Numerical study of microwave induced stress and damage formation in heterogeneous rocks

Dipl.-Ing. Michael Toifl

Supervisors: Univ.-Prof. Dipl.-Ing. Dr. mont. Thomas Antretter
Ao.Univ.-Prof. Dr. phil. Ronald Meisels

Institute of Mechanics
Montanuniversitaet Leoben

Doctoral Thesis

May 2016

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Dipl.-Ing. Michael Toifl
May 2016

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Prof. Thomas Antretter for his continuous support, patience and motivation during my PhD study. With his immense knowledge but also encouragement he has guided me through the three years of the PhD project. Many thanks to Prof. Friedemar Kuchar and Prof. Ronald Meisels for proofreading this thesis and the technical support during the research. I would also like to acknowledge Dr. Philipp Hartlieb for his valuable inputs and the stimulating discussions. In addition, I would like to give thanks to the whole project team for coming up with the idea of this thesis.

Moreover, I would like to thank the Austrian Science Fund (FWF) for financing our project "Microwave-induced modifications of thermal and mechanical properties of rocks" (TRP 284-N30) and for supporting my scientific career by providing me the opportunity to publish papers and join conferences.

I also wish to express my gratitude to my colleagues from the Institute of Mechanics for the pleasant working atmosphere and their technical support not only during this research but also on teaching issues. I really enjoyed the last three years at the institute and I will miss the fun we had, especially with my office colleagues Dipl.-Ing. Andreas Kaufmann and Dr. Richard Jurisits. I would also like to thank Mag. Markus Orthaber and Dipl.-Ing. Karl Flicker for supporting me in IT and cluster problems.

I would like to thank my parents and grandparents for their financial support and encouragement throughout my education. Finally, my greatest thanks go to my girlfriend Victoria for her patience and unconditional support.

Abstract

Mechanical comminution of rocks is an energy intensive process with energy efficiency around 1%. A possible way to enhance the efficiency is the prior application of high-power microwaves. The aim of this thesis is to determine the microwave induced stresses and damage in heterogeneous (e.g. granite) as well as homogeneous hard rocks (e.g. basalt).

In the heterogeneous case a novel 3D simulation procedure to assess microwave induced stresses at a microstructure level is presented. For a realistic rock model two and three component 3D microstructures are generated by a Voronoi tessellation algorithm. In order to calculate the electromagnetic field inside the inhomogeneous rock, a 3D finite-difference time-domain (FDTD) simulation is performed. A microwave source with a typical technical frequency of 2.45 GHz is assumed. The absorbed heat is computed and applied as temperature distribution in a subsequent thermo-mechanical finite element (FE) analysis in order to calculate the thermally induced stresses and damage.

With a 3D two component model the influence of the microstructure on the microwave induced stress formation during microwave irradiation with a 25 kW source for 15 s and 25 s is assessed. In the 25 s case the effect of the α to β phase transformation of quartz at 573°C is investigated. The influence of the anisotropic nature of the quartz grains is assessed by comparing the stresses in the isotropic with the anisotropic case. High maximum principal stresses on the boundaries of the strong microwave absorbing phase exceeding the tensile strength are observed in the 15 s irradiation model. After 25 s of microwave irradiation even higher stresses as a consequence of phase transformation of quartz are determined. In the anisotropic case a significantly higher fraction exhibiting high maximum principal stresses especially in the microwave transparent phase are observed. By considering a non-linear damage material model, damage initiation around the main heated area and at the phase boundaries of the strong absorbing phase are determined. These observations correlate qualitatively with microwave irradiation experiments. It is concluded that the formation of stress and damage is highly influenced by the microstructure and the micromechanical behavior of the constituents (quartz phase transformation, anisotropic behavior).

In order to assess the industrial applicability, numerous 3D numerical analyses with varying irradiation times as well as microwave powers are performed on granite three component models. To this end measured dielectric and thermo-mechanical properties are used. Both constant microwave power and varying irradiation times as well as constant microwave energy and different irradiation time / power cases are investigated. Under constant power the largest maximum principal stresses rise linearly with the irradiation time whereas with constant energy an optimum irradiation time can be found giving maximum stresses. The presented 3D inhomogeneous simulations methodology allows to determine the optimum microwave irradiation parameters for the investigated granite.

Kurzfassung

Die mechanische Gesteinszerkleinerung ist ein energieintensiver Prozess und weist darüber hinaus lediglich einen Wirkungsgrad von ungefähr 1% auf. Ein vielversprechender Ansatz zur Steigerung der Effizienz des Prozesses ist die vorgelagerte Behandlung des Gesteins mit Mikrowellen. Das Ziel der vorliegenden Arbeit ist die Bestimmung der mikrowelleninduzierten Spannungen und Schädigungen sowohl in einem heterogenen (z.B.: Granit) als auch einem homogenen (z.B.: Basalt) Gestein.

Für den heterogenen Fall wurde eine neuartige 3D Simulationsmethodik entwickelt, um die mikrowelleninduzierten Spannungen auf Ebene der Mikrostruktur quantifizieren und analysieren zu können. Um ein realistisches Gesteinsmodell zu erhalten, wird mithilfe eines Voronoi Tessellations Algorithmus 3D Mikrostrukturen mit zwei und drei Komponenten erzeugt. Mit Hilfe eines 3D Finite Differenzen Verfahrens (FDTD, finite-difference time-domain) wird das elektromagnetische Feld in dem inhomogenen Gestein numerisch berechnet. Hierfür wird eine Mikrowellenquelle mit einer typischen technischen Frequenz von 2.45 GHz verwendet. Anschließend wird die absorbierte Wärme mit einem thermischen Finiten Elemente (FE) Modell analysiert. Das resultierende transiente Temperaturfeld wird in einer darauffolgenden thermomechanischen FE Analyse verwendet, um Spannungen und Schädigungen ableiten zu können.

Der Einfluss der Mikrostruktur auf die mikrowelleninduzierten Spannungen wird in einem 3D Modell mit zwei Gesteinskomponenten und mit einer Mikrowellenleistung von 25 kW sowie Bestrahlungszeiten von 15 s und 25 s bewertet. Die Quarzphasenumwandlung bei einer Temperatur von 573°C wurde nach einer Mikrowellenbestrahlung von 25 s untersucht. Der Einfluss des anisotropen Materialverhaltens von Quarz auf die Spannungsverteilung wird durch den Vergleich mit einem isotropen Materialmodell bewertet. Nach einer Mikrowellenbestrahlungszeit von 15 s werden Hauptnormalspannungen, welche die Zugfestigkeit übersteigen an den Phasengrenzen der stark absorbierenden Phase beobachtet. Aufgrund der Quarzphasenumwandlung werden nach einer Bestrahlungszeit von 25 s noch höhere Spannungen festgestellt. Im anisotropen Modell kann eine größere Volumenfraktion mit sehr hohen Hauptnormalspannungen, speziell in der mikrowellentransparenten Phase, quantifiziert werden. Unter Verwendung eines nichtlinearen Schädigungsmodells können Schädigungsinitiierungen in der Umgebung der heißesten Regionen sowie entlang der Phasengrenzen der stark absorbierenden Phase festgestellt werden. Diese Ergebnisse korrelieren qualitativ sehr gut mit den Experimenten. Zusammenfassend kann festgestellt werden, dass die Spannungs- und Schädigungsverteilung stark von der Mikrostruktur und dem mikromechanischen Verhalten der einzelnen Phasen (Quarzphasenumwandlung, anisotropes Materialverhalten) abhängt.

Für die industrielle Anwendung wurden vielzählige numerische 3D Analysen mit variierenden Bestrahlungszeiten und Mikrowellenleistungen an einem Granit Modell mit drei Gesteinskomponenten durchgeführt. Hierfür wurden gemessene dielektrische und thermomechanische Materialeigenschaften verwendet. Sowohl der Fall mit konstanter Mikrowellenleistung und unterschiedlichen Bestrahlungszeiten als auch konstanter Mikrowellenenergie und verschiedenen Kombinationen von Bestrahlungszeit und Leistung wurden analysiert. Die größte Hauptnormalspannung steigt linear mit der Bestrahlungszeit unter der Annahme einer konstanten Leistung an, wohingegen bei einer konstanten Energie ein lokales Extremum in der Spannungs-Bestrahlungszeit Kurve identifiziert werden kann. Die präsentierte 3D inhomogene Simulationskette erlaubt es, für den untersuchten Granit optimale Mikrowellenparameter zu identifizieren.

Table of contents

List of figures

List of tables

Nomenclature

Roman symbols

A	surface
a	lattice constant of 2D grid
a^*	material parameter in dilation angle function
$\underline{b}$	body force vector
B	magnetic flux density
b^*	material parameter in dilation angle function
$\{C\}$	heat capacity matrix
C	elastic constant
c	speed of light: 299792458 $^{\mathrm{m}}/_{\mathrm{s}}$
c^*	material parameter in dilation angle function
c_p	specific heat capacity at constant pressure
D	electric flux density
d	damage variable
D_p	penetration depth
$\boldsymbol{D}^{el}$	degraded elasticity matrix
$\boldsymbol{D}_0^{el}$	initial undamaged elasticity matrix
$\underline{e}$	unit vector

E	electric field strength
E_0	electric field strength at the Gaussian axis
E^*	Young's modulus
$\mathfrak{E}$	energy
$\overline{E^2}$	time averaged squared electric field
$\underline{F}$	load vector
f	filling factor
f^*	frequency
G	flow potential function
G_{IC}	critical energy release rate of mode 1
H	magnetic field strength
$\overline{H^2}$	time averaged squared magnetic field
$\boldsymbol{I}$	identity tensor
i	unit imaginary number $\sqrt{-1}$
J	current density
$\{K\}$	stiffness matrix
k	thermal conductance
K^*	magnetic current density
k^*	wave vector
K_{IC}	fracture toughness of mode 1
$\underline{n}$	unit normal vector
n	number of time increments
O	big O according to Bachmann-Landau notation
$\overline{p}$	hydrostatic pressure stress

P	power
P_{abs}	absorbed power density
$\overline{q}$	Mises equivalent stress
$\underline{Q}$	heat flux vector
$\underline{r}$	position vector
r	radial coordinate
$\overline{\boldsymbol{S}}$	deviatoric stress tensor
S	Courant number
s	body heat flux
sd	standard deviation
$\mathfrak{T}$	period
T	temperature
t	time
u	displacement
u_{t0}	cracking displacement
V	volume
w_0	waist radius of Gaussian beam at the source position
x	x-coordinate
$\bar{x}$	arithmetic mean
y	y-coordinate
z	z-coordinate

Greek symbols

α	uniaxial thermal expansion coefficient
α^*	material parameter of yield function

δ^*	loss angle
δ	disorder
δ_d	Dirac delta function
ϵ	permittivity
ϵ^*	eccentricity
ε	engineering strain
ϵ_0	permittivity of vacuum: $8.8543 \times 10^{-12}\ \mathrm{F}/\mathrm{m}$
$\tilde{\varepsilon}^{pl}$	hardening variable
$\dot{\boldsymbol{\varepsilon}}^{pl}$	equivalent plastic strain rate tensor
$\boldsymbol{\varepsilon}_v^{pl}$	viscoplastic strain tensor
γ^{pl}	maximum plastic shear strain
γ	material parameter of yield function
λ	wavelength
$\dot{\lambda}$	plastic multiplier
μ	permeability
μ^*	viscosity parameter
μ_0	permeability of vacuum: $4\pi 10^{-7}\ \mathrm{H}/\mathrm{m}$
ν	Poisson number
ω	angular frequency
ϕ	friction angle
$\partial\Omega$	surface of domain Ω
ψ	dilation angle
ρ	charge density
$\rho^\diamond$	density

ρ^*	equivalent magnetic resistivity
$r(\widehat{\overline{\boldsymbol{\sigma}}})$	stress weight factor
σ^*	conductivity
σ	Cauchy stress
σ_{b0}	biaxial compression strength
σ_{c0}	uniaxial compressive strength
σ_{cu}	ultimate uniaxial compressive strength
σ_D^*	D-conductivity
$\overline{\boldsymbol{\sigma}}$	effective stress tensor
σ_{t0}	uniaxial tensile strength
θ	temperature rise

Superscripts

el	elastic
$'$	real part of complex number
$\tilde{\cdot}$	numerical
$''$	imaginary part of complex number
pl	plastic
$\star$	desired

Subscripts

0	vacuum, initial
A	phase A (microwave absorbing)
b	basalt
c	compression
d	disc

eff	effective
end	end increment
g	granite
hom	homogeneous model
i	integration point
$inhom$	inhomogeneous model
inc	increment
j	integration point at front surface
k	amount of integration points at front surface
m	muscovite
ma	matrix
max	maximum
mw	microwave
p	plagioclase
$prov$	provided
q	quartz
r	relative
T	phase T (microwave transparent)
t	tension
th	thermal
tr	threshold
v	viscous
w	amount of integration points
x	x-component

y	y-component
z	z-component

Other symbols

$:$	tensor product
$\cdot$	vector dot product
Δ	difference, increment
$\sharp dimension$	number of dimensions
$\widehat{E}$	eigenvalue
$\{M\}$	matrix
$\langle x \rangle$	Macaulay brackets $\langle x \rangle = \frac{1}{2}(\lvert x \rvert + x)$
∇	Nabla operator
∂	partial derivative
$\boldsymbol{T}$	tensor
$\dot{t}$	time derivative
$\times$	vector cross product
$\underline{V}$	vector

Acronyms / Abbreviations

Ab	Albite
An	Anorthite
BTS	Biaxial Tensile Strength
CDP	Concrete Damaged Plasticity
CT	Computer Tomography
CTE	Coefficient of Thermal Expansion
DC	Direct Current

DEM	Discrete Element Method
EHF	Extremely High Frequency
EHT	Extra-High Tension
FDTD	Finite-Difference Time-Domain
FEM	Finite Element Method
FIM	Finite Integral Method
HFL	Heat Flux
IVOL	Integration point volume
Meep	MIT Electromagnetic Equation Propagation
MKS	Meter, Kilogram, Second
MoM	Method of Moments
PM	thermal-based Particle Modeling
SCM	Strongly Coupled Model
SHF	Super High Frequency
UCS	Uniaxial Compressive Strength
UHF	Ultra High Frequency
WCM	Weakly Coupled Model
XFEM	Extended Finite Element Method

Chapter 1

Introduction

1.1 Motivation

Classical mechanical comminution processes are highly energy intensive consuming up to 2% of the total energy in several mining countries such as USA, Australia and South Africa (Tromans, 2008). The typical energy consumption in a mineral processing plant can run into hundreds of megawatt hours per year (Jones et al., 2005). About 30 - 70% (for hard ores) of the total plant power is attributable to comminution (Napier-Munn, 1996). Only less than 1% of the applied energy is actually used to generate new surfaces (DOE, 2007; Fuerstenau and Abouzeid, 2002). In other words, almost the complete energy provided for a comminution process is dissipated in the form of heat and noise. Moreover, comminution equipment is expensive which allocates typically 20 - 50% of the capital cost of a mineral processing plant (Bradshaw et al., 2007). High energy and maintenance costs as well as strict sustainability regulations call for more efficient mining processes.

