSZ ' stabilityString ' sliding (years).'])
        obj.saveFigure(...
            [ stabilityString 'ParameterStudyFailureTime'...
            failureFlagString]);
    end

    function modeOfFailurePcolor(...
            obj,...
            gridVariable,...
            PlotStruct,...
            failureString)

        % Two dimensional mode of failure plot.

                    if any(...
                            isfinite(...
                            obj.ParameterStudyFailureTime.(...
                            failureString)(:)))
            figure;
            hold on
            pcolor(gridVariable{1}, gridVariable{2}, ...
                obj.ParameterStudyFailureTime.(failureString));
            shading interp
            axis tight;
            hold off
            xlabel(PlotStruct.PlotString1)
            ylabel(PlotStruct.PlotString2)
            title(...
```

```
                ['Timing of OFC' failureString 'Failure '])
            drawnow
            xlabel(colorbar, ...
                ['Timing of OFC' failureString 'Failure '])
            obj.saveFigure([failureString ' Failure Time']);
                    end
    end

    function modeOfFailureContour(obj, gridVariable, PlotStruct)

        % Add contour to two dimensional plot, to indicate mode of
        % failure.

        mark = obj.setContourVariable(gridVariable);
        contourNumber = max(max(mark));

        if ~all(mark(:) == 0)
            figure;
            hold on
            [C, h] = contourf(gridVariable{1},...
                gridVariable{2}, mark,...
                contourNumber, 'k');
            v = [0, 1, 2, 3];
            if ~isempty(C)
                clabel(C, h, v);
                text(C(1, 2), C(2, 2), 'Brittle');
                text(C(1, end), C(2, end), 'Ductile');
            end
            axis tight;
            hold off
            xlabel(PlotStruct.PlotString1)
            ylabel(PlotStruct.PlotString2)
            title(...
                'Brittle and ductile failure regions ')
            drawnow;
            obj.saveFigure('ModeOfFailureContour');
        end
    end

    function mark = setContourVariable(obj, gridVariable)

        % Assign the values of the variable used in two dimensional
        % contour plots.

        mark = zeros(size(gridVariable{1}));
        mark(isfinite(obj.ParameterStudyFailureTime.Brittle)) = 1 ;
        mark(isfinite(obj.ParameterStudyFailureTime.Ductile)) = 2 ;
        mark(isfinite(obj.ParameterStudyFailureTime.Brittle)...
            & isfinite(obj.ParameterStudyFailureTime.Ductile))...
            = 3;
    end

    function parameterPcolor(...
            obj,...
            gridVariable,...
            PlotStruct,...
            fileName,...
            plotVariable,...
            titleString)

        % Plot pcolor for two dimensional parameter study.

        figure;
        hold on
        pcolor(gridVariable{1}, gridVariable{2},...
```

```
            plotVariable)
        shading interp
        axis tight;
        hold off
        xlabel(PlotStruct.PlotString1)
        ylabel(PlotStruct.PlotString2)

        title(titleString)
        drawnow
        xlabel(colorbar, titleString)
        obj.saveFigure(fileName);
    end

    function obj = threePlot(obj, gridVariable, PlotStruct)

        % Plot three dimensional volume slice plot of parameter study.

        obj.reshapeInputsToGridDimensions(gridVariable);

        SliceStruct = struct;
        SliceStruct.X = [min(PlotStruct.plotVariable1)...
            mean(PlotStruct.plotVariable1)];
        SliceStruct.Y = max(PlotStruct.plotVariable2);
        SliceStruct.Z = [mean(PlotStruct.plotVariable3) ...
            min(PlotStruct.plotVariable3)];

        if any(isfinite(obj.ParameterStudyFailureTime.Brittle(:)))
            obj.parameterSlice(...
                PlotStruct,...
                obj.ParameterStudyFailureTime.Brittle,...
                ['Timing of Brittle Failure (years).'],...
                'BrittleParameterStudyFailureTime',...
                SliceStruct);
        end

        if any(isfinite(obj.ParameterStudyFailureTime.Ductile(:)))
            obj.parameterSlice(...
                PlotStruct, obj.ParameterStudyFailureTime.Ductile,...
                ['Timing of Ductile Failure (years).'],...
                'DuctileParameterStudyFailureTime', SliceStruct);
        end

        obj.parameterSlice(...
            PlotStruct,...
            obj.ParameterStudyFailureTime.Stable.Failure,...
            ['Timing of Stable Sliding (years).'],...
            'StableSlidingTimeFailure',...
            SliceStruct);

        obj.parameterSlice(...
            PlotStruct,...
            obj.ParameterStudyFailureTime.Unstable.Failure,...
            ['Timing of Unstable Sliding (years).'],...
            'UnstableSlidingTimeFailure',...
            SliceStruct);

        obj.parameterSlice(...
            PlotStruct,...
            obj.ParameterStudyFailureTime.Stable.NoFailure,...
            ['Timing of Stable Sliding (years).'],...
            'StableSlidingTimeNoFailure',...
            SliceStruct);

        obj.parameterSlice(...
            PlotStruct,...
```

```
            obj.ParameterStudyFailureTime.Unstable.NoFailure,...
            ['Timing of Unstable Sliding (years).'],...
            'UnstableSlidingTimeNoFailure',...
            SliceStruct);

        nucleationTime...
            = (obj.ParameterStudyFailureTime.Unstable.Failure...
            - obj.ParameterStudyFailureTime.Stable.Failure);

        nucleationTimeNoFailure = (...
            obj.ParameterStudyFailureTime.Unstable.NoFailure...
            - obj.ParameterStudyFailureTime.Stable.NoFailure);

        obj.parameterSlice(...
            PlotStruct, nucleationTime,...
            ['Nucleation Time (years).'],...
            'NucleationTimeFailure',...
            SliceStruct);

        obj.parameterSlice(...
            PlotStruct, nucleationTimeNoFailure,...
            ['Nucleation Time (years).'],...
            'NucleationTimeNoFailure',...
            SliceStruct);
    end

    function parameterSlice(...
            obj,...
            PlotStruct,...
            plotVariable,...
            titleString,...
            fileName,...
            SliceStruct)

        % Plot two dimensional parameter study through three
        % dimensional volume plot.

        figure;
        hold on
        view(45, 45);
        slice(...
            PlotStruct.plotVariable1',...
            PlotStruct.plotVariable2',...
            PlotStruct.plotVariable3',...
            plotVariable, SliceStruct.X,...
            SliceStruct.Y, SliceStruct.Z);
        shading interp
        axis tight;
        hold off
        xlabel(PlotStruct.PlotString2)
        ylabel(PlotStruct.PlotString1)
        zlabel(PlotStruct.plotString3)

        title(...
            titleString)
        drawnow
        xlabel(colorbar, titleString)
        obj.saveFigure(fileName);
    end

    function reshapeInputsToGridDimensions(obj, gridVariable)

        % Take vector input and reshape to plot grid.

        obj.ParameterStudyFailureTime.Stable.Failure = reshape(...
```

```
            obj.ParameterStudyFailureTime.Stable.Failure,...
            size(gridVariable{1}));
        obj.ParameterStudyFailureTime.Unstable.Failure = reshape(...
            obj.ParameterStudyFailureTime.Unstable.Failure,...
            size(gridVariable{1}));
        obj.ParameterStudyFailureTime.Stable.NoFailure = reshape(...
            obj.ParameterStudyFailureTime.Stable.NoFailure,...
            size(gridVariable{1}));
        obj.ParameterStudyFailureTime.Unstable.NoFailure = reshape(...
            obj.ParameterStudyFailureTime.Unstable.NoFailure,...
            size(gridVariable{1}));
        obj.ParameterStudyFailureTime.Brittle = reshape(...
            obj.ParameterStudyFailureTime.Brittle, size(...
            gridVariable{1}));
        obj.ParameterStudyFailureTime.Ductile = reshape(...
            obj.ParameterStudyFailureTime.Ductile, size(...
            gridVariable{1}));
    end


    function parameterTransfer(obj, faultFluidFlowObj)

        % Transfer parameters representing solver tolerance.

        obj.options = faultFluidFlowObj.options;
        obj.simulatedFaultWidth...
            = faultFluidFlowObj.simulatedFaultWidth;
        obj.simulatedFaultHeight...
            = faultFluidFlowObj.simulatedFaultHeight;
        obj.horizontalArrayLength...
            = faultFluidFlowObj.horizontalArrayLength;
        obj.verticalArrayLength...
            = faultFluidFlowObj.verticalArrayLength;


    end

    function outputVector  = resultProcessing(...
            obj,...
            faultFluidFlowObj)

        % Process results for plotting at each result time step.

        obj.parameterTransfer(faultFluidFlowObj);

        FaultFluidFlowClass.printProgressString(...
            'Returning parameter study result...');

        obj.initialiseFailurePlaneStruct(...
            faultFluidFlowObj);

        obj.initialiseFailureLengthVariables;
        stableSlidingTrigger = false;
        faultFluidFlowObj.slidingStress = NaN(...
            obj.verticalArrayLength,...
            obj.horizontalArrayLength);

        for loopTimeCounter = 1:length(faultFluidFlowObj.time)
            loopTime = faultFluidFlowObj.time(loopTimeCounter);

            if ~obj.processingEarthquakeTrigger
                obj.postProcessingPhysicalVariables(...
                    faultFluidFlowObj,...
                    loopTimeCounter,...
                    loopTime);
```

```
                if obj.brittleFailureCondition
                    obj.FailureTime.Brittle...
                        = faultFluidFlowObj.time(...
                        loopTimeCounter);
                    obj.brittleFailureTimeTrigger...
                        = true;
                end

                if obj.ductileFailureCondition
                    obj.FailureTime.Ductile...
                        = faultFluidFlowObj.time(...
                        loopTimeCounter);
                    obj.ductileFailureTimeTrigger...
                        = true;
                end

                if obj.onsetOfStableSlidingCondition(...
                        faultFluidFlowObj,...
                        loopTimeCounter)...
                        && ~stableSlidingTrigger

                    obj.FailureTime.Stable...
                        = faultFluidFlowObj.time(loopTimeCounter);

                    obj.SlidingLength.Failure.Stable...
                        = faultFluidFlowObj.EarthquakeLengthVector(...
                        loopTimeCounter).Failure;

                    obj.SlidingLength.Nucleation.Stable...
                        = faultFluidFlowObj.EarthquakeLengthVector(...
                        loopTimeCounter).Nucleation;

                    obj.FailureExtent.Stable =...
                        faultFluidFlowObj.failureExtent;

                    stableSlidingTrigger = true;
                end

                if obj.unstableSlidingCondition(...
                        faultFluidFlowObj,...
                        loopTimeCounter)
                    obj.processingEarthquakeTrigger...
                        = true;

                    obj.FailureTime.Unstable...
                        = faultFluidFlowObj.time(...
                        loopTimeCounter);

                    obj.SlidingLength.Failure.Unstable...
                        = faultFluidFlowObj.EarthquakeLengthVector(...
                        loopTimeCounter).Failure;

                    obj.SlidingLength.Nucleation.Unstable...
                        = faultFluidFlowObj.EarthquakeLengthVector(...
                        loopTimeCounter).Nucleation;

                    obj.FailureExtent.Unstable =...
                        faultFluidFlowObj.failureExtent;
                end
            end
        end

        faultFluidFlowObj.calculateAnalyticalTime;
```

```
            outputVector = obj.parameterStudyOutputVector(...
                faultFluidFlowObj);
        end

        function obj = initialiseFailureLengthVariables(obj)

            % Initialise variables for storing failure lengths.

            obj.SlidingLength.Failure.Stable = NaN;
            obj.SlidingLength.Failure.Unstable = NaN;
            obj.SlidingLength.Nucleation.Stable = NaN;
            obj.SlidingLength.Nucleation.Unstable = NaN;
        end

        function outputVector = parameterStudyOutputVector(...
                obj,...
                faultFluidFlowObj)

            % Create output vector from a single parameter's study.

            timeUnit = FaultFluidFlowClass.SECONDS_PER_YEAR;

            outputVector = [...
                obj.FailureTime.Brittle / timeUnit,...
                obj.FailureTime.Ductile / timeUnit,...
                obj.FailureTime.Stable / timeUnit,...
                obj.FailureTime.Unstable / timeUnit,...
                faultFluidFlowObj.analyticalTime / timeUnit,...
                obj.SlidingLength.Failure.Stable,...
                obj.SlidingLength.Failure.Unstable,...
                obj.SlidingLength.Nucleation.Stable,...
                obj.SlidingLength.Nucleation.Unstable,...
                obj.FailureExtent.Stable,...
                obj.FailureExtent.Unstable];
        end

        function [p1, p2] = addFailureRectanglesToPlot(...
                obj,...
                PlotStruct, ...
                varargin)

            % Add rectangles to parameter study plot indicating mode of
            % failure.

            Y.max = max(max([varargin{:}]));
            Y.min = min(min([varargin{:}]));
            p1 = ParameterStudyPlottingClass.failureRectangle(...
                obj.ParameterStudyFailureTime.Brittle,...
                PlotStruct,...
                Y,...
                'k');
            p2 = ParameterStudyPlottingClass.failureRectangle(...
                obj.ParameterStudyFailureTime.Ductile,...
                PlotStruct,...
                Y,...
                'r');
        end
    end

    methods(Static)
        function  p = failureRectangle(...
                ParameterStudyFailureTime,...
                PlotStruct,...
                Y,...
                rectangleColor)
```

```
            % Add rectangle indicating failure to line plot.

            X = PlotStruct.plotVariable1(...
                isfinite(...
                ParameterStudyFailureTime));

            p = [];

            if ~isempty(X) > 0
                p = patch(...
                    [min(X) max(X) max(X) min(X)],...
                    [Y.min Y.min Y.max Y.max],...
                    rectangleColor,...
                    'FaceAlpha',...
                    0.25,...
                    'EdgeColor',...
                    'none');
            end
        end

        function legcell = plotFailureBox(...
                ParameterStudyFailureTime,...
                PlotStruct,...
                legcell,...
                legString,...
                plotType)

            % Add failure box to plot.

            if ~all(isnan(ParameterStudyFailureTime(:)))
                plot(...
                    PlotStruct.plotVariable1,...
                    ParameterStudyFailureTime,...
                    plotType{:})
                legcell = {legcell{:}, legString};
            end
        end

        function imageCell = resizeImagesToFirstImage(imageCell)
            sizeTemp = size(imageCell{1, 1});
            for cellCounter1 = 1:size(imageCell, 1)
                for cellCounter2 = 1:size(imageCell, 2)
                    imageCell{cellCounter1, cellCounter2}...
                        = imresize(...
                        imageCell{...
                        cellCounter1, cellCounter2}, sizeTemp(1:2));
                end
            end
        end
    end
end
```

```
ans =

  ParameterStudyPlottingClass with properties:

                  FailureTime: [1×1 struct]
                FailureExtent: [1×1 struct]
      ParameterStudyFailureTime: [1×1 struct]
    ParameterStudyFailureExtent: [1×1 struct]
                    PatchSize: []
                analyticalTime: []
                 SlidingLength: []
```

```
                poreFluidFactorList: []
                tectonicLoadingRate: []
                          faultDepth: []
                  confinementFactor: []
                 overpressureHeight: []
                           OFCwidth: []
                           IFCwidth: []
                  failureStringCell: {'Failure'  'NoFailure'}
                            options: []
                 simulatedFaultWidth: []
                simulatedFaultHeight: []
               horizontalArrayLength: []
                 verticalArrayLength: []
              EarthquakeLengthVector: []
                    StabilityLengths: []
                       FailureMarker: []
                slidingFailureMarker: []
                       oldFailureMap: []
                            pressure: []
                              stress: []
                     effectiveStress: []
                      outputPressure: []
                          MohrCircle: [1×1 struct]
                       MohrGeometry: [1×1 struct]
                               time: []
      effectiveNormalStressForPlot: []
         EffectiveNormalStressStep: 10000
       processingEarthquakeTrigger: 0
         brittleFailureTimeTrigger: 0
         ductileFailureTimeTrigger: 0
                steadyStateTrigger: 0
                brittleFailureTime: NaN
                ductileFailureTime: NaN
                  SubplotFileNames: [1×1 struct]
                   mohrFigureScale: []
                     plotTimeScale: []
                     decimalPlaces: 1
                       limitYValue: 40
                   pressureSubplot: []
                       mohrSubplot: []
                     stressSubplot: []
            pressureSubplotElement: []
                mohrSubplotElement: []
              stressSubplotElement: []
                       legendVector: []
                         folderName: []
                   folderCheckFlag: 1
                       plotCounter: 0
        effectiveStressPatchVector: []
              effectiveStressPatch: []
                     computerStore: []
                      lowMohrLimit: []
                      lastPressure: []
                  steadyStateLimit: 1.0000e-12
                     gaussianWidth: 1.2500
          lastSlidingFailureMarker: []
                 lastFailureMarker: []
```

---

*Published with MATLAB® R2018b*

```
classdef ResultPlottingClass < handle
    properties
        horizontalArrayLength;
        verticalArrayLength;
        EarthquakeLengthVector;
        StabilityLengths
        FailureMarker;
        slidingFailureMarker;
        oldFailureMap;
        pressure;
        stress;
        effectiveStress;
        outputPressure;
        MohrCircle = struct;
        MohrGeometry = struct('Radius', struct,'Geometry', struct);
        time;

        effectiveNormalStressForPlot;
        EffectiveNormalStressStep = 1E4;
        processingEarthquakeTrigger = false;
        brittleFailureTimeTrigger = false;
        ductileFailureTimeTrigger = false;
        steadyStateTrigger = false;
        brittleFailureTime = NaN;
        ductileFailureTime = NaN;

        SubplotFileNames = struct;
        mohrFigureScale;
        plotTimeScale;
        decimalPlaces = 1;
        limitYValue = 40;
        pressureSubplot;
        mohrSubplot;
        stressSubplot;
        pressureSubplotElement;
        mohrSubplotElement;
        stressSubplotElement;
        legendVector;
        folderName;
        folderCheckFlag = true;
        plotCounter = 0;
        effectiveStressPatchVector;
        effectiveStressPatch;
        computerStore;
        lowMohrLimit;
        lastPressure = [];
        steadyStateLimit = 1E-12;
        gaussianWidth = 1.25;
        lastSlidingFailureMarker;
        lastFailureMarker;
    end

    methods
        function initialise(obj, faultFluidFlowObj, folderName)

            % Initialise instance of result plotting class.

            obj.folderName = folderName;

            obj.initialiseFailurePlaneVariables(faultFluidFlowObj);
        end

        function initialiseFailurePlaneVariables(obj, faultFluidFlowObj)
```

```
            % Initialise failure plane variables.

            obj.lowMohrLimit = 0;

            obj.effectiveNormalStressForPlot =...
                obj.lowMohrLimit:obj.EffectiveNormalStressStep:...
                obj.roundToMPa(...
                faultFluidFlowObj.lithostaticStress...
                - faultFluidFlowObj.hydrostaticStress);

            obj.computerStore = computer;
            obj.oldFailureMap = false(...
                faultFluidFlowObj.verticalArrayLength,...
                faultFluidFlowObj.horizontalArrayLength);

            obj.initialiseFailurePlaneStruct(faultFluidFlowObj);
        end

        function obj = caseStudyFigures(obj, faultFluidFlowObj)

            % Plot figures for case study.

            faultFluidFlowObj.printProgressString(...
                'Plotting and saving results...');

            obj.initialiseFailurePlaneStruct(faultFluidFlowObj);
            obj.resultProcessing(faultFluidFlowObj);
            obj.faultPlaneFailurePlot(faultFluidFlowObj);
        end

        function obj = resultProcessing(obj, faultFluidFlowObj)

            % Process results for plotting at each result time step.

            stableSlidingTrigger = false;
            faultFluidFlowObj.slidingStress = NaN(...
                faultFluidFlowObj.verticalArrayLength,...
                faultFluidFlowObj.horizontalArrayLength);

            for loopCounter = 1:length(faultFluidFlowObj.time)
                loopTime = faultFluidFlowObj.time(loopCounter);

                faultFluidFlowObj = obj.postProcessingPhysicalVariables(...
                    faultFluidFlowObj,...
                    loopCounter,...
                    loopTime);

                if ~obj.processingEarthquakeTrigger
                    if ResultPlottingClass.initialTimeCondition(...
                            loopCounter)
                        obj.printFigure(faultFluidFlowObj, loopTime);
                    end

                    if obj.onsetOfStableSlidingCondition(...
                            faultFluidFlowObj,...
                            loopTime)...
                            && ~stableSlidingTrigger
                        obj.printFigure(faultFluidFlowObj, loopTime);
                        stableSlidingTrigger = true;
                    end

                    if obj.unstableSlidingCondition(...
                            faultFluidFlowObj,...
                            loopTime)
```

```
                    obj.printFigure(faultFluidFlowObj, loopTime);
                    obj.processingEarthquakeTrigger = true;
                end

                if obj.brittleFailureCondition
                    obj.printFigure(faultFluidFlowObj, loopTime);
                    obj.brittleFailureTimeTrigger = true;
                end

                if obj.ductileFailureCondition
                    obj.printFigure(faultFluidFlowObj, loopTime);
                    obj.ductileFailureTimeTrigger = true;
                end
            end
        end
    end

    function obj = printFigure(obj, faultFluidFlowObj, time)

        % Print result figure.

        obj.plotCounter = obj.plotCounter + 1;

        obj.pressureFigure(faultFluidFlowObj, time);
        obj.mohrFigure(faultFluidFlowObj, time);
    end

    function outputFigure = pressureFigure(...
            obj,...
            faultFluidFlowObj,...
            time,...
            varargin)

        % Plot pressure figure and save.

        outputFigure = figure;

        [C, h] = contourf(...
            faultFluidFlowObj.x,...
            faultFluidFlowObj.z,...
            obj.pressure / 1E6);
        clabel(C,h);

        shading interp;
        colorbar;
        axis tight;

        if isnan(faultFluidFlowObj.confinementFactor)
            faultFluidFlowObj.confinementFactor...
                = max(max(...
                obj.pressure / faultFluidFlowObj.lithostaticStress));
        end

        hold on;
        colorbar;

        limy = get(gca, 'YLim');
        ylim([limy(1) obj.limitYValue])

        xlabel('x [m]')
        ylabel('y [m]')
        xlabel(colorbar, 'Pressure [MPa]')
        if strcmp(faultFluidFlowObj.plotTimeScale, 'years')
            title(['Absolute pressure at ', num2str(round(time / ...
                FaultFluidFlowClass.SECONDS_PER_YEAR,...
```