Conventional research has concentrated on incrementally improving the size reduction process (Jones et al., 2005). Recently, new ways to reduce the strength and subsequently fracture the rock have been investigated (Prokopenko, 2011):

- electrical (sputter-ion, electrostrictive and piezoelectric)
- magnetic (magnetostrictive)
- electromagnetic (laser)
- sound (impact plastic, ultrasonic)

- beam (electrons, protons and plasma)
- thermal methods (conventional heating, microwave heating)

All mentioned methods create mechanical stresses which eventually result in damage (cracks, spallation) as soon as the strength limit is reached. Thermal methods are promising and can potentially lead to step-changes in efficiency by lowering the mechanical properties (Jones et al., 2005; Prokopenko, 2011). In order to reach high stresses, prompt heating at a great depth is required. Since the thermal conductivity is low in hard rocks, conventional heating is too slow to introduce high enough stresses. In contrast, microwave heating can reach high heating rates combined with a significant irradiation depth (Prokopenko, 2011). Moreover, microwave heating offers a number of advantages compared to conventional heating such as (Haque, 1999; Jones et al., 2002; Kingman, 2006):

- non-contact heating
- rapid heating which can be faster than the heat conduction
- selective heating
- volumetric heating
- quick start-up and stopping
- heating starts from interior of the material body
- higher level of safety and automation
- energy savings

Microwave treatment of rocks has the potential to decrease the energy consumption of mineral comminution processes (Vorster et al., 2001). For example, a significant decrease in power consumption for grinding of iron ores after microwave treatment combined with an increase of liberation of individual mineral phases was shown in Walkiewicz et al. (1991). Kingman et al. (2004a) and Kingman (2006) deduced that microwave assisted comminution can become economic.

Microwave induced heating of rocks is driven by the absorption of microwaves combined with the conversion of the electromagnetic energy into heat. Unlike classical convective heating the heat flux is directly created inside the material. Furthermore, different minerals show varying microwave absorbing behaviors. Consequently, an inhomogeneous thermal field on the grain level is expected. This effect combined with the variation of thermal expansion coefficients of the minerals results in the formation of significant stresses within the rock which can exceed the strength limit leading to fracture. Recently, several numerical studies of rocks with heterogeneous microstructures showed that the resulting stresses around the phase boundaries of strong-absorbing particles are high enough to initiate cracks which can propagate further into the material (Ali and Bradshaw, 2009, 2010, 2011; Jones et al., 2005; Wang et al., 2008; Wang and Djordjevic, 2014). These thermally induced cracks may lead to a significant reduction of grinding resistance during comminution processes (Fitzgibbon and Veasey, 1990).

Not only rocks with strongly varying dielectric values of the different minerals (inhomogeneous rocks) show a potential to initiate severe damage but also rather homogeneous ones. Experiments on approximately homogeneous rocks such as basalt reveal significant microwave induced damage without any highly absorbing particles (Hartlieb et al., 2012; Peinsitt et al., 2010; Satish et al., 2006). There the thermal gradients between the homogeneously heated area and the remaining material as well as the heat loss at the free surfaces lead to high stresses. Another effect that induces stronger microwave heating of rocks is the increase of the microwave absorption with elevated temperatures of various rocks and ores. Higher temperatures enhance the absorption that can lead to extreme thermal runaway (Jerby et al., 2013; Peinsitt et al., 2010). This can even cause melting of the irradiated samples (Hartlieb et al., 2012; Hassani et al., 2016; Peinsitt et al., 2010).

1.2 State of the art

1.2.1 Experimental investigations

In the early 1960s the microwave rock breakage technique was introduced. However, due to technical issues it was not deemed economically at this time (Maurer, 1968). The technical interests in the microwave treatment of rocks was renewed by Chen et al. (1984) not earlier than in 1984. They investigated the relative transparency of minerals to microwave energy. Later, this work was extended by measuring the dielectric properties and resulting temperature levels with low power microwave sources (1 – 2.6 kW) of many common ore-forming minerals and rocks (Chunpeng et al., 1990; Church et al., 1988; McGill and Walkiewicz, 1987; Nelson et al., 1989; Santamarina, 1989; Walkiewicz et al., 1988; Webb and Church, 1986). These studies concluded that most aluminosilicates, micas, carbonates and sulphates (rock-forming minerals) showed little heating whereas most sulphides and metal oxides heated significantly when irradiating with microwaves.

The early experimental test work concerning microwave treatment of minerals was carried out using standard multi-mode cavities which are similar to the microwave ovens for domestic use. The second design is the single mode cavity which has been investigated more recently. The detailed design and the main differences are elucidated in section 2.3.2.

1.2.1.1 Multi-mode cavities

Microwave irradiation experiments on iron ores with a source of 3 kW and 2.45 GHz were performed by Walkiewicz et al. (1991). In this work it was concluded that ores containing absorbing minerals in a non-absorbing matrix were subjected to high thermal stresses. These stresses caused cracks along the grain boundaries. In this study the microwave treatment reduced the work index during standard Bond grindability tests by 10 to 24%. Experimental test results of microwave heating and drilling in basalt and granodiorite were reported by Lindroth et al. (1993). At the highest temperatures an increase of the drilling rate by a factor of up to 6.5 due to microwave treatment was determined.

Kingman et al. (2000) studied the influence of the mineralogy on the microwave heating behavior. This paper concluded that ores which have consistent mineralogy and contain a good microwave absorber in a transparent matrix are most responsive to microwave treatment. In contrast, ores containing small particles that are finely dispersed are shown to respond not favorable to microwave treatment in terms of reduction of the required grinding energy.

Vorster et al. (2001) determined the effect of microwave treatment on Neves Corvo copper ores. There a reduction in Bond work index of 70% after microwave irradiation with 2.6 kW for 90 s was observed.

In Kingman et al. (2004a) the influence of microwave treatment on lead-zinc ores on the change in strength was investigated. This study used both multi-mode and single mode cavities. Samples treated with 10 kW and 5 s in the multi-mode cavity resulted in a reduction in strength of about 50%. Moreover, the strength of the samples treated in the multi-mode cavity were related to the applied microwave power level.

Amankwah et al. (2005) investigated the microwave irradiation behavior of gold ores. An increasing temperature with higher sample mass, processing time and microwave power was determined. A reduction in crushing strength of about 31% and a decrease in Bond work index of 18% was observed. Due to microwave treatment the gold recovery by gravity separation was improved by up to 12%.

Satish et al. (2006) demonstrated that basalt specimens were responsive to low power microwave irradiation and showed an almost linear temperature increase with time. The point load strength tests gave an indication that the microwaved samples weakened caused by the different thermal heating of various mineral phases in the rock.

Olubambi et al. (2007) assessed the microwave treatment of complex sulphide ores (containing silicia, siderite, ferrous sphalerite, galena, pyrite, covelite) in a multi-mode cavity with a maximum power of 1100 W and 250 GHz frequency. A maximum temperature of 270°C was obtained after 5 min irradiating with a microwave power of 1100 W. The study showed that the application of microwave heating had a beneficial effect on the processing behavior of the sulphide ore and its dissolution in sulphuric and hydrochloric acid.

Peinsitt et al. (2010) irradiated dried basalt samples and reached a maximum temperature of 330°C after 60 s of microwave irradiation in a 3 kW multi-mode oven. Conversely, it took dried granite 300 s to reach a temperature of 220°C. The heating rates in the water-saturated samples compared to the dried ones were unchanged for basalt, doubled for granite and increased fourfold for sandstone. A significant decrease in uniaxial compressive strength and p-wave velocities reflecting a reduced rock strength was reported. In the sandstone samples the water saturation led to very large cracks and even bursting of the samples was observed.

Samouhos et al. (2012) reported a maximum temperature of 900°C after 120 s microwave irradiation of a laterite–lignite mixture with a power of 800 W. Moreover, significant mineralogical changes were determined which could be applied in mineral processing positively.

Hartlieb et al. (2012) heated cylindrical basalt samples with a microwave source of 3.2 kW in a multimode cavity. After 60 s of microwave irradiation maximum temperatures of up to 250°C at the surface and 440°C in the middle of the sample were determined. Various cracks aligned along the rotational axis as well as the radial direction of the cylindrical basalt were observed. Besides, the sound velocities dropped significantly from 5500 $^{m}/_{s}$ in the untreated basalt to 3500 $^{m}/_{s}$ after 120 s of microwave irradiation.

In Hassani et al. (2016) the temperature profiles, BTS and UCS were measured after microwave treatment in a multimode cavity (2.45 GHz) for four different rock types: mafic norite, granite from Vermont, basalt from California and basalt from China. The dry specimens were treated with power levels between 1.2 kW and 5 kW combined with irradiation times between 10 s and 120 s. No reduction in the BTS of norite was observed after 10 s of microwave treatment regardless of the applied power. After 65 s and 5 kW a reduced BTS of

50% was determined. In the granite samples the BTS decreased with 3 kW and 5 kW by 20%. In the basalt samples the BTS was reduced by up to 30% after microwave treatment with 1.2 kW and 120 s. After a treatment duration of 120 s and 3 kW the basalt sample melted in a disc-shaped area. Furthermore, microwave irradiation experiments with water saturated samples were performed. The authors concluded that high power densities are required for rock samples with very low water permeability in order to evaporate the thin water layer quickly.

1.2.1.2 Single mode cavities

Besides the multi-mode cavity experiments also single mode tests were performed in Kingman et al. (2004a). In the multi-mode case a microwave irradiation with 10 kW for 5 s was required to reach a strength reduction of about 50% whereas in the single mode cavity only an irradiation time of 0.5 s was needed. The microwave energy required for the multi-mode cavity application (10 kW for 5 s) was 13.88 $^{\mathrm{kW\,h}}\!/_{\mathrm{t}}$ compared to 1.38 $^{\mathrm{kW\,h}}\!/_{\mathrm{t}}$ in the single mode cavity (10 kW for 0.5 s).

Kingman et al. (2004b) elucidated the influence of high electric field strength microwave energy, generated in a 15 kW single mode cavity, on copper carbonatite ores. It was shown that significant reductions in strength can be achieved within very short microwave durations. Moreover, a reduction of required breakage energy during drop weight test of up to 30% was concluded for microwave energy inputs less than 1 $^{\mathrm{kW\,h}}\!/_{\mathrm{t}}$.

The influence of microwave treatment with power levels between 5 - 12 kW (single mode cavity) and 0.1 - 0.5 s on copper flotation was investigated in Sahyoun et al. (2005). It was found that initial recovery was higher in the microwave treated samples for all power levels and exposure times. At a power level of 12 kW both, grade and recovery, were significantly improved for microwave irradiation times of 0.1 and 0.5 s. Finally, a simplified economic analysis showed that the recoveries are economically attractive.

Scott et al. (2008) analyzed the effects of microwave treatment for 0.5 s at 10.5 kW in a single mode cavity of rod-milled South African carbonatite ores on the liberation spectrum. They observed intergranular fractures which were introduced between microwave absorbing and non-absorbing minerals. The recovery of copper increased from 81% in the untreated material up to 89.5% after microwave treatment.

Localized microwave irradiation of basalt samples resulted in rock melting in Jerby et al. (2013). There the thermal runaway instability caused by the improved absorption behavior with increasing temperatures was responsible for the melting. Therefore, high thermal stresses were introduced causing crack formation.

1.2.1.3 Comparison between single and multi-mode cavity

The multi-mode cavity design leads to low electric field strengths. Due to the low power densities and the long treatment time used as a consequence, energetically inefficient treatment of the mineral ores was concluded (Ali and Bradshaw, 2010).

Single mode cavities are capable of generating heating rates which are many orders of magnitude higher than those produced by multi-mode cavities (Jones et al., 2007; Metaxas and Meredith, 1993; Whittles et al., 2003). In general, for the same power applied a single mode cavity will establish significantly higher electric field strengths compared to the multi-mode design (Metaxas and Meredith, 1993). This cavity design results in very high heating rates even exceeding $1000\,{}^{\circ}\mathrm{C}/\mathrm{s}$ in strong absorbers, which offers the ability to heat materials that would appear transparent to microwaves in ordinary multimode cavities (Jones et al., 2007). Additionally, experimental studies by Kingman et al. (2004a,b) and Sahyoun et al. (2005) revealed that high power density treatments (typically >3 kW in single mode cavities) allowed for a similar degree of microwave-assisted breakage at significantly lower energy inputs than treatments at low power density (typically <3 kW in multimode cavities) due to significantly higher heating rates. However, the dimensions of the single mode cavity are restricted to the magnitude of the wavelength whereas the multi-mode cavity can be larger and thus contains more material.

1.2.2 Numerical investigations

Various numerical studies have been performed in order to understand microwave induced stresses and the resulting damage formation. Mainly, two different cases have been investigated: In the first case a rather homogeneous rock, such as basalt or sandstone, is treated as a continuum and irradiated with microwaves. The second case describes the condition where an inhomogeneous rock, such as granite, is treated with microwaves. There different phases with varying absorption behavior are considered.

1.2.2.1 Homogeneous models

In the numerical part of Hartlieb et al. (2012) the temperature and resulting stress field in a cylindrical basalt sample were evaluated by means of a finite element model. There the constant heat source was determined by a thermal FE-model and by comparing it to experimental results of the temperature distribution inside the sample. The highest stresses were determined at the free surfaces of the basalt cylinder.

Hassani et al. (2016) modeled the 3D electromagnetic distribution and the resulting temperature field of a cuboid homogeneous basalt sample in a closed cavity by the multiphysics program *COMSOL*. The thermo-mechanical as well as dielectric material properties were assumed constantly (not a function of temperature) during the simulations. After a microwave treatment for 120 s with a source of 3 kW and 2.45 GHz a maximum surface temperature of about 280°C was reached. The results of the thermal simulations were compared with experiments where the temperature was measured at different positions in the depth of the material. This was achieved by cutting twelve slabs out of the material prior to the microwave treatment. The electric field intensity diminished exponentially within the rock.