```
                obj.decimalPlaces)), ' years'])
        elseif strcmp(faultFluidFlowObj.plotTimeScale, 'days')
            title(['Absolute pressure at ', num2str(round(time / ...
                FaultFluidFlowClass.SECONDS_PER_DAY,...
                obj.decimalPlaces)), ' days'])
        end

        if any(...
                strcmp(...
                faultFluidFlowObj.faultArchitectureList, 'OFC'))
        elseif any(...
                strcmp(...
                faultFluidFlowObj.faultArchitectureList, 'Fracture'))
            ResultPlottingClass.markFracture(faultFluidFlowObj);
        elseif any(...
                strcmp(...
                faultFluidFlowObj.faultArchitectureList,...
                'PSZ'))
            ResultPlottingClass.markFault(faultFluidFlowObj)
        end

                obj.failureContour(...
                    faultFluidFlowObj,...
                    obj.gaussianWidth);

        hold off
        drawnow

        if isempty(varargin)
            counter = num2str(obj.plotCounter);
        else
            counter = varargin{:};
        end

        obj.saveFigure(...
            ['PressureFigure' counter]);
    end

    function outputFigure = stressFigure(obj, faultFluidFlowObj, time)

        % Plot stress figure and save.

        outputFigure = figure;

        quiver(...
            faultFluidFlowObj.x,...
            faultFluidFlowObj.z,...
            obj.stress(:, :, 1),...
            obj.stress(:, :, 2));

        axis tight;

        xlabel('x [m]')
        ylabel('y [m]')

        if strcmp(faultFluidFlowObj.plotTimeScale, 'years')
            title(['Stress at ', num2str(round(time / ...
                FaultFluidFlowClass.SECONDS_PER_YEAR,...
                obj.decimalPlaces)), ' years'])
        elseif strcmp(faultFluidFlowObj.plotTimeScale, 'days')
            title(['Stress at ', num2str(round(time / ...
                FaultFluidFlowClass.SECONDS_PER_DAY,...
                obj.decimalPlaces)), ' days'])
        end
```

```
            if any(...
                    strcmp(...
                    faultFluidFlowObj.faultArchitectureList, 'OFC'))
                ResultPlottingClass.markOFCBoundary(faultFluidFlowObj);
            elseif any(...
                    strcmp(...
                    faultFluidFlowObj.faultArchitectureList, 'Fracture'))
                ResultPlottingClass.markFracture(faultFluidFlowObj);
            elseif any(...
                    strcmp(...
                    faultFluidFlowObj.faultArchitectureList, 'PSZ'))
                ResultPlottingClass.markFault(faultFluidFlowObj)
            end

            switch faultFluidFlowObj.contourPlotMode
                case 'failure'
                    obj.failureContour(...
                        faultFluidFlowObj,...
                        obj.gaussianWidth);
                otherwise
                    error('Contour plot mode not recognised.');
            end
            hold off
            drawnow

            obj.saveFigure(['StressFigure' num2str(...
                obj.plotCounter)]);
        end

        function failureContour(...
                obj,...
                faultFluidFlowObj,...
                gaussianWidth)

            % Add local failure contour to pressure plot.

            if ~all(obj.FailureMarker.Central(:) == 0)
                Zsmooth1 = imgaussfilt(...
                    obj.FailureMarker.Central,...
                    gaussianWidth);
                contour(...
                    faultFluidFlowObj.x,...
                    faultFluidFlowObj.z,...
                    obj.FailureMarker.Central,...
                    1,...
                    'w');
            end
        end

        function obj = mohrFigure(obj, faultFluidFlowObj, time)

            % Plot Mohr analysis figure.

            obj.mohrPlotVariables(faultFluidFlowObj, time);
            obj.setMohrFigureScale(faultFluidFlowObj);

            figure;
            hold on;
            axis('equal')
            xlim([min(obj.effectiveNormalStressForPlot),...
                max(obj.effectiveNormalStressForPlot)])
            ylim([0, obj.mohrFigureScale]);

            obj.legendVector = zeros(...
                size(...
```

```
            faultFluidFlowObj.faultArchitectureList));

        if ~faultFluidFlowObj.modeOfFailureFlag
            for loopCounter = 1:length(...
                    faultFluidFlowObj.faultArchitectureList)
                architectureComponent...
                    = faultFluidFlowObj.faultArchitectureList{...
                    loopCounter};
                if faultFluidFlowObj.ModeOfFailureArchitectureFlag.(...
                        architectureComponent)
                    faultFluidFlowObj.PlotProperties.(...
                        architectureComponent)...
                        = {'--', faultFluidFlowObj.PlotProperties.(...
                        architectureComponent){:}};
                end
            end
        end

        for loopCounter = 1:length(...
                faultFluidFlowObj.faultArchitectureList)
            architectureComponent =...
                faultFluidFlowObj.faultArchitectureList{...
                loopCounter};

            plot(obj.effectiveNormalStressForPlot,...
                faultFluidFlowObj.FailureEnvelope.(...
                architectureComponent));

            obj.legendVector(loopCounter) = plot(...
                obj.effectiveNormalStressForPlot,...
                obj.MohrCircle.(architectureComponent));
        end

        hold off
        ResultPlottingClass.convertToMpa(gca);
        xlabel('Effective normal stress [MPa]')
        ylabel('Shear stress [MPa]')

        if strcmp(faultFluidFlowObj.plotTimeScale, 'years')
            title(['Mohr failure envelope at '  ...
                , num2str(round(time /...
                FaultFluidFlowClass.SECONDS_PER_YEAR,...
                obj.decimalPlaces)), ' years'])

        elseif strcmp(faultFluidFlowObj.plotTimeScale, 'days')
            title(['Mohr failure envelope at '  ...
                , num2str(round(time /...
                FaultFluidFlowClass.SECONDS_PER_DAY,...
                obj.decimalPlaces)), ' days'])
        end

        for loopCounter = 1:length(...
                faultFluidFlowObj.faultArchitectureList)
            if faultFluidFlowObj.SlidingFailureFlag.(...
                    faultFluidFlowObj.faultArchitectureList{...
                    loopCounter})
                obj.plotFailurePlane(...
                    faultFluidFlowObj,...
                    faultFluidFlowObj.faultArchitectureList{...
                    loopCounter});
            end
        end

        legend(...
            obj.legendVector,...
```

```
            faultFluidFlowObj.faultArchitectureList);
        drawnow

        obj.saveMohrFigure;
    end

    function setMohrFigureScale(obj, faultFluidFlowObj)

        % Set Mohr figure plot scaling based on preset.

                obj.localFailureMohrFigureScaling(faultFluidFlowObj);

    end

    function localFailureMohrFigureScaling(obj, faultFluidFlowObj)

        % Set Mohr figure scale for local failure.

        obj.mohrFigureScale = 1.5...
            * max(faultFluidFlowObj.FailureEnvelope.OFC);
    end

    function saveMohrFigure(obj, varargin)

        % Save a Mohr plot.

        if isempty(varargin)
            counter = num2str(obj.plotCounter);
        else
            counter = varargin{:};
        end

        obj.saveFigure(['MohrFigure' counter]);
    end

    function obj = mohrPlotVariables(obj, faultFluidFlowObj, time)

        % Generate variables for Mohr plot.

        faultFluidFlowObj.FailureEnvelope = struct;

        for loopCounter = 1:length(...
                faultFluidFlowObj.faultArchitectureList)

            architectureComponent =...
                faultFluidFlowObj.faultArchitectureList{...
                loopCounter};

            ResultPlottingClass.failureEnvelopeForPlot(...
                faultFluidFlowObj,...
                obj.effectiveNormalStressForPlot,...
                architectureComponent);

            obj.mohrCircle(...
                faultFluidFlowObj,...
                time,...
                architectureComponent);
        end
    end

    function obj = mohrCircle(...
            obj,...
            faultFluidFlowObj,...
            time,...
            architectureComponent)
```

```
            % Calculate values for Mohr circle.

            faultFluidFlowObj.calculateMaximumStress;
            faultFluidFlowObj.calculateMinimumStress(...
                time);

            obj.MohrCircleGeometry(...
                faultFluidFlowObj,...
                obj.pressure,...
                architectureComponent);

            obj.MohrCircle.(architectureComponent) =...
                zeros(size(obj.effectiveNormalStressForPlot));
            for loopCounter = 1:length(obj.MohrGeometry.Radius.(...
                    architectureComponent))
                obj.MohrCircle.(architectureComponent) =...
                    FaultFluidFlowClass.semiCircleHeight(...
                    obj.effectiveNormalStressForPlot,...
                    obj.MohrGeometry.Radius.(architectureComponent),...
                    obj.MohrGeometry.Centre.(architectureComponent));
            end
            obj.MohrCircle.(architectureComponent)(...
                obj.MohrCircle.(architectureComponent) <= 0) = NaN;
        end

        function obj = MohrCircleGeometry(...
                obj,...
                faultFluidFlowObj,...
                pressure,...
                architectureComponent)

            % Calculate the geometry of the Mohr Circle at every spatial
            % array point.

            tempPressure =...
                pressure(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central);

            arrayDims = ndims(tempPressure);

            if any(size(tempPressure) == 0)
                arrayDims = arrayDims - 1;
            end

            if arrayDims == 1
                maxPressure = max(tempPressure);

            elseif arrayDims == 2
                maxPressure = max(max(tempPressure));

            end

            mohrPressure = maxPressure;

            minimumEffectiveStress...
                = faultFluidFlowObj.minimumStress(...
                pressure == maxPressure)...
                - mohrPressure;
            minimumEffectiveStress = minimumEffectiveStress(1);

            maximumEffectiveStress...
                = faultFluidFlowObj.maximumStress(...
                pressure == maxPressure)...
```

```
            - mohrPressure;

        maximumEffectiveStress = maximumEffectiveStress(1);

        obj.MohrGeometry.Radius.(architectureComponent)...
            = (maximumEffectiveStress...
            - minimumEffectiveStress) / 2;

        obj.MohrGeometry.Centre.(architectureComponent)...
            = (maximumEffectiveStress...
            + minimumEffectiveStress) / 2;

        if faultFluidFlowObj.SlidingFailureFlag.(...
                architectureComponent)
            if isfinite(obj.effectiveStressPatch) == 1
                obj.MohrGeometry.Radius.(architectureComponent)...
                    = faultFluidFlowObj.FrictionCoefficient.Brittle.(...
                    architectureComponent) * obj.MohrGeometry.Centre.(...
                    architectureComponent)...
                    ./ (faultFluidFlowObj.FrictionCoefficient.Brittle.(...
                    architectureComponent)...
                    * cosd(2 * faultFluidFlowObj.faultAngle)...
                    - sind(2 * faultFluidFlowObj.faultAngle));
            end
        end
    end

    function faultFluidFlowObj = postProcessingPhysicalVariables(...
            obj,...
            faultFluidFlowObj,...
            loopCounter,...
            loopTime)

        % Extract and calculate physical variables from simulation
        % results.

        faultFluidFlowObj = obj.processFailureMarkerResult(...
            faultFluidFlowObj,...
            loopCounter);

        faultFluidFlowObj = obj.processSlidingFailureMarkerResult(...
            faultFluidFlowObj,...
            loopCounter);

        obj.processPressureResult(faultFluidFlowObj, loopCounter);

        obj.pressure = faultFluidFlowObj.pressureBCS(...
            obj.pressure);