1.2.2.2 Inhomogeneous models

The thermo-mechanical response of a single spherical pyrite particle embedded in a calcite rock during short-pulse microwave heating was assessed by Salsman et al. (1996). There a 2D finite element analysis was performed by assuming a constant microwave absorption only in the pyrite particle. They observed significant temperature differences between the two phases and predicted tensile stresses along the pyrite-calcite interface which exceeded the tensile strength of common rock materials (especially in the 10^{14} $^{W}/_{m^3}$ for 40 μs and 10^{12} $^{W}/_{m^3}$ for 40 ms cases). However, for a given provided microwave energy the temperature difference between the phases as well as the peak tensile stress in the host rock was reduced for smaller mineral particle size. Finally, they suggested that the economy of microwave assisted grinding can be improved significantly by using very high power for a short period of time.

Later, Whittles et al. (2003) performed a more sophisticated 2D finite difference analysis with multiple quadratic (1x1 mm^2) pyrite grains distributed in a calcite host. Only the pyrite phase was heated by a varying power density between 3×10^9 $^{W}/_{m^3}$ and 9×10^9 $^{W}/_{m^3}$ as a function of the temperature (conditions which correlate to 2.6 kW, 2.45 GHz multimode cavity). Also, a case with 10^{11} $^{W}/_{m^3}$ was investigated which could be achieved by microwave heating in a single mode cavity with 15 kW. A Mohr-Coulomb model was applied to model an uniaxial compression test in order to quantify the impact of the microwave irradiation. With the multimode conditions (2.6 kW, 2.45 GHz) a reduction of the unconfined compressive strength from 126 MPa to 79 MPa after 30 s of microwave treatment was observed. A higher power density of 10^{11} $^{W}/_{m^3}$ (15 kW single mode) led to a stronger reduction to 25 MPa after 1 s irradiation. Finally, they concluded that by increasing the power density, higher stresses were introduced for much lower energy inputs.

A single pyrite spherical grain surrounded by a calcite matrix has been investigated in Jones et al. (2005) by a 2D finite difference analysis. There the crack pattern caused by the microwave heating was analyzed by a Mohr-Coulomb constitutive model. Once again, calcite was deemed as non absorbing for microwaves and four different constant power densities in the range between 10^8 $^{W}/_{m^3}$ to 10^{11} $^{W}/_{m^3}$ were applied. Due to high thermo-mechanical stresses both radial tensile fracturing within the calcite matrix and shear failure concentrated along the grain boundary of the pyrite particle were observed. By reducing the size of the pyrite sphere more energy was required to sufficiently raise the temperature in order to introduce stresses which are high enough to cause damage.

Later, Jones et al. (2007) expanded the study to a 2D model containing various quadratic pyrite particles (1% pyrite) in a calcite matrix. Again, a finite difference analysis combined with a Mohr-Coulomb model was used. The pyrite grains were heated with microwaves and power densities ranging from 10^9 $^{W}/_{m^3}$ to 10^{10} $^{W}/_{m^3}$ and for a pulsed simulation from 10^{13} $^{W}/_{m^3}$ to 2×10^{15} $^{W}/_{m^3}$. Afterwards, an unconfined compressive strength test was simulated. A reduction in the UCS of up to 50% after microwave treatment with 10^{15} $^{W}/_{m^3}$ was determined. In general, a greater reduction in strength was observed for a given total energy input when the exposure time was reduced. Based on their models the authors suggested that for future microwave comminution power densities between 10^{10} $^{W}/_{m^3}$ and 10^{12} $^{W}/_{m^3}$ combined with irradiation times in the range of 0.2 s and 0.002 s would be favorable.

Wang et al. (2008) applied the thermal-based particle modeling (PM), which is a discrete element method (DEM), on 2D models with spherical pyrite grains (between one and nine) in a calcite matrix. This numerical method (PM) decomposed the total interaction force between the discrete particles to a mechanical and a thermal part. In that paper it was assumed that the introduced microwave energy input completely contributed to enhance the repulsive bond strength between the particles. Moreover, a linear elastic-brittle type of interaction force was considered. Micro-cracking along the phase boundaries was observed. Furthermore, the fracture density defined as the ratio between broken bonds to original bonds increased as the microwave irradiation time was extended.

Ali and Bradshaw (2009) performed 2D finite difference simulations including a Mohr-Coulomb model on randomly distributed rectangular microwave absorbing grains (10%) in a transparent matrix. Two different binary systems were investigated: galena (absorbing) - caclite (transparent) and magnetite (absorbing) - dolomite (transparent). However, in this study only the thermo-mechanical material properties were effected by the different binary system but not the microwave absorption behavior. A set of simulations was performed with a power density of 10^{10} $^{W}/_{m^3}$ corresponding to a pulsed microwave source and another with 10^9 $^{W}/_{m^3}$ for the case of a 30 kW source with a frequency of 2.45 GHz. Tensile failure was determined around the strongly absorbing grains. After 0.75 ms of microwave treatment with a power density of 10^{10} $^{W}/_{m^3}$ 36.6% of the grain boundary zone exceeded the tensile strength. For the same energy input but different power densities and irradiation times, different grain boundary damage was determined (74.3% with 10^{10} $^{W}/_{m^3}$, 0.001 s compared to less than 50% with 10^9 $^{W}/_{m^3}$, 0.01 s). This paper concluded that the amount of damage was depended on the ore mineralogy and its texture.

Ali and Bradshaw (2010) applied DEM to analyze the damage behavior of different models containing 10% microwave absorbing galena grains with varying sizes and shapes inside a transparent calcite matrix. In order to use this simulation method, a set of numerical material tests had to be performed to obtain the micro properties of the discrete elements. The thermal material in these models was represented as a network of heat reservoirs and thermal pipes. A power density of 10^9 $^{W}/_{m^3}$ corresponding to a 30 kW source and 10^{11} $^{W}/_{m^3}$ for short-duration pulsed type microwave equipment were investigated. It was concluded that for the same energy input and mineralogy the amount of micro-cracks and the crack pattern were depended on the applied power density and on the size of the absorbing grains. By applying high power densities, it was possible to reduce the energy input and to localize the damage near the grain boundaries independent from the shape of the microwave absorbing grains. Finally, the work concluded that a higher power density was required in order to treat fine-grained ores at economic energy inputs.

Confined particle bed crushing combined with microwave treatment was investigated by Ali and Bradshaw (2011) using a DEM code. The microstructure of the single particles was modeled with 10% randomly dispersed microwave absorbing galena particles in a calcite matrix. In the 2D model, 25 particles were arranged between rigid walls and were first microwave irradiated and then compressed. The research showed that microwave irradiation at high power densities changed the progeny size distribution significantly and improved the degree of liberation of an ore in the confined bed breakage test. However, a considerable increase in liberation was also observed at the ore treated at low power density compared with the untreated ore, although the resulting progeny size distributions were fairly similar.

Moreover, they obtained higher liberations for both, the microwave treated but also the untreated ores, when the crushing velocity decreased.

Wang and Djordjevic (2014) studied the microwave induced stresses and cracks in a 2D circular plate containing a disc-shaped microwave absorbing pyrite particle in a transparent calcite matrix. The simulations were performed with a finite element (FE) program assuming axial symmetry. In order to simulate the rock breakage behavior they used a thermal fracture model which had been developed by Wang (2013) based on a texture-based finite element method (FEM) modeling technique (cf. Wang (2015)). The initial cracks were caused by tensile thermo-mechanical stresses which gradually propagate in radial direction from the calcite matrix. The cracks in the non-absorbing matrix (calcite) were initiated close to the phase boundary. The main factor affecting the location of maximum stress was the thermal expansion of the matrix. The longer the exposure time, the further away the peak stress is from the pyrite-calcite interface. Finally, it is concluded that larger pyrite grains increased the maximum stress whereas a larger matrix had the opposite effect.

1.3 Framework of the thesis

In this thesis a numerical framework which is capable of determining microwave induced stresses and damage is presented. In order to define the properties and functionality which should be considered for such an approach, the previous numerical studies are summarized and simplifications used therein are outlined.

1.3.1 Summary of previous numerical works

Although various numerical studies have been conducted since the 1990s, important features of hard rocks have been omitted and, furthermore, no comprehensive conclusions on the microwave induced fragmentation behavior could be drawn (see section 1.2). The microwave induced stresses and damage in inhomogeneous rocks have been investigated by some studies in a two dimensional artificial model microstructure with one or more circular or quadratic microwave absorbing particles inside a transparent matrix. However, only a constant absorbed power density in each of the absorbing particles was assumed. In other words, the distribution of the electromagnetic field inside the inhomogeneous rock has not been considered. Moreover, the micromechanical behavior of the rock such as the anisotropic grain behavior and the phase transformation of the quartz grains have not been investigated in those studies. Finally, the thermo-mechanical and dielectric material properties were taken from literature and were not measured in most of the papers.

In one of the papers dealing with homogeneous rocks, the distribution of the electromagnetic field was considered. However, no coupling between temperature and electromagnetic field has been taken into account (Hassani et al., 2016). Moreover, only a temperature but no stress field has been calculated.

Certainly, all studies have concluded that the numerical analysis allows valuable insight in the formation of microwave induced stresses as well as on the identification of optimum irradiation parameters. Based on the drawbacks of the previous work outlined in this section the objective of the work has been defined as follows:

1.3.2 Aim of the work

The aim of this work is to derive a three dimensional numerical framework in order to calculate the microwave induced stresses and damage in various inhomogeneous as well as homogeneous hard rocks with different irradiation parameters. Taking into account the resulting transient temperature and stress fields helps to better understand the complex nature of the formation of microwave induced damage. Hence, the main aims of the work are to:

1. Determine temperature, stress and damage fields in realistic homogeneous as well as inhomogeneous hard rock models.
2. Gain deeper understanding of the microwave induced damage behavior.
3. Suggest optimal microwave irradiation parameters to achieve maximum induced damage.

In order to reach these objectives a sophisticated simulation chain is derived. Beside the main aims described above the following properties should be considered in the numerical work for an **inhomogeneous** hard rock:

- 3D microstructure with realistically resolved grains
- Randomly assigned phases to the grains with an arbitrary number of materials
- Electromagnetic field inside the microstructure
- Phase transformation of quartz from α to β at 573°C

- Anisotropic behavior of quartz grains
- Non-linear constitutive law in order to determine the damage pattern

The simulation chain has to be flexible in order to include further properties easily or to perform parameter studies automatically. The following properties for the models describing the **homogeneous** hard rocks should be taken into account:

- Electromagnetic field inside the homogeneous model
- Coupling between temperature and electromagnetic field
- Phase transformation of quartz from α to β at 573°C

Chapter 2

Principles of microwave heating

2.1 History

During the Second World War intensive research on high-definition radar led to the development of microwave frequencies. In particular, the magnetron valve as a microwave generator of high power output with good efficiency was invented (Meredith, 1998). In 1945, Percy Spencer filed a patent describing the principle of a microwave oven. In the early fifties a commercial microwave oven was first developed by the Raytheon company, where Spencer was employed. The devise was about two meters high and weighed hundreds of kilos. Ovens for domestic purposes became available in the early 1960s (Osepchuk, 1984).

For more than 40 years microwave ovens have been important devices in most kitchens exploiting the advantage of fast cooking times and energy savings compared to conventional cooking (Meredith, 1998). More recently the microwave technology has also been applied in industrial applications such as mineral processing and comminution.

2.2 Basic concept of microwave heating

Microwaves are electromagnetic waves within a specific range of frequency (cf. figure 2.2). In order to understand the microwave induced heating the electromagnetic theory is shortly summarized.

2.2.1 Electromagnetic waves

In 1864, Maxwell expressed four existing equations in a set of expression with complete generality and conciseness (Fließbach, 2012). These vector equations describe the electric and magnetic fields as well as their interactions with each other and with the material (Schwab, 2013). Electromagnetic phenomena can be determined by these expressions and, hence, also electromagnetic waves (Jackson, 2011). For heterogeneous, isotropic, linear and stationary media Maxwell's equations in the time domain, differential form and using the MKS units are presented (equations 2.1 - 2.4, cf. Gupta and Wong (2007); Jackson (2011)):

Faraday's law of induction

$$\underline{\nabla} \times \underline{E} = -\frac{\partial \underline{B}}{\partial t} \tag{2.1}$$

Maxwell's modified Ampere's circuital law

$$\underline{\nabla} \times \underline{H} = \frac{\partial \underline{D}}{\partial t} + \underline{J} \tag{2.2}$$

Gauss's law for the electric field

$$\underline{\nabla} \cdot \underline{D} = \rho \tag{2.3}$$

Gauss's law for the magnetic field

$$\underline{\nabla} \cdot \underline{B} = 0 \tag{2.4}$$

In equations 2.1 - 2.4 $\underline{E}$, $\underline{H}$, $\underline{B}$, $\underline{D}$ and $\underline{J}$ define the electric and magnetic field strength vector, magnetic and electric flux density vector and current density vector, respectively. The electric sources are described by the charge density ρ. All these field quantities, $\underline{E}$, $\underline{H}$, $\underline{B}$, $\underline{D}$, $\underline{J}$ and ρ are time-varying and each is a function of the space coordinates and time $\underline{E} = f(x,y,z,t)$ (Balanis, 2012). The flux densities can be related to the field strengths and the current density to the electric field strength by the constitutive relations (equations 2.5- 2.7, cf. Balanis (2012); Jackson (2011)).

$$\underline{D} = \epsilon \underline{E} \tag{2.5}$$

$$\underline{J} = \sigma^* \underline{E} \tag{2.6}$$

$$\underline{B} = \mu \underline{H} \tag{2.7}$$

In the constitutive equations ϵ is the permittivity, σ^* the conductivity and μ the permeability of the material. In general, these parameters are functions of the applied field strength, the spatial location within the material, the direction of the applied field and the frequency of operation (Balanis, 2012). The permittivity can be decomposed into the relative permittivity ϵ_r which is a function of the material and the permittivity of free space ϵ_0 (equation 2.8). The same decomposition can be performed for the permeability of a material as it can be seen in equation 2.9 (Cassidy, 2009).

$$\epsilon = \epsilon_r \epsilon_0 \tag{2.8}$$

$$\mu = \mu_r \mu_0 \tag{2.9}$$

The Maxwell's relations present a set of partial differential equations which can be solved analytically only for very specific cases with simple geometry and boundary conditions. For more sophisticated conditions numerical methods have to be used. These are summarized in chapter 3.1.1. One of the simplest solutions of Maxwell's equations is the propagation of plane waves in unbounded loss-free space. Figure 2.1 shows a plane wave which consists of an electric and a complementary magnetic field vector orthogonal to each other and to the direction of propagation (Meredith, 1998).