        [~,...
            ~,...
            ~,...
            ~,...
            failureLength,...
            nucleationLength,...
            ~]...
            = faultFluidFlowObj.rockMatrixState(...
            obj.pressure,...
            loopTime);

        if (nucleationLength - failureLength)...
                / nucleationLength...
                < (faultFluidFlowObj.options.RelTol)
            failureLength = nucleationLength;
        end
```

```
            faultFluidFlowObj.EarthquakeLengthVector(...
                loopCounter).Failure = failureLength;
            faultFluidFlowObj.EarthquakeLengthVector(...
                loopCounter).Nucleation = nucleationLength;
        end

        function processPressureResult(...
                obj,...
                faultFluidFlowObj,...
                timeLoopCounter)

            % Select pressure from solver output variable at a given time
            % and reshape to spatial dimensions.

            obj.pressure = faultFluidFlowObj.outputPressure(...
                :,...
                timeLoopCounter);

            obj.pressure = reshape(...
                obj.pressure,...
                faultFluidFlowObj.verticalArrayLength,...
                faultFluidFlowObj.horizontalArrayLength);
        end

        function faultFluidFlowObj = processFailureMarkerResult(...
                obj,...
                faultFluidFlowObj,...
                loopCounter)

            % Retrieve failure marker from results at a given timestep.

            obj.FailureMarker = faultFluidFlowObj.FailureMarkerStore(...
                loopCounter);

            if loopCounter > 1
                faultFluidFlowObj.FailureMarker...
                    = faultFluidFlowObj.FailureMarkerStore(...
                    :,...
                    :,...
                    loopCounter - 1);
            else
                faultFluidFlowObj.FailureMarker...
                    = faultFluidFlowObj.FailureMarkerStore(...
                    :,...
                    :,...
                    loopCounter);
            end

            faultFluidFlowObj = obj.horizontalFailureExtent(...
                faultFluidFlowObj);

        end

        function faultFluidFlowObj = processSlidingFailureMarkerResult(...
                obj,...
                faultFluidFlowObj,...
                loopCounter)

            % Retrieve sliding failure marker from results at a given
            % timestep.

            obj.slidingFailureMarker...
                = faultFluidFlowObj.slidingFailureMarkerStore(...
                :, :, loopCounter);
```

```
            if loopCounter > 1
                faultFluidFlowObj.slidingFailureMarker...
                    = faultFluidFlowObj.slidingFailureMarkerStore(...
                    :, :, loopCounter - 1);

                faultFluidFlowObj.slidingStress...
                    = faultFluidFlowObj.slidingStressStore(...
                    :, :, loopCounter - 1);
            else
                faultFluidFlowObj.slidingFailureMarker...
                    = faultFluidFlowObj.slidingFailureMarkerStore(...
                    :, :, loopCounter);

                faultFluidFlowObj.slidingStress...
                    = faultFluidFlowObj.slidingStressStore(...
                    :, :, loopCounter);
            end

        end

        function obj = faultPlaneFailurePlot(...
                obj,...
                faultFluidFlowObj,...
                varargin)

            % Plot state of failure on the fault plane.

            failureLengthVector...
                = [faultFluidFlowObj.EarthquakeLengthVector(:).Failure];
            nucleationLengthVector...
                = [faultFluidFlowObj.EarthquakeLengthVector(:).Nucleation];

            if(any(isfinite([...
                    faultFluidFlowObj.EarthquakeLengthVector(:).Failure])))
                figure;
                hold on;
                plot(faultFluidFlowObj.time...
                    / faultFluidFlowObj.SECONDS_PER_YEAR,...
                    nucleationLengthVector, 'r');
                    plot(faultFluidFlowObj.time...
                    / faultFluidFlowObj.SECONDS_PER_YEAR,...
                    failureLengthVector, 'k');
                hold off
                legend({'Nucleation length [m]',...
                    'Shear Failure length [m]'},...
                    'Location',...
                    'southeast');
                colormap jet;
                xlabel('Time [years]');
                ylabel('Length [m]');
                title(['Failure lengths (PSZ)']);
                if isfinite(obj.brittleFailureTime) ...
                        && ~obj.processingEarthquakeTrigger
                    line([obj.brittleFailureTime obj.brittleFailureTime]...
                        / Constants.SECONDS_PER_YEAR, ...
                        get(gca, 'ylim'), ...
                        'Color', 'red',...
                        'LineStyle', '--');
                end

                if isfinite(obj.ductileFailureTime) == 1 ...
                        && ~obj.processingEarthquakeTrigger
                    line([obj.ductileFailureTime...
                        obj.ductileFailureTime]...
```

```
                    / Constants.SECONDS_PER_YEAR,...
                    get(gca, 'ylim'), ...
                    'Color', 'black',...
                    'LineStyle', '--');
            end
            drawnow

            if isempty(varargin)
                counter = '';
            else
                counter = varargin{:};
            end

            obj.saveFigure(['FaultPlaneFailure', num2str(counter)])
        end
    end

    function nucleationLength = nucleationLength(...
            obj,...
            faultFluidFlowObj)

        % Calculate nucleation length stability criterion.

        nucleationLength = faultFluidFlowObj.psi...
            * faultFluidFlowObj.shearModulus...
            * faultFluidFlowObj.criticalSlipDistance...
            ./ (obj.effectiveStressPatch...
            * faultFluidFlowObj.rateAndStateDifference);
    end

    function plotFailurePlane(...
            obj,...
            faultFluidFlowObj,...
            architectureComponent)

        % Plot failure and nucleation length.

        faultAngle = faultFluidFlowObj.faultAngle;

        line([...
            obj.MohrGeometry.Centre.(architectureComponent)...
            obj.effectiveStressPatch],...
            [...
            0 ...
            obj.MohrGeometry.Radius.(architectureComponent)...
            * sind(2 * faultAngle)],...
            'Color',...
            'black');
    end

    function obj = initialiseFailurePlaneStruct(...
            obj,...
            faultFluidFlowObj)

        % Initialise struct to hold failure plane parameters.

        faultFluidFlowObj.EarthquakeLengthVector = repmat(...
            struct(...
            'Failure', NaN,...
            'Nucleation', NaN),...
            length(faultFluidFlowObj.time),...
            1);
    end

    function condition = onsetOfStableSlidingCondition(...
```

```
            obj,...
            faultFluidFlowObj,...
            loopTimeCounter)

        % Condition for fault to begin stable sliding.

        condition = any(obj.slidingFailureMarker(:))...
            || obj.unstableSlidingCondition(...
            faultFluidFlowObj,...
            loopTimeCounter);
    end

    function condition = unstableSlidingCondition(...
            obj,...
            faultFluidFlowObj,...
            loopTimeCounter)

        % Condition for fault to begin unstable sliding.

        condition = loopTimeCounter == length(faultFluidFlowObj.time);
    end

    function condition = steadyStateCondition(obj)

        % Condition for fluid flow to have reached an approximate
        % steady state.

        condition = false;

        if ~isempty(obj.lastPressure)
            meanPressureChange = mean(...
                mean(...
                obj.pressure - obj.lastPressure));

            if (obj.steadyStateLimit...
                    > meanPressureChange / mean(mean(obj.pressure)))...
                    && obj.steadyStateTrigger ~= 1

                condition = true;

            end
        end

        obj.lastPressure = obj.pressure;
    end

    function condition = brittleFailureCondition(obj)

        % Condition for fault zone to undergo brittle failure at any
        % spatial array point.

        condition = (isnan(obj.brittleFailureTime)...
            && ~obj.brittleFailureTimeTrigger...
            && (any(obj.FailureMarker.X(:) == 1)...
            || any(obj.FailureMarker.Z(:) == 1)))...
            && obj.processingEarthquakeTrigger ~= 1;
    end

    function condition = ductileFailureCondition(obj)

        % Condition for fault zone to undergo ductile failure at any
        % spatial array point.

        condition = (isnan(obj.ductileFailureTime)...
            && obj.ductileFailureTimeTrigger ~= 1 ...
```

```
                && (any(obj.FailureMarker.X(:) == 2)...
                || any(obj.FailureMarker.Z(:) == 2)))...
                && obj.processingEarthquakeTrigger ~= 1;
        end

        function saveFigure(obj, fileName)

            % Save .fig and .tiff copies of a figure.

            saveas(gcf, [obj.folderName '/' fileName], 'fig');
            saveas(gcf, [obj.folderName '/' fileName], 'tiff');
        end
    end

    methods(Static)
        function faultFluidFlowObj = horizontalFailureExtent(...
                faultFluidFlowObj)

            % Record extent of horizontal failure.

            faultFluidFlowObj.failureExtent = max(...
                faultFluidFlowObj.x(...
                faultFluidFlowObj.FailureMarker.Central ~= 0))...
                / faultFluidFlowObj.OFCwidth;

            if isempty(faultFluidFlowObj.failureExtent)
               faultFluidFlowObj.failureExtent = 0;
            end
        end

        function failureMap = failureMap(...
                faultFluidFlowObj,...
                shearStrengthExcess,...
                architectureComponent)

            % Map failure on fault plane.

            failureMap...
                = faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central;
            failureMap(shearStrengthExcess > 0) = false;

            failureMap(isnan(failureMap)) = 0;
        end

        function failureEnvelopeForPlot =...
                failureEnvelopeForPlot(...
                faultFluidFlowObj,...
                effectiveNormalStress,...
                architectureComponent)

            % Return failure envelope for plotting.

            failureEnvelopeForPlot(effectiveNormalStress...
                <= faultFluidFlowObj.FailureModeBoundaryStress.(...
                architectureComponent))...
                = faultFluidFlowObj.FrictionCoefficient.Brittle.(...
                architectureComponent)...
                * effectiveNormalStress(effectiveNormalStress...
                <= faultFluidFlowObj.FailureModeBoundaryStress.(...
                architectureComponent))...
                + faultFluidFlowObj.Cohesion.Brittle.(...
                architectureComponent);

            failureEnvelopeForPlot(effectiveNormalStress...
```

```
            > faultFluidFlowObj.FailureModeBoundaryStress.(...
            architectureComponent))...
            = faultFluidFlowObj.FrictionCoefficient.Ductile.(...
            architectureComponent)...
            * effectiveNormalStress(effectiveNormalStress...
            > faultFluidFlowObj.FailureModeBoundaryStress.(...
            architectureComponent))...
            + faultFluidFlowObj.Cohesion.Ductile.(...
            architectureComponent);

        failureEnvelopeForPlot(failureEnvelopeForPlot < 0) = 0;

        faultFluidFlowObj.FailureEnvelope.(architectureComponent)...
            = failureEnvelopeForPlot;
    end

    function failureLength = failureLength(...
            faultFluidFlowObj,...
            shearStrengthExcess,...
            failureMap)

        % Calculate failure length.