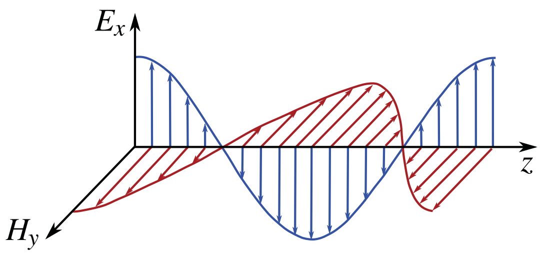


Fig. 2.1 Plane electromagnetic wave (Balanis, 2012).

The correlation between orientation of the electric and the magnetic field of the plane electromagnetic wave is defined by Maxwell's equations (Scheck, 2006). According to equation 2.2, a time-varying electric field generates a magnetic field and vice versa, the change of the magnetic field induces an electric field (equation 2.1). The plane waves travel in vacuum with a velocity c according to equation 2.10 (Jackson, 2011).

$$c = \sqrt{\frac{1}{\mu_0\ \epsilon_0}} = \sqrt{\frac{1}{4\pi 10^{-7} \times 8.8543 \times 10^{-12}}} = 299792458\ {}^{\mathrm{m}}/_{\mathrm{s}} \tag{2.10}$$

As can be seen in equation 2.10, the velocity of the wave is independent of the frequency in free space and is therefore the same for the whole electromagnetic spectrum including visible light (Meredith, 1998). According to their wavelengths, the electromagnetic waves can be classified into radio waves, microwaves, infrared, light waves (which is the visible region for humans), ultraviolet, X-rays and gamma rays. The whole electromagnetic spectrum is visualized in figure 2.2.

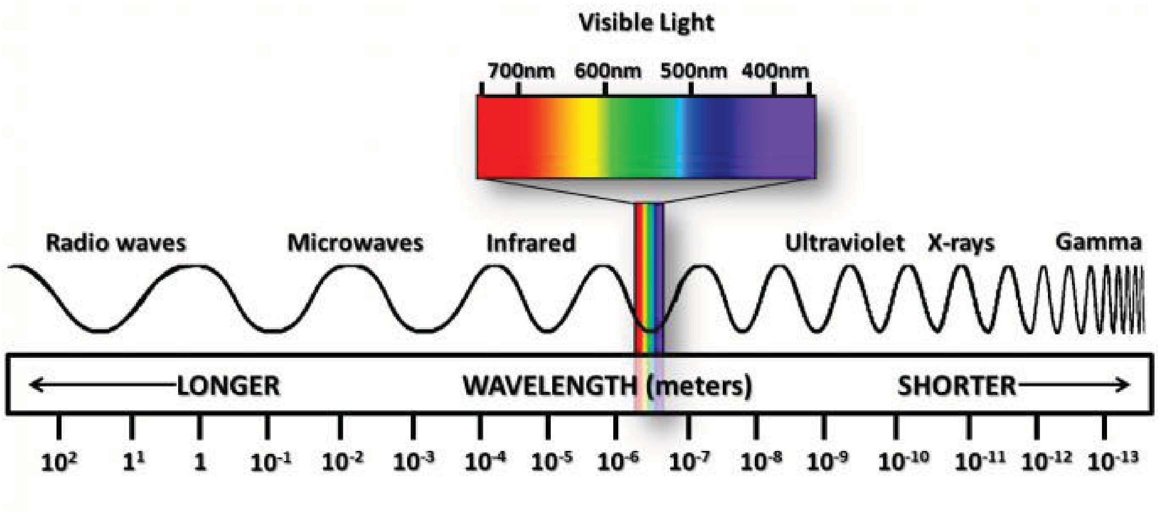


Fig. 2.2 Electromagnetic spectrum (Lambert and Edwards, 2016).

Microwaves are defined as electromagnetic waves in a frequency range between 300 MHz and 300 GHz (cf. figure 2.2). According to equation 2.11 this corresponds to a wavelength in vacuum λ_0 of 1 m for a frequency f^* of 300 MHz and 1 mm in the 300 GHz case.

$$c = f^* \lambda_0 \tag{2.11}$$

The microwaves can be further classified into ultra high frequency (UHF: 300 MHz - 3 GHz), super high frequency (SHF: 3 GHZ - 30 GHz) and extremely high frequency (EHF: 30 GHz - 300 GHz). In industrial and domestic applications a frequency of f^* = 2.45 GHz (λ_0 = 12.24 cm) is most commonly used (Haque, 1999).

2.2.2 Physical mechansim of microwave heating

When microwaves are applied to a material, three different mechanisms can be observed. These basic classes of microwave-material interaction are visualized in figure 2.3 (Church et al., 1988).

Conductors do not allow microwaves to pass through and reflect the total amount of energy at the surface. Transparent materials transmit the microwaves without causing heating.

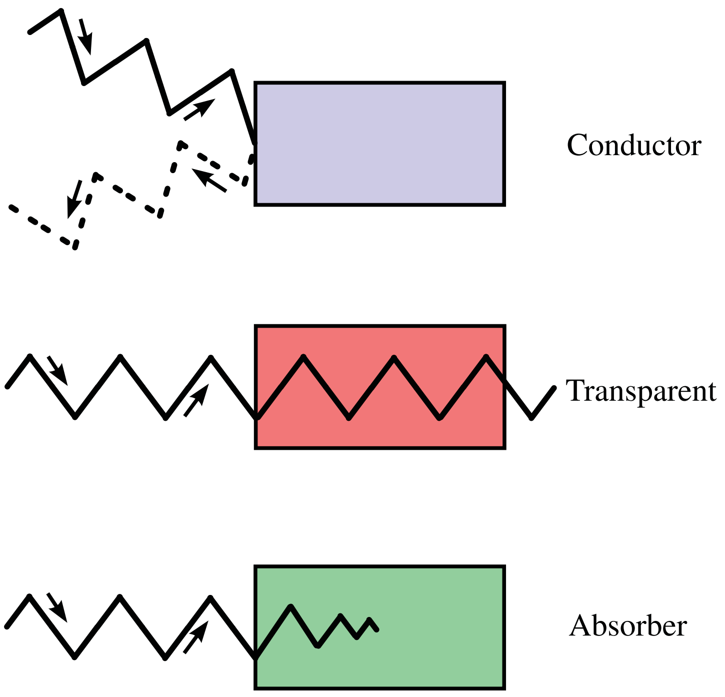


Fig. 2.3 Interaction of microwaves with material (Church et al., 1988; Haque, 1999).

The absorbers convert the electromagnetic energy mainly into heat. These three basic classes represent ideal fictitious cases which cannot be found in nature. Real materials are always a combination of the three mentioned extreme cases (Church et al., 1988).

In general, metals have free electrons and high conductivity and are therefore categorized as conductors (Gupta and Wong, 2007). Conductors are often used as waveguides for microwaves (Haque, 1999). Glass, ceramic and air are treated as transparent materials and, therefore, absorb microwaves to a negligible extent and allow the waves to pass through easily (Gupta and Wong, 2007). Salt water and many food products are excellent absorbers for microwave energy and are categorized as dielectrics (Gupta and Wong, 2007; Haque, 1999). However, also magnetic materials such as ferrites are heated by microwaves due to the interaction with the magnetic component of the electromagnetic wave (Gupta and Wong, 2007).

2.2.2.1 Dielectric heating on a microstructure scale

Microwave irradiation of a dielectric material leads to losses and heating. Ideal dielectrics do not contain free charges (as would be the case in conductors) and, moreover, their atoms and molecules are microscopically neutral as illustrated in figure 2.4. When external electromagnetic fields are applied, the respective centroids of the positive and negative charges can shift slightly in positions relative to each other, thus creating numerous electric dipoles (cf. figure 2.5). This process is called electronic polarization (Balanis, 2012).

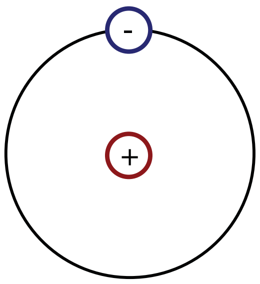


Fig. 2.4 Atom without applied field (Balanis, 2012).

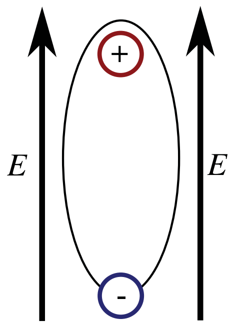


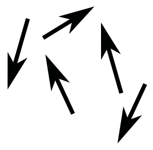
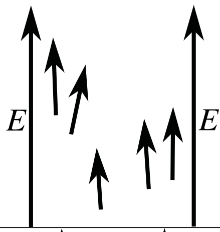
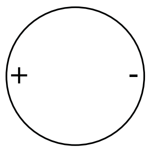
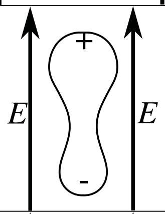
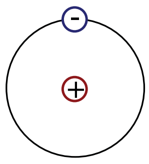
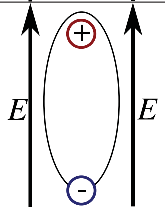
Fig. 2.5 Atom under applied field (Balanis, 2012).

Unlike the dielectric materials, the negative and positive charges would move to the surface of the material in the case of conductors. There the charges are separated by macroscopic distances. This does not occur in dielectric materials which represent the fundamental difference between bound charges in dielectrics and true charges in conductors. Besides the electronic polarization of dielectrics during microwave treatment there are two additional mechanism causing polarization. All three types of polarization are summarized in figure 2.1 (Balanis, 2012).

Dipole or Orientational Polarization is evident in materials that randomly contain dipole moments in the absence of any applied field. When an electric field is applied the dipole moments try to align along the field. An example of such a material is water or any other polar material. **Ionic or Molecular Polarization** takes place in materials that possess positive and negative ions and they tend to displace themselves when an electric field is applied (for example NaCl). **Electronic Polarization** is evident in most of the dielectric materials (Balanis, 2012).

The initial propagating electromagnetic wave inside a dielectric material supplies energy in form of acceleration for the separation of the charges. This generates a small displacement current that produces radiating electromagnetic energy. The localized energy is slightly out of phase with the incident wave which results in a ’slow down’ of the main body of the

Table 2.1 Different polarization mechanism (Balanis, 2012).

Mechanism	**Without field**	**With field**
Dipole or Orientational Polarization		
Ionic or Molecular Polarization		
Electronic Polarization		

propagating wave. Finally, the microwave heating of the material is caused by the energy transfer from the vibrations of the charged particles to mechanical vibrations of atoms or molecules in the lattice of the solid material (Cassidy, 2009).

The described polarization effects are typically frequency-dependent. At low frequencies the particles are able to 'react quickly' to the applied field and stay in phase with its changes. With very high frequencies the dipoles have no time to move and stay at their initial positions. In the microwave frequency range (which is between these two extreme cases) the dipoles have enough time to respond to the alternating wave but the frequency is high enough that the movement does not precisely follow the field. When the dipoles reorientate to align with the field, the field has already changed and a phase difference evolves between field and dipole which causes heating (Cassidy, 2009).

2.2.2.2 Dielectric heating on a macroscopic scale

On a macroscopic scale the microwave heating of a material depends on the constitutive parameters ϵ_r and μ_r. These material constants are complex numbers (cf. equations 2.12 and

2.13) (Gupta and Wong, 2007).

$$\epsilon_r = \epsilon_r' - i\epsilon_r'' \tag{2.12}$$

$$\mu_r = \mu_r' - i\mu_r'' \tag{2.13}$$

The imaginary part of the permittivity ϵ_r'', which is also called loss factor, governs the absorption behavior of the material (Gupta and Wong, 2007). Another parameter often used to quantify the efficiency of the material to convert microwave energy into heat is the loss tangent $\tan\delta^*$ or loss angle δ^* (cf. equation 2.14) (Meredith, 1998).

$$\tan\delta^* = \frac{\epsilon_r''}{\epsilon_r'} \tag{2.14}$$

The loss angle δ^* describes the phase difference between the oscillating electric field and the polarization of the material (Gupta and Wong, 2007). Moreover, the electromagnetic wave inside a dielectric material has the same frequency as in vacuum but a different wavelength $\lambda_{material}$. In equation 2.15, the reduction of the wavelength compared to vacuum (cf. equation 2.11) is represented by the relative permittivity (Meredith, 1998).

$$\lambda_{material} = \frac{\lambda_0}{\sqrt{\epsilon_r}} \tag{2.15}$$

The microwave heating on a macroscopic scale is quantified by the absorbed power density (P_{abs}). In general, this property is the sum of the electric and magnetic losses (cf. equation 2.16) (Lee and Kim, 2011).

$$P_{abs} = \omega\epsilon_0\epsilon_r''\overline{E^2} + \omega\mu_0\mu_r''\overline{H^2} \tag{2.16}$$

In equation 2.16, $\overline{E^2}$ defines the time averaged squared electric field and $\overline{H^2}$ that of the magnetic field. Moreover, ω is the angular frequency ($= 2\pi f^*$). The magnetic losses contribute to the absorbed power density when magnetic materials are investigated (Cassidy, 2009). Since rocks are non-magnetic materials, equation 2.16 can be reduced to expression 2.17.

$$P_{abs} = 2\pi f^*\epsilon_0\epsilon_r''\overline{E^2} \tag{2.17}$$

In practice the complex relative permittivity ϵ_r varies with frequency, temperature, moisture content, physical state (solid or liquid) and composition (Meredith, 1998). For constant conditions some literature values are summarized in table 2.2.

The amplitude of the microwaves propagating into a dielectric material diminishes owing to absorption of power. The field intensity and the associated power flux density fall exponentially with distance from the surface if no reflected waves occur in the material. Furthermore, the power dissipation also falls exponentially from the surface. In order to assess the rate of decay of the power dissipation a parameter D_p is proposed (Meredith, 1998). The penetration depth D_p is defined as the depth into the material at which the power flux has fallen to $1/e$ (≈ 0.368) of the surface value (equation 2.18) (Metaxas and Meredith, 1993).

$$D_p = \frac{\lambda_0}{2\pi\sqrt{2\epsilon_r'}} \frac{1}{\sqrt{\sqrt{1+\left(\frac{\epsilon_r''}{\epsilon_r'}\right)^2}-1}} \tag{2.18}$$

Since the imaginary part is usually smaller than the real part of the relative permittivity in rock materials, equation 2.18 can be simplified (equation 2.19). The maximum error of this simplification is 10% (Meredith, 1998).

$$D_p \approx \frac{\lambda_0\sqrt{\epsilon_r'}}{2\pi\epsilon_r''} \tag{2.19}$$

Equation 2.19 shows that the penetration depth decreases with increasing imaginary part of the relative permittivity (ϵ_r'') and increases with the real part (ϵ_r'). Besides the description of the decay behavior of the power dissipation, the penetration depth also allows estimations about the thermal field inside the material. In a semi-infinite slab of ideal material (i.e. constant permittivity regarding the temperature) and with a plane wave at normal incidence the temperature rise θ_z along the depth z is described in equation 2.20 (Meredith, 1998).

$$\theta_z = \theta_0 \, e^{\frac{-z}{D_p}} \tag{2.20}$$

By integration of equation 2.20 it can be derived that the heat dissipated between the surface and the depth D_p is 63.2% of the total dissipated heat (Meredith, 1998).