        if all(~failureMap)
            failureLength = NaN;
        else
            failureHeight...
                = max(max(faultFluidFlowObj.z(failureMap)))...
                - min(min(faultFluidFlowObj.z(failureMap)));
            failureWidth...
                = max(max(faultFluidFlowObj.x(failureMap)))...
                - min(min(faultFluidFlowObj.x(failureMap)));

            failureLength...
                = 2 * (failureHeight .^ 2 ...
                + failureWidth .^ 2) .^ 0.5;

            if failureWidth == 0
                [~, failureEnd] = max(faultFluidFlowObj.z(failureMap));
                [~, failurePosition] = max(...
                    max(faultFluidFlowObj.x(failureMap)));

                if failureEnd == faultFluidFlowObj.verticalArrayLength
                    failureInterp = 0;
                else
                    failureInterp =...
                        shearStrengthExcess(failureEnd, failurePosition)...
                        * (faultFluidFlowObj.z(failureEnd + 1, failurePosition)...
                        - faultFluidFlowObj.z(failureEnd, failurePosition))...
                        / (shearStrengthExcess(failureEnd + 1, failurePosition)...
                        - shearStrengthExcess(failureEnd, failurePosition));

                    failureInterp(isnan(failureInterp)) = 0;
                end
                failureLength = failureLength + 2 * failureInterp;
            end
        end
    end

    function EffectiveNormalStress = effectiveNormalStress(...
            faultFluidFlowObj,...
            pressure,...
            EffectiveNormalStress,...
            angle,...
            architectureComponent)
```

```
            % Calculate effective normal stress.

            if numel(angle) == 1
                angle = angle * ones(size(pressure));
            end

            EffectiveNormalStress.Central(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central) = 0.5...
                .* ((faultFluidFlowObj.maximumStress(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                - pressure(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central))...
                + (faultFluidFlowObj.minimumStress(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                - pressure(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)))...
                + 0.5 .* (faultFluidFlowObj.maximumStress(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                - (faultFluidFlowObj.minimumStress(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)))...
                .* cosd(...
                2 .* angle(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central));

            EffectiveNormalStress.X =...
                (EffectiveNormalStress.Central(:, 1:end-1)...
                + EffectiveNormalStress.Central(:, 2:end)) / 2;
            EffectiveNormalStress.Z =...
                (EffectiveNormalStress.Central(1:end-1, :)...
                + EffectiveNormalStress.Central(2:end, :)) / 2;
        end

        function shearStrengthExcess =...
                shearStrengthExcess(...
                faultFluidFlowObj,...
                EffectiveNormalStress,...
                architectureComponent,...
                shearStrengthExcess)

            % Shear stress exceeding failure envelope.

            shearStrengthExcess(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central) =...
                faultFluidFlowObj.FrictionCoefficient.Brittle.(...
                architectureComponent)...
                * EffectiveNormalStress.Central(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                - 0.5 * (faultFluidFlowObj.maximumStress(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central)...
                - faultFluidFlowObj.minimumStress(...
                faultFluidFlowObj.ArrayFaultArchitectureMap.(...
                architectureComponent).Central))...
                * sind(2 * faultFluidFlowObj.faultAngle);
```

```
        end

        function [pressure, velocity, displacement]...
                = resultVectorToVars(faultFluidFlowObj, pressureVelocity)

            % Split and reshape solver vector to three variables.

            pressure =  pressureVelocity(...
                1:(...
                faultFluidFlowObj.verticalArrayLength...
                * faultFluidFlowObj.horizontalArrayLength), :);

            velocity = pressureVelocity((...
                faultFluidFlowObj.verticalArrayLength...
                * faultFluidFlowObj.horizontalArrayLength + 1):(...
                3 * faultFluidFlowObj.verticalArrayLength...
                * faultFluidFlowObj.horizontalArrayLength), :);

            displacement = pressureVelocity((...
                3 * faultFluidFlowObj.verticalArrayLength...
                * faultFluidFlowObj.horizontalArrayLength + 1):end, :);
        end

        function markFracture(faultFluidFlowObj)

            % Mark location of fracture on pressure plot.

            line([faultFluidFlowObj.FaultArchitectureEnds.Fracture...
                faultFluidFlowObj.simulatedFaultWidth], [...
                0 (faultFluidFlowObj.simulatedFaultWidth...
                - faultFluidFlowObj.FaultArchitectureEnds.Fracture)...
                * tand(faultFluidFlowObj.faultAngle)], 'Color', 'white',...
                'LineStyle', '--');
        end

        function markFault(faultFluidFlowObj)

            % Mark location of fault zone architecture on plot.

            line([faultFluidFlowObj.FaultArchitectureEnds.Protolith1...
                faultFluidFlowObj.simulatedFaultWidth], [...
                0 (faultFluidFlowObj.simulatedFaultWidth...
                - faultFluidFlowObj.FaultArchitectureEnds.Protolith1)...
                * tand(faultFluidFlowObj.faultAngle)], 'Color', 'white',...
                'LineStyle', '--');

            line([faultFluidFlowObj.FaultArchitectureEnds.PSZ...
                faultFluidFlowObj.simulatedFaultWidth], [...
                0 (faultFluidFlowObj.simulatedFaultWidth...
                - faultFluidFlowObj.FaultArchitectureEnds.PSZ)...
                * tand(faultFluidFlowObj.faultAngle)], 'Color', 'white',...
                'LineStyle', '--');

            line([faultFluidFlowObj.FaultArchitectureEnds.DamageZone2...
                faultFluidFlowObj.simulatedFaultWidth], [...
                0 (faultFluidFlowObj.simulatedFaultWidth...
                - faultFluidFlowObj.FaultArchitectureEnds.DamageZone2)...
                * tand(faultFluidFlowObj.faultAngle)], 'Color', 'white',...
                'LineStyle', '--');
        end

        function image1 = resizeImageToMatch(image1, image2)

            % Resize an image to match another.
```

```
        tempSize = size(image2);
        image1 = imresize(image1, tempSize(1:2));
    end

    function condition = initialTimeCondition(loopCounter)

        % Initial timestep condition.

        condition = (loopCounter == 1);
    end

    function axin = convertToMpa(axin)

        % Convert axis units from Pa to MPa.

        h = get(axin, 'xtick');
        set(axin, 'xticklabel', h / 10 ^ 6);
        h = get(axin, 'ytick');
        set(axin, 'yticklabel', h / 10 ^ 6);
    end

    function output = roundToMPa(input)

        % Round a value from Pa to MPa.

        output = round((input + 1E7) / 1E6, -1) * 1E6;
    end

    function makeDirectory(folderName)

        % Create directory.

        if exist(folderName, 'dir') ~= 7
            mkdir(folderName);
        end
    end

    function value =  minStructValue(inputStruct)

        %Find the minimum value in a struct.

        value = cell2mat(...
            struct2cell(inputStruct));
    end

    function image = copyPlotToImage(fig)

        % Copy a plot to an image.

        saveas(fig, 'temp.tiff')
        image = imread('temp.tiff');
    end

    function copyPlotToSubplot(inputFigure, inputSubplot)

        % Copy a plot to a subplot.

        axisTemp = get(inputFigure, 'CurrentAxes');
        copyobj(allchild(axisTemp), inputSubplot);
        copyobj(get(axisTemp, 'XLabel'), inputSubplot);
        copyobj(get(axisTemp, 'YLabel'), inputSubplot);
    end

    function [ax, h] = subtitle(text, position)
```

```
            % Add a subtitle to a plot.

            ax = axes('Units', 'Normal', 'Position', position,...
                'Visible', 'off');
            set(get(ax, 'Title'), 'Visible', 'on')
            title(text);
            if (nargout < 2)
                return
            end
            h = get(ax, 'Title');
        end

        function output = checkAllStructFields(inputStruct)

            %Check if any of the field values in the first level of a
            % struct are true, given that all the fields are boolean.

            fields = fieldnames(inputStruct);
            output = false;

            for i = 1:length(fields)
                if inputStruct.(fields{i})
                    output = true;
                end
            end

        end
    end
end
```

```
ans =

  ResultPlottingClass with properties:

           horizontalArrayLength: []
             verticalArrayLength: []
          EarthquakeLengthVector: []
                StabilityLengths: []
                   FailureMarker: []
            slidingFailureMarker: []
                   oldFailureMap: []
                        pressure: []
                          stress: []
                 effectiveStress: []
                  outputPressure: []
                      MohrCircle: [1×1 struct]
                   MohrGeometry: [1×1 struct]
                           time: []
    effectiveNormalStressForPlot: []
       EffectiveNormalStressStep: 10000
     processingEarthquakeTrigger: 0
       brittleFailureTimeTrigger: 0
       ductileFailureTimeTrigger: 0
              steadyStateTrigger: 0
              brittleFailureTime: NaN
              ductileFailureTime: NaN
                SubplotFileNames: [1×1 struct]
                 mohrFigureScale: []
                   plotTimeScale: []
                   decimalPlaces: 1
                     limitYValue: 40
                 pressureSubplot: []
                     mohrSubplot: []
                   stressSubplot: []
```

```
    pressureSubplotElement: []
        mohrSubplotElement: []
      stressSubplotElement: []
              legendVector: []
                folderName: []
           folderCheckFlag: 1
               plotCounter: 0
effectiveStressPatchVector: []
      effectiveStressPatch: []
             computerStore: []
              lowMohrLimit: []
              lastPressure: []
         steadyStateLimit: 1.0000e-12
            gaussianWidth: 1.2500
  lastSlidingFailureMarker: []
         lastFailureMarker: []
```

---

*Published with MATLAB® R2018b*

```
classdef SideBySidePlottingClass < ResultPlottingClass
    properties
        SideBySideInitialStruct = struct('Failure', [], 'NoFailure', []);
        SideBySideStruct =...
            repmat(struct('Pressure', [], 'FailureMarker', [],...
            'FailureEnvelope', ...
            struct('OFC', [], 'PSZ', []), 'MohrCircle', ...
            struct('OFC', [], 'PSZ', []), 'Time', [],...
            'PoreFluidFactor', [],...
            'Image', struct('Pressure', [], 'MohrFailure', ...
            struct('OFC', [], 'PSZ'...
            , []), 'EffectiveNormalStressForPlot', []),...
            'FailureLength', [], 'NucleationLength', []), 6, 1);
        SideBySideResultStruct;
        FailureEnvelope = struct;
        poreFluidFactorList

        imageCoordinates
        outputFile;
        imageStore;
    end

    methods
        function resultProcessing(...
                obj,...
                faultFluidFlowObj)

            % Process side by side plotting result.

            FaultFluidFlowClass.printProgressString(...
                'Printing side-by-side results...');

            obj.initialiseFailurePlaneStruct(...
                faultFluidFlowObj);

            stableSlidingTrigger = false;
            faultFluidFlowObj.slidingStress = NaN(...
                obj.verticalArrayLength,...
                obj.horizontalArrayLength);

            for loopCounter = 1:length(faultFluidFlowObj.time)

                if ~obj.processingEarthquakeTrigger
                    loopTime = faultFluidFlowObj.time(loopCounter);
                    obj.postProcessingPhysicalVariables(...
                        faultFluidFlowObj,...
                        loopCounter,...
                        loopTime);

                    if obj.initialTimeCondition(...
                            loopCounter)
                        obj.sideBySideLoopOutput(...
                            faultFluidFlowObj,...
                            loopTime,...
                            loopCounter);
                    end

                    if obj.brittleFailureCondition
                        obj.sideBySideLoopOutput(...
                            faultFluidFlowObj,...
                            loopTime,...
                            loopCounter);
                        obj.brittleFailureTimeTrigger = true;
```

```
                end

                if obj.ductileFailureCondition
                    obj.sideBySideLoopOutput(...
                        faultFluidFlowObj,...
                        loopTime,...
                        loopCounter);
                    obj.ductileFailureTimeTrigger = true;
                end

                if obj.onsetOfStableSlidingCondition(...
                        faultFluidFlowObj,...
                        loopCounter) && ~stableSlidingTrigger

                    if ~obj.brittleFailureTimeTrigger...
                            && ~obj.ductileFailureTimeTrigger
                        obj.plotCounter = obj.plotCounter + 1;
                    end

                    obj.sideBySideLoopOutput(...
                        faultFluidFlowObj,...
                        loopTime,...
                        loopCounter);

                    stableSlidingTrigger = true;
                end

                if obj.unstableSlidingCondition(...
                        faultFluidFlowObj,...
                        loopCounter)
                    obj.sideBySideLoopOutput(...
                        faultFluidFlowObj,...
                        loopTime,...
                        loopCounter);
                    obj.processingEarthquakeTrigger = true;
                end
            end
        end
    end

    function sideBySideLoopOutput(...
            obj,...
            faultFluidFlowObj,...
            loopTime,...
            loopTimeCounter)

        % Output side by side plotting results variables for a given
        %timestep.

        obj.plotCounter = obj.plotCounter + 1;

        obj.mohrPlotVariables(faultFluidFlowObj, loopTime);

        obj.SideBySideStruct(obj.plotCounter).PoreFluidFactor...
            = faultFluidFlowObj.poreFluidFactor;
        obj.SideBySideStruct(obj.plotCounter).Pressure...
            = obj.pressure;
        obj.SideBySideStruct(obj.plotCounter).FailureMarker =...
            obj.FailureMarker;
        obj.SideBySideStruct(obj.plotCounter).Time...
            = loopTime;

        obj.SideBySideStruct(...
            obj.plotCounter).FailureEnvelope.OFC...
            = faultFluidFlowObj.FailureEnvelope.OFC;
```

```
            obj.SideBySideStruct(...
                obj.plotCounter).FailureEnvelope.PSZ...
                = faultFluidFlowObj.FailureEnvelope.PSZ;
            obj.SideBySideStruct(...
                obj.plotCounter).MohrCircle.OFC...
                = obj.MohrCircle.OFC;
            obj.SideBySideStruct(...
                obj.plotCounter).MohrCircle.PSZ...
                = obj.MohrCircle.PSZ;
            obj.SideBySideStruct(...
                obj.plotCounter...
                ).EffectiveNormalStressForPlot...
                = obj.effectiveNormalStressForPlot;

            obj.SideBySideStruct(...
                obj.plotCounter...
                ).FailureLength...
                = faultFluidFlowObj.EarthquakeLengthVector(...
                loopTimeCounter).Failure;
            obj.SideBySideStruct(...
                obj.plotCounter).NucleationLength...
                = faultFluidFlowObj.EarthquakeLengthVector(...
                loopTimeCounter).Nucleation;

        end

        function mohrPlot(...
                obj,...
                faultFluidFlowObj,...
                SideBySideStruct,...
                loopCounter1,...
                loopCounter2,...
                loopCounter3,...
                plotType,...
                component)

            % Plot mohr circle.

            [MohrCircle, temporaryX, temporaryY]...
                = obj.mohrPlotParameters(...
                SideBySideStruct,...
                loopCounter3,...
                component);

            Axes = gca;

            colorString = obj.setMohrPlotColor(...
                length(Axes.Children));

            axis('equal')
            plot(temporaryX, temporaryY, plotType)
            hold on;
            plot(temporaryX, MohrCircle)
            hold on;

            if faultFluidFlowObj.SlidingFailureFlag.(component)
                SideBySidePlottingClass.plotFailurePlane(...
                    faultFluidFlowObj,...
                    temporaryX,...
                    MohrCircle,...
                    colorString);
            end

            obj.mohrAxis(...
                SideBySideStruct,...
```

```
                MohrCircle,...
                temporaryX,...
                loopCounter3);

            obj.saveMohrFigure(...
                [num2str(loopCounter1)...
                '_'...
                num2str(loopCounter2)...
                '_'...
                num2str(loopCounter3)]);
        end

        function sideBySidePrintLine(...
                obj,...
                fields,...
                loopCounter1,...
                loopCounter2,...
                loopCounter3)
            fprintf(...
                obj.outputFile,...
                '%s\n',...
                ['Figure ' num2str(loopCounter2)...
                ', poreFluidFactor ' num2str(...
                obj.SideBySideResultStruct(...
                loopCounter2).(...
                fields{loopCounter1})(loopCounter3).PoreFluidFactor) ...
                ', Failure Length ' num2str(...
                obj.SideBySideResultStruct(...
                loopCounter2).(...
                fields{loopCounter1})(loopCounter3).FailureLength)...
                ', Nucleation Length '...
                num2str(...
                obj.SideBySideResultStruct(...
                loopCounter2).(...
                fields{loopCounter1})(loopCounter3).NucleationLength)]);
        end

        function sideBySidePlot(...
                obj,...
                faultFluidFlowObj)

            % Plot side by side results.

            obj.outputFile = fopen([obj.folderName '/Output.txt'], 'wt');
            fields = fieldnames(obj.SideBySideResultStruct);
            obj.populateporeFluidFactorList(fields);

            for loopCounter1 = 1:length(obj.SideBySideResultStruct)
                for loopCounter2 = 1:length(fields)
                    for loopCounter3 = 1:length(...
                            obj.SideBySideResultStruct(loopCounter1).(...
                            fields{loopCounter2}))

                        eventTime = obj.getSideBySideResultStruct(...
                            fields,...
                            loopCounter1,...
                            loopCounter2,...
                            loopCounter3);

                        obj.plotCounter = obj.plotCounter + 1;

                        if ~isempty(...
                                obj.SideBySideResultStruct(...
                                loopCounter1).(...
                            fields{loopCounter2})(loopCounter3).Pressure)
```

```
                        pressureFigure...
                            = obj.pressureFigure(...
                            faultFluidFlowObj,...
                            eventTime,...
                            [num2str(loopCounter1)...
                            '_'...
                            num2str(loopCounter2)...
                            '_'...
                            num2str(loopCounter3)]);

                        close(pressureFigure);
                    end
                end
            end

            obj.plotCounter = 0;
            for loopCounter2 = 1:length(fields)
                for loopCounter3 = 1:length(...
                        obj.SideBySideResultStruct(loopCounter1).(...
                        fields{loopCounter2}))
                    if ~isempty(...
                            obj.SideBySideResultStruct(...
                            loopCounter1).(...
                            fields{loopCounter2})(loopCounter3).Pressure)

                        obj.postProcessMohrDiagram(...
                            faultFluidFlowObj,...
                            fields,...
                            loopCounter1,...
                            loopCounter2,...
                            loopCounter3);
                    end
                end
            end

            for loopCounter2 = 1:length(fields)
                for loopCounter3 = 1:length(...
                        obj.SideBySideResultStruct(loopCounter1).(...
                        fields{loopCounter2}))
                    pressureImage =...
                        obj.SideBySideResultStruct(loopCounter1).(...
                        fields{...
                        loopCounter2})(loopCounter3).Image.Pressure;
                    if ~isempty(pressureImage)
                        imwrite(pressureImage,...
                            [obj.folderName '/DiagramPressure'...
                            fields{loopCounter2}...
                            num2str(loopCounter1)...
                            '_' num2str(loopCounter3) '.tiff']);
                    end
                end
            end
        end

    end

    function populateporeFluidFactorList(obj, fields)
        obj.poreFluidFactorList = zeros(2, 1);
        for loopCounter = 1:length(obj.SideBySideResultStruct)
            obj.poreFluidFactorList(loopCounter) =...
                obj.SideBySideResultStruct(...
                loopCounter).(fields{1})(1).PoreFluidFactor;
        end
    end
```

```
        function eventTime = getSideBySideResultStruct(...
                obj,...
                fields,...
                loopCounter1,...
                loopCounter2,...
                loopCounter3)

            obj.pressure = obj.SideBySideResultStruct(...
                loopCounter1).(...
                fields{loopCounter2})(loopCounter3).Pressure;

            eventTime = obj.SideBySideResultStruct(loopCounter1).(...
                fields{loopCounter2})(loopCounter3).Time;

            if ~isempty(eventTime)
                obj.FailureMarker.Central...
                    = obj.SideBySideResultStruct(loopCounter1).(...
                    fields{...
                    loopCounter2})(loopCounter3).FailureMarker.Central;
            end
        end

        function trimEmptyParameters(obj, loopCounter1)
            obj.SideBySideResultStruct(loopCounter1) = [];
        end

        function LocalStruct = postProcessMohrDiagram(...
                obj,...
                faultFluidFlowObj,...
                fields,...
                loopCounter1,...
                loopCounter2,...
                loopCounter3)

            ofcFigure = figure;
            pszFigure = figure;

            plotType =...
                SideBySidePlottingClass.omittedFailureEnvelope(...
                loopCounter2);

            for internalCounter = 1:length(obj.SideBySideResultStruct)
                LocalStruct = obj.SideBySideResultStruct(...
                    internalCounter).(fields{loopCounter2});

                obj.plotCounter = obj.plotCounter + 1;
                set(0, 'CurrentFigure', ofcFigure)
                obj.mohrPlot(...
                    faultFluidFlowObj,...
                    LocalStruct,...
                    loopCounter1,...
                    loopCounter2,...
                    loopCounter3,...
                    plotType,...
                    'OFC');

                hold on
                obj.plotCounter = obj.plotCounter + 1;
                set(0, 'CurrentFigure', pszFigure)
                box on
                obj.mohrPlot(...
                    faultFluidFlowObj,...
                    LocalStruct,...
                    loopCounter1,...
                    loopCounter2,...
```

```
                loopCounter3,...
                'k',...
                'PSZ')

            if loopCounter3 <= length(obj.SideBySideResultStruct(...
                    internalCounter).(...
                    fields{loopCounter2}))

                if obj.SideBySideResultStruct(...
                        internalCounter).(...
                        fields{loopCounter2})(loopCounter3).Time == 0
                    SideBySidePlottingClass.addMohrPlotLegends(...
                        ofcFigure,....
                        pszFigure);
                end
            end
        end

        ofcMohrImage = ResultPlottingClass.copyPlotToImage(...
            ofcFigure);
        pszMohrImage = ResultPlottingClass.copyPlotToImage(...
            pszFigure);


        if strcmp(obj.computerStore, 'GLNXA64')
            cropVec = [0, 100, 900, 700];

        else
            cropVec = [0, 100, 1200, 900];

        end

        ofcMohrImage = imcrop(...
            ofcMohrImage, cropVec);
        pszMohrImage = imcrop(...
            pszMohrImage, cropVec);

        imwrite(ofcMohrImage,...
            [obj.folderName '/DiagramOFCMohr'...
            fields{loopCounter2}...
            num2str(loopCounter2)...
            '_' num2str(loopCounter3) '.tiff']);
        imwrite(pszMohrImage,...
            [obj.folderName '/DiagramPSZMohr'...
            fields{loopCounter2}...
            num2str(loopCounter2)...
            '_' num2str(loopCounter3) '.tiff']);

        close(ofcFigure);
        close(pszFigure);
    end

    function mohrAxis(...
            obj,...
            SideBySideResultStruct,...
            MohrCircle,...
            temporaryX,...
            loopCounter3)

        xLimit = 130;
        yLimit = 50; %For consistent paper diagram.
        x1 = min(temporaryX(isfinite(MohrCircle)));
        x2 = max(temporaryX(isfinite(MohrCircle)));
        xlim([0, xLimit]);
        ylim([0, yLimit]);
```