Table 2.2 Permittivity parameters and corresponding penetration depths at room temperature.

Component	***f* [GHz]**	ϵ_r' []	ϵ_r'' []	$\boldsymbol{D_p}$ **[cm]**	**Source**
Water	3	76.7	12.04	1.2	Santamarina (1989)
Granite	3	5.0 - 5.8	0.03 - 0.2	118.5 - 19.2	Santamarina (1989)
Moyite-granite	9.37	5.27	0.20	5.8	Zheng et al. (2005)
Basalt	3 9.37	5.4 - 9.4 7.55 - 7.86	0.08 - 0.8 0.56 - 0.44	46.2 - 6.1 2.5 - 3.2	Santamarina (1989) Zheng et al. (2005)
Marble (dry)	3	8.7	0.14	33.5	Santamarina (1989)
Gabbro	3 9.37	7 7.95	0.13 0.40	32.4 3.6	Santamarina (1989) Zheng et al. (2005)
Muscovite	0.915 1 2.45 2.5 3	4.23 4.46 8.69 1.62 5.4	0.0008 0.0034 0.091 0.005 0.0016	1.3×10^4 2963.7 63.1 485.8 2309.9	Church et al. (1988) Church et al. (1988) Nelson et al. (1989) Meredith (1998) Santamarina (1989)
Plagioclase (in basalt)	9.37	6.56	0.32	4.1	Zheng et al. (2005)
Plagioclase (An_{100})	9.9	7.2	0.004	323.3	Kržmanc et al. (2003)
Plagioclase (Ab_{100})	10.4	5.9	0.013	85.7	Kržmanc et al. (2003)
Plagioclase ($An_{40}Ab_{60}$)	10.4	5.7	0.005	219.1	Kržmanc et al. (2003)
Quartz	1 9.37 10	3.89 4.3 3.8	0.0005 0.0026 0.0004	1.9×10^4 406.1 2325.3	Church et al. (1988) Zheng et al. (2005) Ishii (1995)
Fused quartz	9.37	3.71	0.0033	297.2	Zheng et al. (2005)
Ilemnite	2.45 9.37	23.6 54.3	11.2 32.58	0.8 0.1	Nelson et al. (1989) Zheng et al. (2005)
Pyroxene (in anorthosite)	9.37	10.2	1.62	1	Zheng et al. (2005)

In table 2.2 the penetration depth is calculated by using the simplified formula (equation 2.19). The literature data reveal a strong variation of the permittivity values not only between the different rocks and minerals but also within the same material. Due to the strong variations, a temperature-dependent measurement device for bulk rock blocks was devised in Hartlieb et al. (2016) and the data are used for the three dimensional three component simulations.

2.3 Microwave equipment

In order to irradiate a sample with microwaves three major mechanical parts are necessary (cf. figure 2.6). These parts are the microwave source, waveguide and applicator (Haque, 1999).

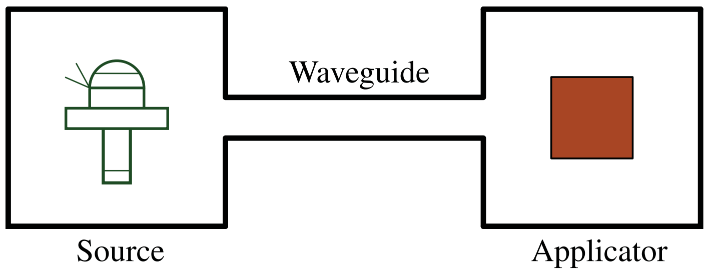


Fig. 2.6 Major components of the microwave heating system (Haque, 1999).

Microwaves are generated in the source by a provided direct current source. Most commonly magnetrons are used in industrial applications as the microwave source (cf. section 2.3.1). Other possibilities to generate microwaves are klystrons, traveling wave thermionic devices, gyrotrons, magnicons, ubitrons and peniotrons (Ishii, 1995). After the microwave is generated, a waveguide transmits the energy from the source to the applicator. Typically, a waveguide is a tube made of a conductive material. In the applicator the microwave interacts with the dielectric material. Based on the design of the applicator / cavity two different concepts are found (cf. chapter 2.3.2).

2.3.1 Magnetron

The generation of microwave power is highly efficient in magentrons with a typical efficiency greater than 70% at 2.45 GHz. Figure 2.7 visualizes the principal components and the function of a high power magnetron (Meredith, 1998).

Basically, a magnetron contains a cylindrical cathode, a circular anode with radial slots forming resonators tuned to the desired microwave frequency, a magnet (permanent in small magnetrons and electromagnet for high power) and a probe antenna or slot coupled to the resonator. The whole devise is enclosed in a vacuum envelope. To start the microwave source, a high value DC EHT voltage is applied between cathode and anode. The heated cathode emits electrons by the resulting applied electric field. The magnetic field induced by the outer magnet is perpendicular to the electric field. Since the electron moves perpendicular to the magnetic field the Lorentz force acts on it. This causes that the electron travels along a spiral

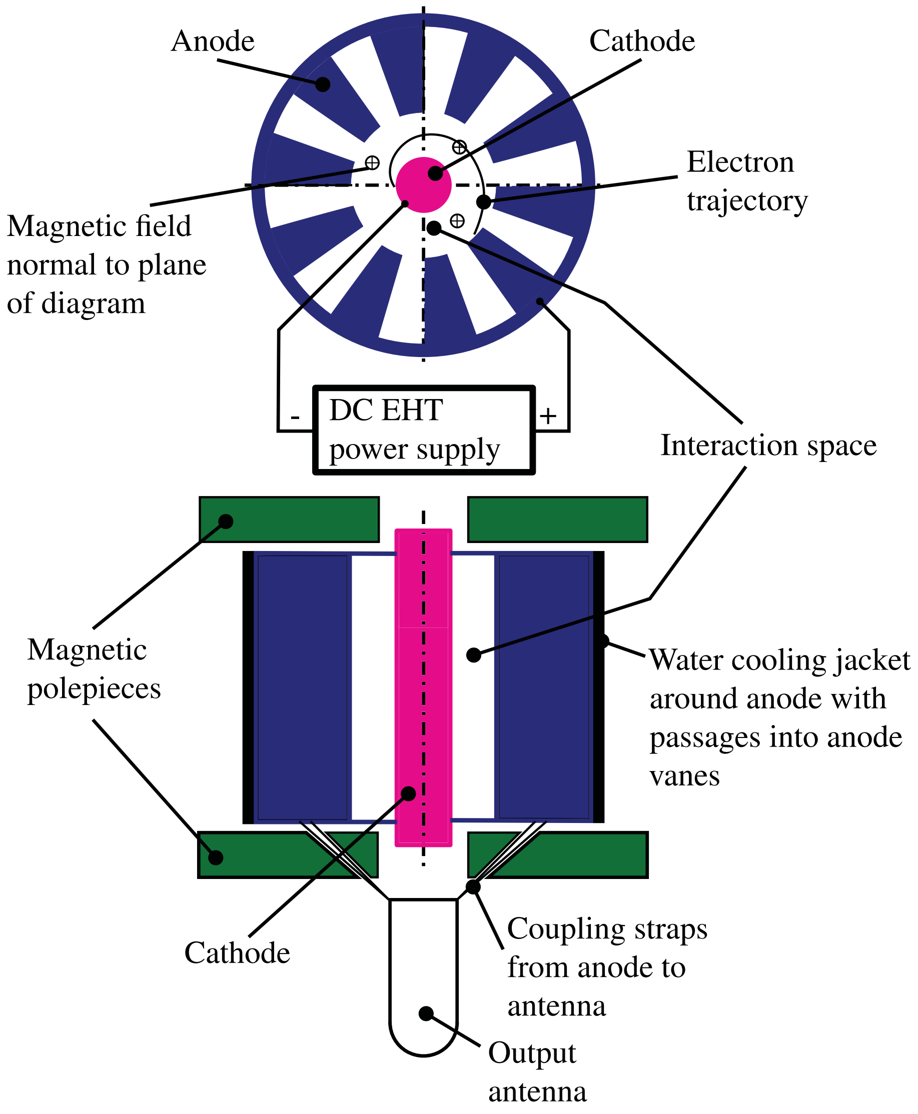

Anode
Cathode
Electron trajectory
Magnetic field normal to plane of diagram
DC EHT power supply
-
+
Interaction space
Magnetic polepieces
Water cooling jacket around anode with passages into anode vanes
Cathode
Coupling straps from anode to antenna
Output antenna

Fig. 2.7 Principle design and function of a magnetron (Meredith, 1998).

path in the space between anode and cathode (cf. figure 2.7). Since the anode has radial slots (resonant cavities), the accelerated electrons oscillate. Finally, the energy is coupled from one of the resonant cavities to the waveguide (Meredith, 1998).

For various applications the output power of the magnetron has to be adjusted. This can be done by following strategies (Ishii, 1995):

- Pulse output power of the magnetron
- Adjust anode current
- Change magnetic field

- Adjust microwave energy that enters the load

2.3.2 Cavity design

Two different types of applicators are available: multimode and single mode cavities. Figure 2.8 illustrates the principle structure of a rectangular multimode cavity which is quite similar to those found in a conventional kitchen microwave oven.

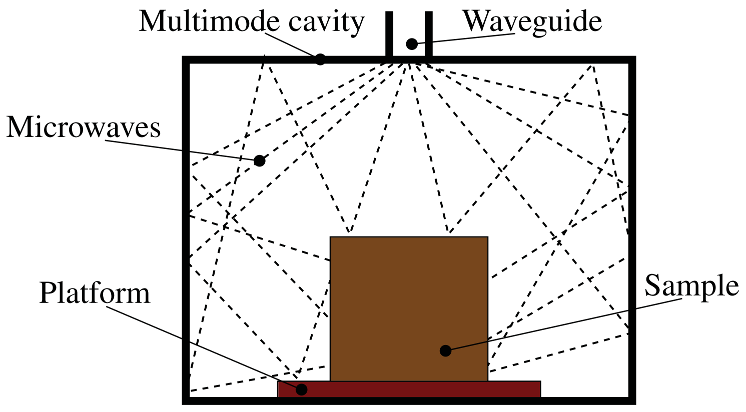


Fig. 2.8 Principle structure of a rectangular multimode microwave cavity (Pickles, 2009).

From a mechanical point of view they are very simple, essentially comprising a closed metal box with accessories. The multimode oven supports a large number of resonant high-order waveguide-type modes simultaneously which give a resultant field pattern (Meredith, 1998). However, this construction leads to low electric field strengths. Due to the low power densities and the long treatment time used as a consequence, energetically inefficient treatment of the mineral ores is concluded (Ali and Bradshaw, 2010).

Single mode cavities are capable of generating heating rates which are many orders of magnitude higher than those produced by multimode cavities (Jones et al., 2007; Metaxas and Meredith, 1993; Whittles et al., 2003). With the single mode cavity (cf. figure 2.9) the rock sample is directly irradiated by an open-end waveguide.

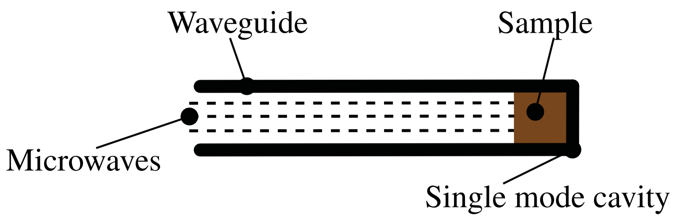


Fig. 2.9 Principle structure of a single mode microwave cavity (Pickles, 2009).

Essentially, a single mode cavity consists of a metallic enclosure and a waveguide. The dimension of the single mode cavity is of the order of the applied microwave wavelength. The launched microwave signal of the correct polarization suffers multiple reflections between preferred directions within the waveguide. Due to the superposition of incident and reflected waves, a standing wave pattern is achieved (Metaxas and Meredith, 1993). In general, for the same power applied a single mode cavity establishes significantly higher electric field strength compared to the multimode design (Metaxas and Meredith, 1993). Experimental studies by Kingman et al. (2004a,b) and Sahyoun et al. (2005) revealed that high power density treatments (typically > 3 kW in single mode cavities) allowed for a similar degree of microwave-assisted breakage at significantly lower energy inputs than treatments at low power density (typically < 3 kW) in multimode cavities due to significantly higher heating rates.

Chapter 3

Simulation strategy

In order to reach the aims outlined in section 1.3.2 various numerical calculations with different solvers have to be performed. The general procedure of the simulations is displayed in figure 3.1.

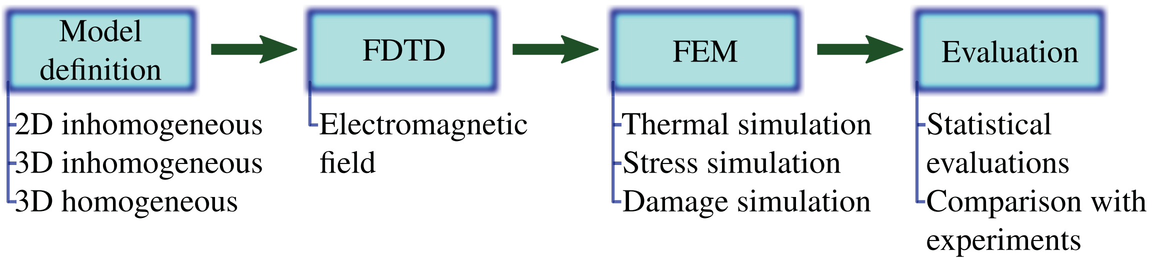


Fig. 3.1 Overview of the simulation procedure.

First, a model of the hard rock including the properties which should be investigated is built (figure 3.1). Mainly 3D models with either inhomogeneous microstructure or homogeneous material definition are used. In preliminary simulations also a 2D model with disc shaped inclusion geometry has been analyzed. In the next step, the electromagnetic field caused by the microwave irradiation of the rock sample is evaluated (figure 3.1). This calculation is performed by a FDTD (Finite-Difference Time-Domain) algorithm which is presented in section 3.1. Afterwards, the resulting transient thermal field is calculated in a FE analysis (figure 3.1). The displacement field does not have any influence on the temperatures so there is only a weak coupling between the thermal analysis and the displacement analysis. Hence, the stress field can be computed in a subsequent analysis following the thermal analysis. A concrete damaged plasticity material model is used in the FE calculation to assess the microwave induced damage (cf. section 3.2.2). Finally, the resulting stress and damage distribution are assessed and statistically evaluated.