```
            xticks([]);
            yticks([]);
        end

        function initialiseImageCoordinates(obj)
            if strcmp(obj.computerStore, 'MACI64')
                obj.imageCoordinates = [100, 288];
            elseif strcmp(obj.computerStore, 'GLNXA64')
                obj.imageCoordinates = [103, 288];
            end
        end

        function image = labelOverpressureEdge(obj, image)
            image = insertShape(image, 'Line',...
                [obj.imageCoordinates(1) + 75,...
                obj.imageCoordinates(2) - 82, ...
                (obj.imageCoordinates(1) + 105),...
                obj.imageCoordinates(2) - 82], 'Color', 'black');
        end

        function image = labelOFCIFCBoundary(obj, image)
            image = insertShape(image, 'Line',...
                [obj.imageCoordinates(1) + 275 ...
                obj.imageCoordinates(2) - 255 ...
                obj.imageCoordinates(1) + 275 ...
                obj.imageCoordinates(2) + 85],...
                'Color', 'black', 'LineWidth', 5);
        end

        function image = labelArchitectureComponent(obj, ...
                image, xOffset, yOffset, labelString)
            image = insertText(image,...
                [obj.imageCoordinates(1) + xOffset...
                obj.imageCoordinates(2) + yOffset],...
                labelString, 'BoxOpacity', 0, 'FontSize', 14);
        end

        function image = labelFaultCore(obj, image)
            image = insertText(image,...
                [obj.imageCoordinates(1) + 100 ...
                obj.imageCoordinates(2) - 280],...
                [sprintf('%s', char(8592))...
                '                FC                '...
                sprintf('%s', char(8594))], 'BoxOpacity', 0,...
                'FontSize', 20);
        end

        function image = labelPoreFluidFactor(...
                obj, image, xOffset, yOffset, valueString, color)
            image = insertText(...
                image, [(obj.imageCoordinates(1) + xOffset) ...
                (obj.imageCoordinates(2) + yOffset)],...
                [sprintf('%sv', char(955))...
                ' = ' valueString],...
                'TextColor', color, 'BoxOpacity', 0,...
                'FontSize', 14);
        end

        function image = labelVerticalStress(...
                obj, image, xOffset, yOffset, faultFluidFlowObj)
            image = insertText(image,...
                [obj.imageCoordinates(1) + xOffset...
                obj.imageCoordinates(2) + yOffset], ...
                [sprintf('%s', char(963))...
                'v = ' num2str(round(...
```

```
            faultFluidFlowObj.lithostaticStress / 1E6, 3,...
            'significant')) ' MPa'], 'TextColor', 'White',...
            'BoxOpacity', 0,...
            'FontSize', 14);
    end

    function image = insertTime(...
            obj,...
            image,...
            loopCounter3,...
            SideBySideResultStruct)
        image = insertText(image, [obj.imageCoordinates(1) + 100 ...
            obj.imageCoordinates(2) - 290], ...
            [num2str(SideBySideResultStruct(loopCounter3).Time...
            / FaultFluidFlowClass.SECONDS_PER_YEAR, 4) ' years'],...
            'Font', 'LucidaSansDemiBold', 'BoxOpacity', 0,...
            'FontSize', 14);
    end

    function image = generalImageProcessing(...
            obj,...
            image,...
            loopCounter3,...
            SideBySideResultStruct)

        if strcmp(obj.computerStore, 'GLNXA64')
            baseVector = [150, 85];
        else
            baseVector = [150, 85];
        end

        image = obj.labelArchitectureComponent(...
            image, 0, 12, 'DZ');
        image = obj.labelArchitectureComponent(...
            image, baseVector(1), baseVector(2), 'OFC');
        image = obj.labelArchitectureComponent(...
            image, baseVector(1)+80, baseVector(2), 'IFC');
        image = obj.labelOverpressureEdge(image);
        image = obj.labelOFCIFCBoundary(image);
        image = obj.insertTime(...
            image, loopCounter3, SideBySideResultStruct);
        image = obj.labelPoreFluidFactor(...
            image, 0, 32, num2str(...
            SideBySideResultStruct(loopCounter3).PoreFluidFactor),...
            'Black');
    end

    function lengthPosition = lengthLabelsPosition(obj)
        lengthPosition = [obj.imageCoordinates(1) + 285 ...
            obj.imageCoordinates(2)...
            + 85 obj.imageCoordinates(1)...
            + 285 obj.imageCoordinates(2) + 85];
    end

    function nucleationLengthPosition...
            = nucleationLengthPosition(...
            obj,...
            SideBySideResultStruct,...
            loopCounter3)

        %

        lengthPosition = obj.lengthLabelsPosition;

        nucleationLengthPosition = lengthPosition;
```

```
            nucleationLengthPosition(4) = nucleationLengthPosition(4)...
                - 4.25...
                * SideBySideResultStruct(loopCounter3).NucleationLength;
            nucleationLengthPosition(2) = nucleationLengthPosition(4);
            nucleationLengthPosition(3) = nucleationLengthPosition(3)...
                + 20;
        end

        function failureLengthPosition = failureLengthPosition(...
                obj,...
                SideBySideResultStruct,...
                loopCounter3)

            lengthPosition = obj.lengthLabelsPosition;
            failureLengthPosition = lengthPosition;
            failureLengthPosition(4) = failureLengthPosition(4)...
                - 4.25...
                * SideBySideResultStruct(loopCounter3).FailureLength;
        end


        function colorString = setMohrPlotColor(...
                obj,...
                counter)

            if counter < 3
                colorString = 'b';
            elseif counter == 6
                colorString = 'r';
            else
                colorString = 'r';
            end
        end

        function image = labelFailureLength(...
                obj,...
                SideBySideResultStruct,...
                image,...
                loopCounter3)

            failureLengthPosition...
                =...
                obj.failureLengthPosition(...
                SideBySideResultStruct,...
                loopCounter3);

            image = insertShape(image, 'Line', failureLengthPosition,...
                'Color', 'black', 'LineWidth', 1);

            image = insertText(...
                image, failureLengthPosition(1:2), sprintf('%s', 'LF'),...
                'BoxOpacity', 0, 'TextColor', 'Black',...
                'FontSize', 14);
        end

        function image = labelNucleationLength(...
                obj,...
                image,...
                SideBySideResultStruct,...
                loopCounter3)

            nucleationLengthPosition...
                = obj.nucleationLengthPosition(...
                SideBySideResultStruct,...
                loopCounter3);
```

```
            image = insertShape(image, 'Line', nucleationLengthPosition,...
                'Color', 'red', 'LineWidth', 1);

            image = insertText(...
                image,...
                nucleationLengthPosition(3:4) + [-25, 0],...
                sprintf('%s', 'LN')...
                , 'BoxOpacity', 0, 'TextColor', 'Red',...
                'FontSize', 14);
        end
    end

    methods(Static)
        function outputImage = concatenateImageRow(...
                outputImage, pressureImage1, pressureImage2, mohrImage1,...
                mohrImage2)

            outputImage = cat(1, outputImage, cat(2, pressureImage1,...
                pressureImage2, imresize(cat(1, mohrImage2,...
                mohrImage1), size(pressureImage1, 1) /...
                (2 * size(mohrImage2, 1)))));

        end

        function addMohrPlotLegends(ofcFigure, pszFigure)
            set(0, 'CurrentFigure', ofcFigure)
            text(1, 20, '\tau = \mu_S \sigma''_N + C',...
                'Rotation', 37,  'FontSize', 14)

            set(0, 'CurrentFigure', pszFigure)
            text(30, 21, '\tau = \mu_S \sigma''_N',...
                'Rotation', 35, 'FontSize', 14)
        end

        function [MohrCircle, temporaryX, temporaryY]...
                = mohrPlotParameters(...
                SideBySideResultStruct,...
                loopCounter3,...
                component)
            unitFactor = 1E6;

            MohrCircle = [];
            temporaryX = [];
            temporaryY = [];

            if length(SideBySideResultStruct) >= loopCounter3

                MohrCircle = SideBySideResultStruct(...
                    loopCounter3).MohrCircle.(component) / unitFactor;

                temporaryX = SideBySideResultStruct(...
                    loopCounter3).EffectiveNormalStressForPlot...
                    / unitFactor;
                temporaryY = SideBySideResultStruct(...
                    loopCounter3).FailureEnvelope.(component)...
                    / unitFactor;
            end
        end

        function mohrArchitectureLabel(architectureString)
            text(1, 45, architectureString);
        end

        function plotType = omittedFailureEnvelope(loopCounter2)
```

```
            if loopCounter2 == 1
                plotType = 'k';
            else
                plotType = 'k--';
            end
        end

        function graphFinalProcessing
            title([]);
            pbaspect([1 2 1]);
        end

        function condition = initialImageCondition(...
                SideBySideResultStruct, loopCounter1, loopCounter3)
            condition = SideBySideResultStruct(loopCounter3).Time == 0 ...
                && loopCounter1 == 1;
        end

        function condition = secondImageCondition(...
                SideBySideResultStruct, loopCounter1, loopCounter3)
            condition = SideBySideResultStruct(loopCounter3).Time == 0 ...
                && loopCounter1 == 2;
        end

        function hydrostaticPoreFluidFactor =...
                hydrostaticPoreFluidFactor(faultFluidFlowObj)
            hydrostaticPoreFluidFactor = round(...
                faultFluidFlowObj.hydrostaticStress...
                / faultFluidFlowObj.lithostaticStress, 1);
        end

        function image1 = resizeImageToMatch(image1, image2)
            tempSize = size(image2);

            if ~isempty(image1)
                image1 = imresize(image1, tempSize(1:2));
            end
        end

        function plotFailurePlane(...
                faultFluidFlowObj,...
                temporaryX,...
                MohrCircle,...
                plotType)

            faultAngle = faultFluidFlowObj.faultAngle;

            mohrRadius = max(MohrCircle);

            mohrCentre = temporaryX(MohrCircle == mohrRadius);

            if ~isempty(mohrRadius) && ~isempty(mohrCentre)
                line([mohrCentre,...
                    (mohrCentre + mohrRadius * cosd(2 * faultAngle))],...
                    [0, ...
                    mohrRadius * sind(2 * faultAngle)], 'Color', plotType);
            end
        end
    end
end
```

```
ans =

  SideBySidePlottingClass with properties:
```

```
           SideBySideInitialStruct: [1×1 struct]
                  SideBySideStruct: [6×1 struct]
            SideBySideResultStruct: []
                   FailureEnvelope: [1×1 struct]
              poreFluidFactorList: []
                 imageCoordinates: []
                       outputFile: []
                       imageStore: []
            horizontalArrayLength: []
              verticalArrayLength: []
           EarthquakeLengthVector: []
                 StabilityLengths: []
                    FailureMarker: []
             slidingFailureMarker: []
                    oldFailureMap: []
                        pressure: []
                          stress: []
                 effectiveStress: []
                  outputPressure: []
                      MohrCircle: [1×1 struct]
                   MohrGeometry: [1×1 struct]
                           time: []
    effectiveNormalStressForPlot: []
       EffectiveNormalStressStep: 10000
     processingEarthquakeTrigger: 0
       brittleFailureTimeTrigger: 0
       ductileFailureTimeTrigger: 0
              steadyStateTrigger: 0
              brittleFailureTime: NaN
              ductileFailureTime: NaN
                SubplotFileNames: [1×1 struct]
                  mohrFigureScale: []
                    plotTimeScale: []
                    decimalPlaces: 1
                      limitYValue: 40
                 pressureSubplot: []
                     mohrSubplot: []
                   stressSubplot: []
          pressureSubplotElement: []
              mohrSubplotElement: []
            stressSubplotElement: []
                    legendVector: []
                      folderName: []
                 folderCheckFlag: 1
                     plotCounter: 0
      effectiveStressPatchVector: []
            effectiveStressPatch: []
                   computerStore: []
                    lowMohrLimit: []
                    lastPressure: []
                steadyStateLimit: 1.0000e-12
                   gaussianWidth: 1.2500
          lastSlidingFailureMarker: []
                 lastFailureMarker: []
```

---

*Published with MATLAB® R2018b*

```
classdef SinglePhaseFluidFlowClass < FaultFluidFlowClass
    methods

        function obj = faultFluidFlowSolver(obj)

            % Solve fault fluid flow problem.