In this chapter the theoretical framework of the used numerical methods is summarized. The technical implementation of the rock models in the numerical programs and the necessary amendments to the codes are discussed in detail in chapter 4.

3.1 FDTD method

3.1.1 Numerical solution of Maxwell's equations

In order to solve Maxwell's equations (equations 2.1 - 2.4) within a sophisticated structure and for three dimensions, numerical methods have to be used. There are many numerical methods available to solve these partial differential equations: the Finite-Difference Time-Domain (FDTD) method, the Finite Element Method (FEM), the Method of Moments (MoM), the Transmission Line Matrix (TLM) method, the Finite Integral Method (FIM) and others (Zhao et al., 2011). The FEM method uses the variational principle as well as shape functions to solve Maxwell's equations. This method leads to sparse matrices and complex structures can be analyzed (Zhao et al., 2011). In the MoM a boundary-integral formulation is used which requires the derivation of geometry-specific Green's function. Moreover, a system of linear equations having dense, complex valued and full coefficient matrices are generated by the MoM method (Umashankar, 1988). TLM uses the analogy between wave propagation in space and voltage and current propagation in a transmission line. The computational domain is modeled by a transmission line network (Zhao et al., 2011). The FIM transforms Maxwell's equations in their integral form into a linear system of equations. The domain is discretized by cuboids and integrated over their surfaces (Weiland, 1977). The details of the FDTD algorithm are summarized in section 3.1.2. Compared to the other mentioned methods the FDTD offers the following advantages (Taflove, 1988, 2005):

- **Fully explicit algorithm:** Since no linear algebra is used which would limit the size of FE electromagnetic models to generally fewer than 10^6 field unknowns, no intrinsic upper bound has to be considered in FDTD. Moreover, the FDTD code can easily be parallelized to run on cluster-computers.
- **Accurate and robust:** FDTD uses a second order accurate central-difference approximation for space and time derivatives of the electric and magnetic field.
- **Treats impulsive behavior naturally:** Due to the used time-domain technique, FDTD directly calculates the impulse response of an electromagnetic system.

- **Treats nonlinear behavior naturally**
- **Systematic approach**: By calculating the electromagnetic field in a new model only a new mesh has to be generated rather than the complex reformulation of an integral equation. Unlike MoM, no structure-dependent Green's functions are required.

3.1.2 FDTD analysis

In this section the FDTD algorithm is introduced. First, the general equations and solving strategies are described and in section 3.1.3 the open source software *Meep* (Oskooi et al., 2010) and its special features are introduced.

3.1.2.1 Maxwell's equations in FDTD notation

In order to derive the full set of FDTD equations, Maxwell's relations are used (equations 2.1 - 2.4) and combined with the constitutive equations 2.5 - 2.7. By assuming a source-free domain with constitutive parameters that are independent of time and using a MKS (Meter, Kilogram, Second) system, equations 3.1 and 3.2 can be written as (Taflove, 1988)

$$\frac{\partial \underline{H}}{\partial t} = -\frac{1}{\mu}\nabla \times \underline{E} - \frac{\rho^*}{\mu}\underline{H} \tag{3.1}$$

$$\frac{\partial \underline{E}}{\partial t} = \frac{1}{\epsilon}\nabla \times \underline{H} - \frac{\sigma}{\epsilon}\underline{E} \tag{3.2}$$

For the FDTD purpose Faraday's law 2.1 is extended by an equivalent magnetic resistivity ρ^* (equation 3.1). This term is added in order to yield symmetric curl equations and to consider magnetic loss mechanisms. Assuming that the constitutive parameters are isotropic and using a rectangular coordinate system a set of six scalar equations can be worked out (equations 3.3 - 3.8) (Taflove, 1988).

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \rho^* H_x\right) \tag{3.3}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} - \rho^* H_y\right) \tag{3.4}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \rho^* H_z\right) \tag{3.5}$$

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x\right) \tag{3.6}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y\right) \tag{3.7}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right) \tag{3.8}$$

This set of six coupled partial differential equations forms the basis for the FDTD algorithm. These equations are discretized by central finite difference expressions (Taflove, 1988).

3.1.2.2 Yee algorithm

For the discretisation of the coupled differential expressions 3.3 - 3.8 the Yee algorithm (Yee, 1966) is used. According to Yee's grid, a point in a rectangular lattice can be described by the notation defined in equation 3.9 (Taflove, 1988; Yee, 1966).

$$(i, j, k) = (i\Delta x, j\Delta y, k\Delta z) \tag{3.9}$$

In equation 3.9 Δx, Δy and Δz are the lattice space increments in the respective coordinate directions and i, j, k are integers. Any function of space and time can be described by using the notation in equation 3.10 (Taflove, 1988; Yee, 1966).

$$F^n(i, j, k) = F(i\Delta x, j\Delta y, k\Delta z, n\Delta t) \tag{3.10}$$

Δt in equation 3.10 is the time increment and n an integer number. The second order accurate time space centered finite difference expression for a function F is denoted in equations 3.11 and 3.12 (Taflove, 1988; Yee, 1966).

$$\frac{\partial F^n(i,j,k)}{\partial x} \approx \frac{F^n\left(i+\frac{1}{2},j,k\right) - F^n\left(i-\frac{1}{2},j,k\right)}{\Delta x} + O\left(\Delta x^2\right) \tag{3.11}$$

$$\frac{\partial F^n(i,j,k)}{\partial t} \approx \frac{F^{n+\frac{1}{2}}(i,j,k) - F^{n-\frac{1}{2}}(i,j,k)}{\Delta t} + O\left(\Delta t^2\right) \tag{3.12}$$

Moreover, Yee positioned $\underline{E}$ and $\underline{H}$ in the finite difference grid with a shift of half a cell (cf. figure 3.2). This pattern guarantees the accuracy of equation 3.11 as well as the definition of the derivatives of the expressions 3.3 - 3.8. Moreover, the Yee grid algorithm evaluates $\underline{E}$ and $\underline{H}$ at alternate half time steps in order to achieve the accuracy of the leapfrog time-stepping (equation 3.12, figure 3.2) (Taflove, 1988, 2005; Yee, 1966).

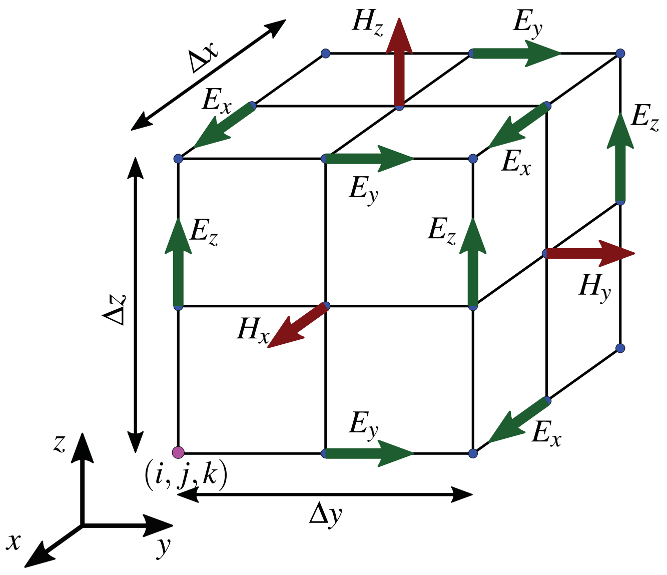


Fig. 3.2 Position of the electric and magnetic field components in a cubic unit cell of the Yee FDTD grid (Yee, 1966).

With the Yee grid mentioned above and the central differences defined in equations 3.11 and 3.12, the explicit expressions of the magnetic and electric field components are derived (cf. the two examples in equations 3.13 and 3.14) (Taflove, 1988).

$$H_x^{n+1/2}\left(i,j+\tfrac{1}{2},k+\tfrac{1}{2}\right)=\frac{1-\frac{\rho^*\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)\Delta t}{2\mu\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)}}{1+\frac{\rho^*\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)\Delta t}{2\mu\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)}}H_x^{n-1/2}\left(i,j+\tfrac{1}{2},k+\tfrac{1}{2}\right)+$$
$$\frac{\frac{\Delta t}{\mu\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)}}{1+\frac{\rho^*\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)\Delta t}{2\mu\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)}}\left(\frac{E_y^n\left(i,j+\frac{1}{2},k+1\right)-E_y^n\left(i,j+\frac{1}{2},k\right)}{\Delta z}+\right.$$
$$\left.\frac{E_z^n\left(i,j,k+\frac{1}{2}\right)-E_z^n\left(i,j+1,k+\frac{1}{2}\right)}{\Delta y}\right) \tag{3.13}$$

$$E_z^{n+1}\left(i,j,k+\tfrac{1}{2}\right)=\frac{1-\frac{\sigma\left(i,j,k+\frac{1}{2}\right)\Delta t}{2\epsilon\left(i,j,k+\frac{1}{2}\right)}}{1+\frac{\sigma\left(i,j,k+\frac{1}{2}\right)\Delta t}{2\epsilon\left(i,j,k+\frac{1}{2}\right)}}E_z^n\left(i,j,k+\tfrac{1}{2}\right)+$$
$$\frac{\frac{\Delta t}{\epsilon\left(i,j,k+\frac{1}{2}\right)}}{1+\frac{\sigma\left(i,j,k+\frac{1}{2}\right)\Delta t}{2\epsilon\left(i,j,k+\frac{1}{2}\right)}}\left(\frac{H_y^{n+1/2}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-H_y^{n+1/2}\left(i-\frac{1}{2},j,k+\frac{1}{2}\right)}{\Delta x}+\right.$$
$$\left.\frac{H_x^{n+1/2}\left(i,j-\frac{1}{2},k+\frac{1}{2}\right)-H_x^{n+1/2}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)}{\Delta y}\right) \tag{3.14}$$

In the two presented examples (equations 3.13, 3.14) of the discretized six finite-difference equations, it can be seen that the new value of the field component only depends on its previous value and on the preceding values of the neighboring components. Since no simultaneous equation solution is needed, the calculations of the field vectors can easily be parallelized. Due to the interleaved electric and magnetic field components in space of the used Yee grid, the continuity of tangential $\underline{E}$ and $\underline{H}$ across an interface of dissimilar material is naturally satisfied. Moreover, the Yee algorithm implicitly enforces the two Gauss's law relations (equations 2.3 and 2.4, for mathematical proof see chapter 3.6.9 in Taflove (2005)). The presented time-stepping algorithm is non-dissipative which means that the electromagnetic waves do not decay due to numerical artifacts of the time stepping (Taflove, 1988, 2005; Yee, 1966).

When solving the finite-difference equations 3.13 and 3.14 some limitations in the choice of the time and space increments have to be considered in order to obtain accurate results. These numerical issues are summarized in the following (Taflove, 1988, 2005):

3.1.2.3 Numerical FDTD issues

Numerical stability

In order to ensure a stable time-stepping algorithm, which is exemplified in equations 3.13 and 3.14, a numerical stability criterion has to be fulfilled. To this end, the Courant number S is defined (equation 3.15) (Taflove, 2005).

$$S = \frac{c\Delta t}{\Delta x} \tag{3.15}$$

The Courant number S describes the relation between the time increment and the grid constant which is exemplarily denoted as Δx in equation 3.15. Usually in finite difference algorithm (and also in *Meep*) the spatial increments $\Delta x = \Delta y = \Delta z$ are forced to be equal. The Courant number has to fulfill the inequality 3.16 in order to ensure numerical stability. This relation is derived by considering a complex-frequency analysis and assuring that the numerical waves have zero exponential attenuation per grid space. For details the reader is referred to chapter 4.7.1 in Taflove (2005).

$$S \leq \frac{1}{\sqrt{\sharp dimension}} \tag{3.16}$$

In equation 3.16 the variable $\sharp dimension$ defines the number of the dimension. For example, in the 3D case $\sharp dimension$ would be 3 (Taflove, 2005). In a FDTD program such as *Meep*, S is defined by the user by considering the inequality 3.16. For standard 3D simulations a Courant number S of 0.5 is recommended (Meep, 2016).

Numerical dispersion

Not only the Courant number S has to be within certain boundaries but also the grid constant or lattice space increment $\Delta x = \Delta y = \Delta z$ is bounded in order to prevent numerical dispersion. With this non-physical dispersion the phase velocity of numerical modes in the FDTD lattice differs from c at an amount varying with the wavelength, direction of propagation and spatial discretization. If this phenomenon is not considered, pulse distortion, artificial anisotropy and pseudorefraction will occur (Taflove, 1988, 2005).

In order to estimate the numerical dispersion caused by the Yee discretization, a monochromatic traveling plane wave trial solution is substituted into the set of finite difference equations (cf. exemplary equations 3.13, 3.14). After various mathematical manipulations (for details the reader is referred to section 4.2 in Taflove (2005)) the numerical dispersion relation is derived (equation 3.17) (Taflove, 2005).

$$\left(\frac{1}{c\Delta t}\sin\left(\frac{\omega\Delta t}{2}\right)\right)^2 = \left(\frac{1}{\Delta x}\sin\left(\frac{\tilde{k}_x^*\Delta x}{2}\right)\right)^2 + \left(\frac{1}{\Delta y}\sin\left(\frac{\tilde{k}_y^*\Delta y}{2}\right)\right)^2 + \left(\frac{1}{\Delta z}\sin\left(\frac{\tilde{k}_z^*\Delta z}{2}\right)\right)^2 \tag{3.17}$$

Equation 3.17 relates the numerical wave vector $\tilde{k}_x^*$, $\tilde{k}_y^*$, $\tilde{k}_z^*$ to the angular frequency of the wave ω, the time increment Δt and space increments Δx, Δy, Δz. The numerical dispersion is compared to the analytical dispersion for a plane wave propagating in three dimensions in a homogeneous lossless medium (equation 3.18) (Taflove, 2005).

$$\left(\frac{\omega}{c}\right)^2 = (k_x^*)^2 + (k_y^*)^2 + (k_z^*)^2 \tag{3.18}$$

It can be shown that the numerical dispersion (equation 3.17) and the analytical dispersion relation (equation 3.18) are equal in the limit case where Δx, Δy, Δz and Δt approach zero. This suggests that the numerical dispersion can be reduced to any desired value if sufficiently small time and space increments are chosen. For a Courant number S of 0.5 at least 20 FDTD grid points per wavelength λ are recommended (Taflove, 2005).

3.1.3 *Meep* features

In the thesis at hand the open source FDTD program *Meep* (Meep, 2016; Oskooi et al., 2010) is used to calculate the electromagnetic waves inside the 3D rock model. This chapter briefly summarizes the features of *Meep* and presents the implementation of a complex permittivity ϵ as well as the definition of sources.

3.1.3.1 *Meep* characteristics

Meep is an open source program for Unix-like systems which is written in *C++*. The major features of the program are (Meep, 2016; Oskooi et al., 2010):

- Simulations in 1D, 2D, 3D and cylindrical coordinates
- Parallel simulations with MPI standard
- Arbitrary anisotropic electric permittivity ϵ and mangnetic permeability μ

- PML (perfectly matched layers) absorbing boundaries and / or perfect conductor and / or Bloch-periodic boundary conditions
- Exploitation of symmetries to reduce the computational effort
- Completely scriptable (*Scheme* or *C++*)
- Field output in HDF5 format which can easily be used by other *C++* programs to perform further calculations

Moreover, *Meep* uses dimensionless units where the permittivity and permeability of vacuum ϵ_0, μ_0 as well as the speed of light c is set to one. Therefore, the scale invariant behavior of Maxwell's equations is exploited (Meep, 2016).

3.1.3.2 Complex permittivity

The complex part of the relative permittivity (ϵ_r''), which describes the absorption behavior of the material, is implemented in *Meep* using the conductivity σ_D^*. Based on this modeling technique, complex electric $\underline{E}$ and magnetic fields $\underline{H}$ (which would double the required memory) and their concomitant numerical issues are avoided. The σ_D^* is determined by using equation 3.19 (Meep, 2016).

$$\sigma_D^* = \frac{2\pi f \epsilon_r''}{\epsilon_r'} \tag{3.19}$$

Note that the frequency f has to be inserted in *Meep* units (scaled by the speed of light c) in equation 3.19. The conductivity required in *Meep* σ_D^* differs from the conductivity used in Maxwell's equations (equations 2.1 - 2.4) by the permittivity (cf. equation 3.20) (Meep, 2016).

$$\sigma^* = \sigma_D^* \epsilon \tag{3.20}$$

3.1.3.3 Source definition

Instead of *hard sources* where specific field values are assigned at the desired source position, *Meep* uses *soft sources*. In the case of *hard sources* the source would cause non-physical scatter waves (e.g. reflected waves) that impinge on the source. Conversely, *soft sources* or *transparent sources* are transparent to other electromagnetic waves due to the linearity of Maxwell's equations. With this method equivalent electric $\underline{J}$ and magnetic currents $\underline{K^*}$ are defined which produce the incident source wave. These equivalent currents are evaluated by using the "total-field / scattered-field" approach which is a special case of the principle of equivalence in electromagnetism (Oskooi and Johnson, 2013; Taflove, 2005).

In the "total-field / scattered-field" approach an infinite medium with the desired incident fields $\underline{H}^{\star}$ and $\underline{E}^{\star}$ is considered first. Afterwards, an imaginary surface $\partial\Omega$ is introduced which divides the space into an inside region, where the desired fields are still present, and an external region, where all fields are zero. In order to fulfill Maxwell's equations electric $\underline{J}$ and magnetic currents $\underline{K}^{*}$ are constructed. In the last step perfectly matched layers (PML) are added to truncate the computation domain and insert a scatterer or other object into the internal region. In this region the electric $\underline{J}$ and magnetic currents $\underline{K}^{*}$ produce the desired incident field where the field in the external region is only the scattered field. With the described approach equation 3.21 is derived (Oskooi and Johnson, 2013). The mathematical details are formulated in Oskooi and Johnson (2013).

$$\begin{bmatrix} \underline{J} \\ \underline{K}^{*} \end{bmatrix} = \delta_d\left(\partial\Omega\right) \begin{bmatrix} \underline{n} \times \underline{H}^{\star} \\ -\underline{n} \times \underline{E}^{\star} \end{bmatrix} \tag{3.21}$$

In equation 3.21 $\underline{n}$ is the unit inward-normal vector of the surface $\partial\Omega$ and δ_d the Dirac delta function. In the section 4.1.3.4 formula 3.21 is used to define the sources of the investigated models in *Meep*.

3.2 FE method

3.2.1 FEM introduction

The finite element method is chosen to calculate the transient temperature as well as the stress / damage field. In the thesis at hand an implicit finite element analysis within the commercial program *Abaqus* (Abaqus, 2014) is used. The finite element method applies the principle of virtual displacements combined with shape functions to obtain a weak form of the boundary value problem which is solved by the FE program. In the decoupled thermo-mechanical analysis two numerical calculations have to be performed. For details on the FE framework the reader is referred to the *Abaqus* manual (Abaqus, 2014) or to Bathe (2007).

3.2.1.1 Thermal model

In the thermal FE model the heat conduction equation (equation 3.22) is solved (Abaqus, 2014).

$$\rho^{\diamond} c_p \frac{\partial T\left(\underline{r},t\right)}{\partial t} = \underline{\nabla} \cdot \left(k \underline{\nabla} T\left(\underline{r},t\right)\right) + s\left(\underline{r},t\right) \tag{3.22}$$

In this equation, T defines the temperature, $\rho^{\diamond}$ the density, c_p the specific heat, k the thermal conductivity, $\underline{r}$ the position vector and s the body heat flux. The discretization of equation 3.22 is given by expression 3.23 (Abaqus, 2014).

$$\{C\}\underline{\dot{T}} + \{K\}\underline{T} = \underline{Q} \tag{3.23}$$

In equation 3.23, $\{C\}$ is the heat capacity matrix, $\{K\}$ the conductivity matrix, $\underline{T}$ the temperature vector and $\underline{Q}$ the heat flux vector. Since the $\{C\}$ and $\{K\}$ matrices are temperature-dependent, a Newton-Raphson algorithm is used to solve the nonlinear set of equations. The time-dependent problem is solved by means of a finite difference scheme (Abaqus, 2014).

3.2.1.2 Stress model

In order to derive the stress formation the equilibrium equation for the static case has to be solved (equation 3.24) (Abaqus, 2014).

$$\underline{\nabla}\boldsymbol{\sigma} + \underline{b} = \underline{0} \tag{3.24}$$

$\boldsymbol{\sigma}$ in equation 3.24 defines the Cauchy stress tensor and $\underline{b}$ the body force vector. The Cauchy stress tensor $\boldsymbol{\sigma}$ is coupled with the strain tensor by the constitutive equation. In the thesis at hand both a linear elastic material model to calculate the elastic stresses and a nonlinear material model to identify damage is used. In the case of small strains the strain tensor $\boldsymbol{\varepsilon}$ can be derived from the displacement vector $\underline{u}$ by equation 3.25 (Abaqus, 2014).

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\underline{\nabla u} + \underline{u\nabla}\right) \tag{3.25}$$

Finally, the finite element equation for a static model is derived (equation 3.26).

$$\{K\}\underline{u} = \underline{F} \tag{3.26}$$

In this equation $\{K\}$ denotes the stiffness matrix and $\underline{F}$ the load vector.

3.2.2 Concrete damaged plasticity model

3.2.2.1 Material model characteristics

The constitutive behavior of hard rocks is described in the thesis at hand by the *Concrete Damaged Plasticity* (CDP) material model available in the commercial FE program *Abaqus* (Abaqus, 2014). The model is based on the plastic-damage model for concrete presented by Lubliner et al. (1989) with adaptions suggested by Lee and Fenves (1998). The applicability of the model for modeling the constitutive behavior of hard rocks has been investigated in various studies (Busetti et al., 2012a,b; Mikl-Resch et al., 2015). The main properties of the CDP model are (Abaqus, 2014):

- Constitutive model for concrete and other quasi-brittle materials
- Different yield strengths in tension and compression can be used
- CDP uses isotropic damaged elasticity combined with isotropic tensile and compressive plasticity to represent inelastic material behavior
- CDP can be used for monotonic, cyclic and dynamic loading
- Combination of nonassociated multi-hardening plasticity and scalar isotropic damaged elasticity to describe the fracturing process
- CDP can be used in conjunction with viscoplastic regularization to enhance the convergence

The CDP model is a continuum plastic-damage model which captures tensile cracking and compressive crushing of quasi brittle materials. The uniaxial material behavior under tension and compression is visualized in figures 3.3 and 3.4 (Abaqus, 2014; Lee and Fenves, 1998; Lubliner et al., 1989).

By loading the CDP model under uniaxial tension the stress-strain response first follows the linear elastic line until the failure stress σ_{t0} is reached (cf. figure 3.3). At this point micro-cracks are initiated in the material. With increasing strains the material softens due to the formation of micro-cracks inducing strain localization in the structure. Conversely, under compression the plastic material behavior (beyond the uniaxial compressive strength σ_{c0}) is characterized by stress hardening (until σ_{cu}) followed by strain softening due to crushing. In both uniaxial loading conditions the elastic stiffness is degraded in the strain softening regime. When the integration point is unloaded from a point on the softening curve the elastic

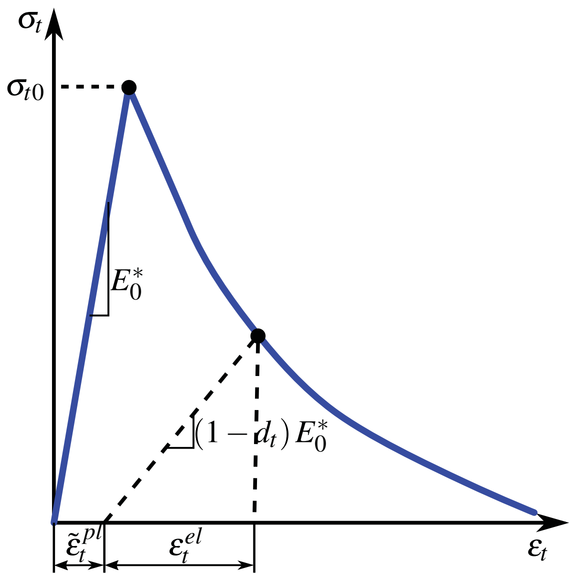


Fig. 3.3 Uniaxial tension behavior of the CDP model (Abaqus, 2014).

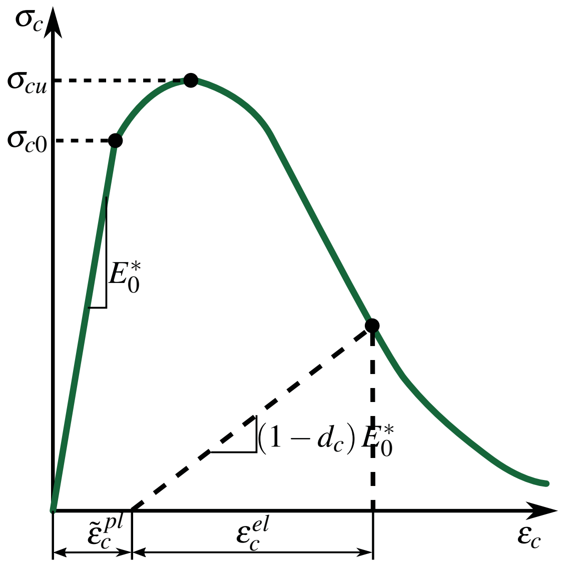


Fig. 3.4 Uniaxial compression behavior of the CDP model (Abaqus, 2014).

stiffness is reduced through damage (Abaqus, 2014; Lee and Fenves, 1998; Lubliner et al., 1989).

The implementation of the material characteristic described above for a three dimensional case into the FE framework follows the equations summarized in section 3.2.2.2. For details the reader is referred to chapter 23.6.3 of the *Abaqus Analysis User's Guide* (Abaqus, 2014) or chapter 4.5.2 of the *Abaqus Theory Guide* (Abaqus, 2014).

3.2.2.2 Constitutive relations

Stress-strain relations

By considering the inelastic material behavior the total strain tensor ($\boldsymbol{\varepsilon}$) is decomposed in an elastic ($\boldsymbol{\varepsilon}^{el}$) and a plastic part ($\boldsymbol{\varepsilon}^{pl}$) (equation 3.27). For small strains the plastic stain $\boldsymbol{\varepsilon}^{pl}$ is the remaining strain in the relaxed configuration (Abaqus, 2014; Lemaître and Chaboche, 1990).