            FaultFluidFlowClass.printProgressString(...
                'Solving single phase fluid flow problem...')

            endTrigger = false;
            timeVector = obj.time;
            timeOutput = [];
            outputPressure = [];

            options = obj.initialiseSolverOptions;
            storeStruct = struct;

            storeStruct.FailureMarkerStore = obj.initialiseXZCMidpointStruct;
            storeStruct.slidingFailureMarkerStore = obj.slidingFailureMarker;
            storeStruct.slidingStressStore = obj.slidingStress;

            storeStruct.EarthquakeLengthStore = struct(...
                'Failure', NaN,...
                'Nucleation', NaN);

            initialSolverVariable = obj.initialSolverVariable;
            obj.initialSolverVariable = [];

            while length(timeVector) ~= 1

                lastwarn('');

                [timeVector,...
                    solverVariable,...
                    eventTime,...
                    eventSolverVariable]...
                    = ode23tb(...
                    @obj.differentialPressureEquation,...
                    timeVector,...
                    initialSolverVariable,...
                    options);

                [~, msgid] = lastwarn;

                if  strcmp(msgid, 'MATLAB:ode15s:IntegrationTolNotMet')
                    throw(...
                    MException(...
                    'CUSTOM:TolErr',...
                    'Unable to meet integration tolerance.'));
                end

                timeLength = length(timeVector);
                timeOutput = [timeOutput timeVector'];

                outputPressure...
                    = [...
                    outputPressure...
                    solverVariable'];

                if length(eventTime) ~= 1 && isempty(eventTime) == 0
                    eventTime = eventTime(end);
                    eventSolverVariable = eventSolverVariable(end, :);
```

```
            end

            if isempty(eventTime) == 1
                eventSolverVariable = solverVariable(end, :);
                eventTime = obj.time(end);
            end

            eventPressure = eventSolverVariable;

            [failureLength, nucleationLength, storeStruct] =...
                obj.discontinuousModeOfFailure(...
                eventPressure,...
                timeLength,...
                eventTime,...
                storeStruct);

            if nucleationLength <= failureLength...
                    && ~isnan(failureLength)
                eventTime = [];
            end

            if endTrigger
                break
            end

            if isempty(eventTime) == 1 ...
                    || nucleationLength <= failureLength...
                    && ~isnan(failureLength)
                obj.time = timeOutput;
                endTrigger = true;
            end

            if isempty(eventTime) == 1
                obj.time = timeOutput;
                break;
            end

            options = odeset(...
                options,...
                'InitialStep',...
                (timeVector(end) - timeVector(end-1)) / 1000);

            timeVector...
                = eventTime:(obj.SECONDS_PER_YEAR /...
                obj.timeVectorDensity):obj.maximumSimulationTime;

            initialSolverVariable = eventSolverVariable;
        end

        obj.outputPressure = outputPressure;
        obj.FailureMarkerStore = storeStruct.FailureMarkerStore;
        obj.slidingFailureMarkerStore = storeStruct.slidingFailureMarkerStore;
        obj.slidingStressStore = storeStruct.slidingStressStore;

        obj.EarthquakeLengthStore = storeStruct.EarthquakeLengthStore;

        obj.options = options;
    end

    function initialiseIntensiveVariables(obj, confinementFactor)

        % Initialise intensive physical variables.

        obj.initialiseNonSolverVariables(confinementFactor);
```

```
            obj.initialSolverVariable = reshape(...
                obj.initialPressure, [], 1);
            obj.initialPressure = [];
        end

        function [failureLength, nucleationLength, storeStruct] =...
                discontinuousModeOfFailure(...
                obj,...
                pressure,...
                timeLength,...
                eventTime,...
                storeStruct)

            % Make mode of failure update at non-smooth ODE
            % interruptions.

            pressure = reshape(...
                pressure,...
                obj.verticalArrayLength, ...
                obj.horizontalArrayLength);

            pressure = obj.pressureBCS(pressure);

            %if ~obj.modeOfFailureFlag
            %    obj.FailureMarker = obj.removeModeOfFailure;
            %end

            oldSlidingFailureMarker = obj.slidingFailureMarker;
            OldFailureMarker = obj.FailureMarker;
            oldSlidingStress = obj.slidingStress;

            [obj.Permeability,...
                obj.FailureMarker,...
                obj.slidingFailureMarker,...
                ~,...
                failureLength,...
                nucleationLength,...
                obj.slidingStress]...
                = obj.rockMatrixState(pressure, eventTime);

            if length(obj.EarthquakeLengthStore) == 1
                timeLength = timeLength - 1;
            end

            storeStruct.FailureMarkerStore = cat(...
                3,...
                storeStruct.FailureMarkerStore,...
                repmat(OldFailureMarker, 1, 1, timeLength));

            storeStruct.slidingFailureMarkerStore = cat(...
                3,...
                storeStruct.slidingFailureMarkerStore,...
                repmat(oldSlidingFailureMarker, 1, 1, timeLength));

            storeStruct.slidingStressStore = cat(...
                3,...
                storeStruct.slidingStressStore,...
                repmat(oldSlidingStress, 1, 1, timeLength));

            storeStruct.EarthquakeLengthStore = cat(...
                3,...
                storeStruct.EarthquakeLengthStore,...
                repmat(...
                storeStruct.EarthquakeLengthStore(1),...
                1, 1, timeLength));
```

```
            if (nucleationLength - failureLength)...
                    / nucleationLength...
                    < 0
                failureLength = nucleationLength;
            end

            storeStruct.EarthquakeLengthStore(end) = struct(...
                'Failure', failureLength,...
                'Nucleation', nucleationLength);

            storeStruct.FailureMarkerStore(:, :, end) = obj.FailureMarker;
            storeStruct.slidingFailureMarkerStore(:, :, end)...
                = obj.slidingFailureMarker;
            storeStruct.slidingStressStore(:, :, end) = obj.slidingStress;

        end

        function pressureTimeDerivative = differentialPressureEquation(...
                obj,...
                time,...
                pressure)

            % Derivative of pressure with respect to time for each spatial
            % array point.

            pressure = reshape(pressure, obj.verticalArrayLength,...
                obj.horizontalArrayLength);

            pressure = obj.rockFluidCoupling(pressure, time);

            PressureFlux = obj.pressureFlux(pressure);
            PressureFluxDivergence = obj.spatialSecondDerivative(...
                PressureFlux);

            PressureFluxDivergence = obj.fluxDivergenceBCS(...
                PressureFlux,...
                PressureFluxDivergence);

            pressureTimeDerivative = obj.pressureTimeDerivative(...
                PressureFluxDivergence);

            pressureTimeDerivative = reshape(...
                pressureTimeDerivative, [], 1);
        end

        function initialiseSpatialArray(obj)

            % Initialise spatial arrays for simulation

            obj.initialiseCommonSpatialArray;
        end

        function initialiseRockMatrixVariables(obj)

            % Initialise rock matrix variables.

            obj.initialiseCommonRockMatrixVariables;
        end

        function options = initialiseSolverOptions(obj)

            % Initialise options for solvers used in simulation.

            jacobianPattern = spdiags(...
```

```
                ones(obj.horizontalArrayLength...
                * obj.verticalArrayLength, 5),...
                [-obj.verticalArrayLength;...
                -1;...
                0;...
                1;...
                obj.verticalArrayLength;],...
                obj.horizontalArrayLength...
                * obj.verticalArrayLength,...
                obj.horizontalArrayLength...
                * obj.verticalArrayLength);

            options = odeset(...
                'JPattern', jacobianPattern,...
                'Events',...
                @(time, pressure)obj.events(time, pressure),...
                'RelTol', 1E-8);

            obj.time = [];
        end
    end
end
```

```
ans =

  SinglePhaseFluidFlowClass with properties:

                SECONDS_PER_YEAR: 31556900
                 SECONDS_PER_DAY: 86400
           GRAVITATIONAL_CONSTANT: 9.8100
                   analyticalTime: []
            maximumSimulationTime: []
                 timeVectorLength: []
                             time: []
                       timeOutput: []
                timeVectorDensity: []
                                x: []
                                z: []
                            Delta: []
            faultArchitectureList: []
              simulatedFaultWidth: []
             simulatedFaultHeight: []
            horizontalArrayLength: []
              verticalArrayLength: []
            FaultArchitectureEnds: []
    ModeOfFailureArchitectureFlag: []
               SlidingFailureFlag: []
                  FineFeatureFlag: []
               overpressureHeight: []
                  overpressureMap: []
                          pszWidth: []
                    blankingArray: []
            EarthquakeLengthStore: []
           EarthquakeLengthVector: []
                     CohesiveFlag: []
        nucleationDetectionFactor: 500000
                     faultPreset: []
                     rockDensity: []
                      faultAngle: []
              FailureModeBoundary: []
              FrictionCoefficient: []
                        porosity: []
                 porosityStates: []
                 compressiblity: []
```

```
                    Viscosity: []
        UnstressedPermeability: []
           PressureSensitivity: []
                      Cohesion: []
           initialStressField: []
      arrayoverpressureHeight: []
          contactOverpressure: []
              initialPressure: []
        initialSolverVariable: []
                      pressure: []
                       Density: []
                  shearModulus: []
       rateAndStateDifference: []
         criticalSlipDistance: []
                           psi: []
                 slidingStress: []
                   FailureTime: [1×1 struct]
                 SlidingLength: [1×1 struct]
                 cohesionLimit: []
              failureStateList: []
                  FailureAngle: []
           TwoCosFailureAngle: []
             twoCosFaultAngle: []
    ArrayFaultArchitectureEnds: []
     ArrayFaultArchitectureMap: []
     FailureModeBoundaryStress: []
            hydrostaticStress: []
            lithostaticStress: []
                FailureMarker: []
         slidingFailureMarker: []
                failureExtent: []
                maximumStress: []
                minimumStress: []
                 Permeability: []
                      options: []
               outputPressure: []
         outputSolverVariable: []
           FailureMarkerStore: []
    slidingFailureMarkerStore: []
           slidingStressStore: []
             oldFailureMarker: []
             newFailureMarker: []
               PlotProperties: [1×1 struct]
                  twoCosAngle: []
                  twoSinAngle: []
        internalFrictionArray: []
                    cohesion: []
           TwoSinFailureAngle: []
             twoSinFaultAngle: []
              FailureEnvelope: []
              poreFluidFactor: []
          tectonicLoadingRate: []
                    faultDepth: []
           confinementFactor: []
                    OFCwidth: []
                    IFCwidth: []
                initialStress: []
            modeOfFailureFlag: 0
                plotTimeScale: []
                PlottingAngle: []
```

---

*Published with MATLAB® R2018b*