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{el} + \boldsymbol{\varepsilon}^{pl} \tag{3.27}$$

Afterwards, the multiaxial stress-strain relation of the CDP model is derived by rearranging equation 3.27 and considering a scalar damage variable d (equation 3.28). The variable d takes values between zero, which corresponds to an undamaged material, and one, i.e. a fully damaged material. Therefore, damage in the CDP model (cracking and crushing) results in an isotropic reduction of the stiffness (Abaqus, 2014).

$$\boldsymbol{\sigma} = (1-d)\boldsymbol{D}_0^{el} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{pl}) = \boldsymbol{D}^{el} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{pl}) \tag{3.28}$$

In equation 3.28, $\boldsymbol{\sigma}$ describes the Cauchy stress tensor, $\boldsymbol{D}_0^{el}$ the initial undamaged elasticity matrix and $\boldsymbol{D}^{el}$ the degraded elasticity matrix. Following the continuum damage mechanics notation an effective stress tensor $\overline{\boldsymbol{\sigma}}$ is defined in the undamaged regime (equation 3.29) (Abaqus, 2014).

$$\overline{\boldsymbol{\sigma}} = \boldsymbol{D}_0^{el} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{pl}) \tag{3.29}$$

Hardening variables and their evolution

In order to identify damage states two independent hardening variables in tension $\tilde{\varepsilon}_t^{pl}$ and compression $\tilde{\varepsilon}_c^{pl}$ are introduced, which are referred to as equivalent plastic strains in tension and compression (equation 3.30). Increasing values of the hardening variables correspond to microcracking and crushing of the material. These two variables control the degradation of the elastic stiffness as well as the yield surface (cf. equation 3.34). The evolution of the hardening variables is described by the equivalent plastic strain rate tensor $\dot{\tilde{\boldsymbol{\varepsilon}}}^{pl}$ given by equation 3.31 (Abaqus, 2014).

$$\tilde{\boldsymbol{\varepsilon}}^{pl} = \begin{pmatrix} \tilde{\varepsilon}_t^{pl} \\ \tilde{\varepsilon}_c^{pl} \end{pmatrix} \tag{3.30}$$

$$\dot{\tilde{\boldsymbol{\varepsilon}}}^{pl} = \widehat{\boldsymbol{h}} \cdot \widehat{\dot{\boldsymbol{\varepsilon}}}^{pl} \text{where } \widehat{\dot{\boldsymbol{\varepsilon}}}^{pl} = \begin{pmatrix} \widehat{\dot{\varepsilon}}_1^{pl} \\ \widehat{\dot{\varepsilon}}_2^{pl} \\ \widehat{\dot{\varepsilon}}_3^{pl} \end{pmatrix} \tag{3.31}$$

The equivalent plastic strain rate tensor $\dot{\tilde{\boldsymbol{\varepsilon}}}^{pl}$ is a function of the matrix $\widehat{\boldsymbol{h}}$ and the eigenvalues of the plastic strain rate tensor $\dot{\boldsymbol{\varepsilon}}^{pl}$ ($\widehat{\dot{\boldsymbol{\varepsilon}}}^{pl}$) defined as $\widehat{\dot{\varepsilon}}_1^{pl} \geq \widehat{\dot{\varepsilon}}_2^{pl} \geq \widehat{\dot{\varepsilon}}_3^{pl}$. The matrix $\widehat{\boldsymbol{h}}$ is expressed by using the stress weight factor $r(\widehat{\overline{\boldsymbol{\sigma}}})$ (equation 3.32) (Abaqus, 2014; Lee and Fenves, 1998).

$$\widehat{\boldsymbol{h}} = \begin{pmatrix} r(\widehat{\overline{\boldsymbol{\sigma}}}) & 0 & 0 \\ 0 & 0 & -(1-r(\widehat{\overline{\boldsymbol{\sigma}}})) \end{pmatrix} \tag{3.32}$$

$$r(\widehat{\overline{\boldsymbol{\sigma}}}) = \frac{\sum_{i=1}^{3} \langle \widehat{\overline{\sigma}}_i \rangle}{\sum_{i=1}^{3} |\widehat{\overline{\sigma}}_i|}, \ 0 \leq r(\widehat{\overline{\boldsymbol{\sigma}}}) \leq 1 \tag{3.33}$$

$\widehat{\overline{\sigma}}_i$ in equation 3.33 defines the principal stresses of the effective stress tensor $\overline{\boldsymbol{\sigma}}$ and $\langle \cdot \rangle$ is the Macaulay brackets defined by $\langle x \rangle = \frac{1}{2}(|x| + x)$. If all three principal stresses are positive, the stress weight factor $r(\widehat{\overline{\boldsymbol{\sigma}}})$ is one and zero if all principal stresses are negative.

Yield function

The yield function F (equation 3.34) defines a surface in the effective stress space which determines the onset of failure or plasticity. The function F can only take values less or equal to zero (Abaqus, 2014; Lee and Fenves, 1998; Lemaître and Chaboche, 1990).

$$\begin{aligned} F(\overline{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\varepsilon}}^{pl}) =& \frac{1}{1-\alpha^*}(\overline{q} - 3\alpha^*\overline{p} + \beta(\tilde{\boldsymbol{\varepsilon}}^{pl})\langle \widehat{\overline{\sigma}}_{max} \rangle - \\ & \gamma\langle -\widehat{\overline{\sigma}}_{max} \rangle) - \overline{\sigma}_c(\tilde{\varepsilon}_c^{pl}) \end{aligned} \tag{3.34}$$

$$\beta(\tilde{\boldsymbol{\varepsilon}}^{pl}) = \frac{\overline{\sigma}_c(\tilde{\varepsilon}_c^{pl})}{\overline{\sigma}_t(\tilde{\varepsilon}_t^{pl})}(1-\alpha^*) - (1+\alpha^*) \tag{3.35}$$

In equation 3.34, α^* and γ are dimensionless material constants, $\widehat{\overline{\sigma}}_{max}$ is the maximum principal stress, $\overline{q}$ is the Mises equivalent stress (defined in equation 3.37) and $\overline{p}$ is the hydrostatic pressure stress (defined in equation 3.36). The function $\beta(\tilde{\boldsymbol{\varepsilon}}^{pl})$ is expressed in equation 3.35 where $\overline{\sigma}_c$ and $\overline{\sigma}_t$ are the effective compression and tension cohesion stress (Abaqus, 2014).

$$\overline{p} = -\frac{1}{3}\overline{\boldsymbol{\sigma}} : \boldsymbol{I} \tag{3.36}$$

$$\overline{q} = \sqrt{\frac{3}{2}\overline{\boldsymbol{S}} : \overline{\boldsymbol{S}}} \tag{3.37}$$

$$\overline{\boldsymbol{S}} = \overline{p}\boldsymbol{I} + \overline{\boldsymbol{\sigma}} \tag{3.38}$$

Variable $\boldsymbol{I}$ in equation 3.36 and 3.38 is the identity tensor and $\overline{\boldsymbol{S}}$ the deviatoric stress tensor. A schematic visualization of the yield surface in plane stress is depicted in figure 3.5. Here different loading scenarios and the difference between tension and compression can be seen.

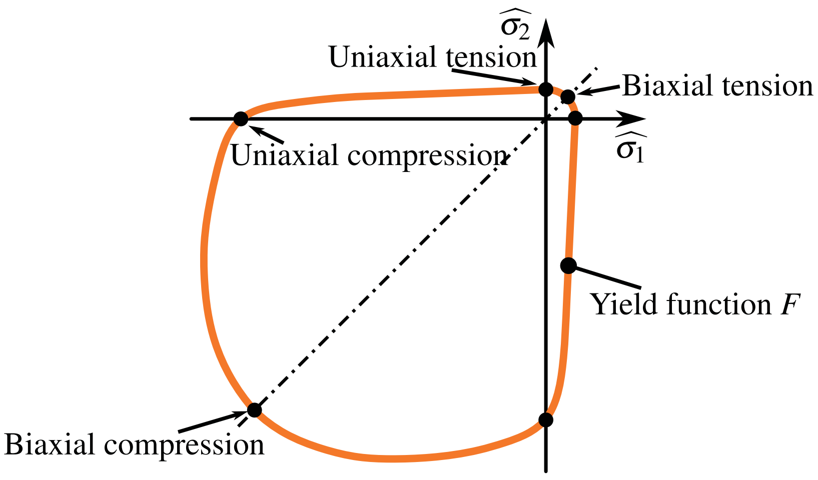


Fig. 3.5 Biaxial yield surface in plane stress (Abaqus, 2014).

Plastic flow occurs if the yield condition (equation 3.39) as well as the consistency condition is satisfied (equation 3.40). If F is less than zero or F is zero but $\dot{F}$ is less than zero, only elastic behavior of the material is observed (Abaqus, 2014; Lemaître and Chaboche, 1990).

$$F(\overline{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\varepsilon}}^{pl}) = 0 \tag{3.39}$$

$$\dot{F}(\overline{\boldsymbol{\sigma}}, \tilde{\boldsymbol{\varepsilon}}^{pl}) = 0 \tag{3.40}$$

In addition to the two conditions mentioned above (equations 3.39 and 3.40), a flow rule has to be defined in order to assess the amount and direction of inelastic strains. Plastic flow in the CDP model is determined by a nonassociated flow rule (equation 3.41) (Abaqus, 2014; Lemaître and Chaboche, 1990).

$$\dot{\boldsymbol{\varepsilon}}^{pl} = \dot{\lambda} \frac{\partial G(\overline{\boldsymbol{\sigma}})}{\partial \overline{\boldsymbol{\sigma}}} \tag{3.41}$$

In equation 3.41, $\dot{\lambda}$ defines the plastic multiplier and $\partial G / \partial \overline{\boldsymbol{\sigma}}$ the direction of the plastic strain vector. The flow potential function G is defined according to a hyperbolic *Drucker-Prager* function (equation 3.42) (Abaqus, 2014).

$$G = \sqrt{(\epsilon^* \sigma_{t0} \tan\psi)^2 + \overline{q}^2} - \overline{p} \tan\psi \tag{3.42}$$

Variable σ_{t0} in equation 3.42 is the uniaxial tensile stress at failure and ϵ^* is a parameter referred to as the eccentricity of the function. Moreover, ψ defines the dilation angle which is measured in the $\overline{p}$-$\overline{q}$ plane and determines the volumetric change due to plastic deformation (cf. figure 3.6). Unlike an associated flow rule the plastic strain increment is not normal to the yield function but to the plastic flow potential G (cf. figure 3.6) (Abaqus, 2014; Lemaître and Chaboche, 1990).

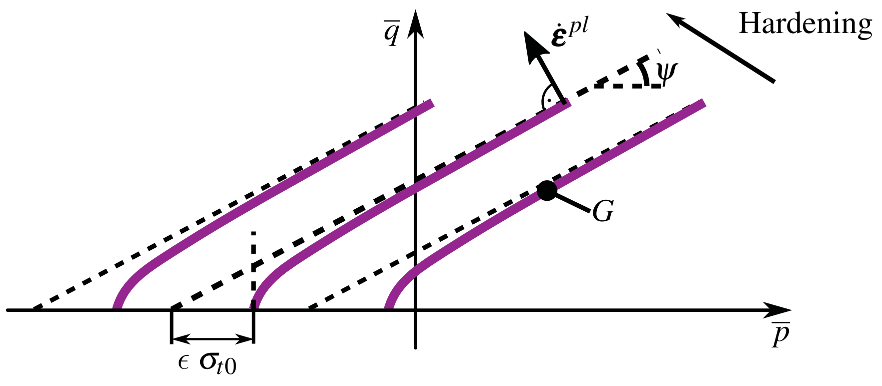


Fig. 3.6 Family of *Drucker-Prager* hyperbolic flow potentials in the $\overline{p}$-$\overline{q}$ plane (Abaqus, 2014).

3.2.2.3 Numerical issues

Since the CDP material model exhibits a softening behavior and stiffness degradation, severe convergence issues during an implicit analysis can occur. The CDP model in *Abaqus* offers the possibility to add viscoplastic regularization. This viscoplastic regularization causes the consistent tangent stiffness of the constitutive model to become positive for sufficiently small time increments. Therefore, a generalization of the Duvaut-Lions regularization is used. The viscoplastic strain rate tensor $\dot{\boldsymbol{\varepsilon}}_v^{pl}$ is defined according to equation 3.43 (Abaqus, 2014).

$$\dot{\boldsymbol{\varepsilon}}_v^{pl} = \frac{1}{\mu^*}\left(\boldsymbol{\varepsilon}^{pl} - \boldsymbol{\varepsilon}_v^{pl}\right) \tag{3.43}$$

In equation 3.43, μ^* represents the viscosity parameter which corresponds to the relaxation time of the viscoplastic system and $\boldsymbol{\varepsilon}^{pl}$ is the plastic strain in the inviscid model. Moreover, the scalar degradation variable d in equation 3.28 is substituted by the viscous stiffness degradation d_v. Its evolution is described in equation 3.44 (Abaqus, 2014).

$$\dot{d}_v = \frac{1}{\mu^*}(d - d_v) \tag{3.44}$$

The solution of the viscoplastic system relaxes to the inviscid case when $t/\mu^* \rightarrow \infty$. Therefore, small μ^* values usually improve the rate of convergence.

3.2.2.4 Dilation angle

The volumetric change during plastic deformation is characterized by the dilation angle ψ. In the CDP model, implemented in the FE program *Abaqus*, a dilation angle ψ at high confining pressure has to be defined (Abaqus, 2014). By default ψ is constant over the plastic deformation and can therefore cause unrealistic swelling. Triaxial experimental strength investigations show that the dilation angle drops rapidly with increasing plastic flow (cf. for example Arzúa and Alejano (2013)). In order to consider this behavior a user subroutine has been written in *FORTRAN 77* to extend the CDP model in *Abaqus* (Abaqus, 2014), see the implementation according to Arzúa and Alejano (2013), equation 3.45.

$$\psi(\gamma^{pl}) = \begin{cases} \psi_{max} & if\ \gamma^{pl} \leq \gamma_{tr}^{pl} \\ \frac{a^* b^* \left(e^{-b^* \gamma^{pl}} - e^{-c^* \gamma^{pl}}\right)}{c^* - b^*} & else \end{cases} \tag{3.45}$$

In equation 3.45 a^*, b^*, c^* are material parameters which can be found in Arzúa and Alejano (2013) and γ^{pl} is the maximum plastic shear strain expressed in percent. The original expression postulated by Arzúa and Alejano (2013) (else branch in equation 3.45) considers an initially steep increase in dilation followed by an exponential decrease if a certain threshold γ_{tr}^{pl} is reached. For numerical reasons this first sharp increase is substituted by a constant ψ_{max} followed by the proposed decrease (cf. equation 3.45). The details of the implementation in *Abaqus* and the used material parameters can be found in chapter 4.

Chapter 4

Numerical models

Various numerical studies have been performed during this thesis in order to identify the mechanisms which governs the microwave induced stresses and damage. Moreover, numerous parameter studies have been conducted to identify optimum irradiation conditions. Therefore, 2D and mainly 3D simulations are performed on an inhomogeneous rock model as well as 3D simulations on a homogeneous rock, the latter considering the strong electromagnetic-thermal field coupling. In all presented simulations a microwave beam with a typical technical frequency of 2.45 GHz as the microwave source is applied. The shape of the beam at the source position is Gaussian (Kogelnik and Li, 1966) with a waist radius w_0 of 4.3 cm which corresponds to an opening of a waveguide at 2.45 GHz. The function of the Gaussain beam at the source plane is defined by equation 4.1 (Meschede, 2008).

$$E\left(r\right) = E_0 \; e^{\frac{-r^2}{w_0^2}} \tag{4.1}$$

In equation 4.1 E is any electric field component at the source plane, E_0 the value at the Gaussian axis and r is the radial coordinate. The source is positioned 1 cm in front of the material. The specific characteristics of the investigated models are described in the following sections.

4.1 Inhomogeneous models

By using models where the microstructure of the hard rock is resolved (inhomogeneous model), the microwave induced stresses and damages in rather inhomogeneous rocks such as granite can be investigated. Moreover, the influences of grain properties such as varying microwave absorption behavior, different thermal and mechanical properties, anisotropic behavior as well as phase transformation are considered. Therefore, the stress field of the inhomogeneous microstructure models is compared to a model with homogeneous material definition. This allows to draw conclusions on the influence of the microstructural details.

4.1.1 Methodology

In order to perform the simulations on inhomogeneous hard rocks, a comprehensive simulation methodology connecting different simulation modules is required. This thesis presents a simulation chain for the inhomogeneous models without taking any feedback of temperature changes on the electromagnetic properties into account. Furthermore, there is no influence of the displacements on the thermal field (i.e. a weak coupling) (figure 4.1).

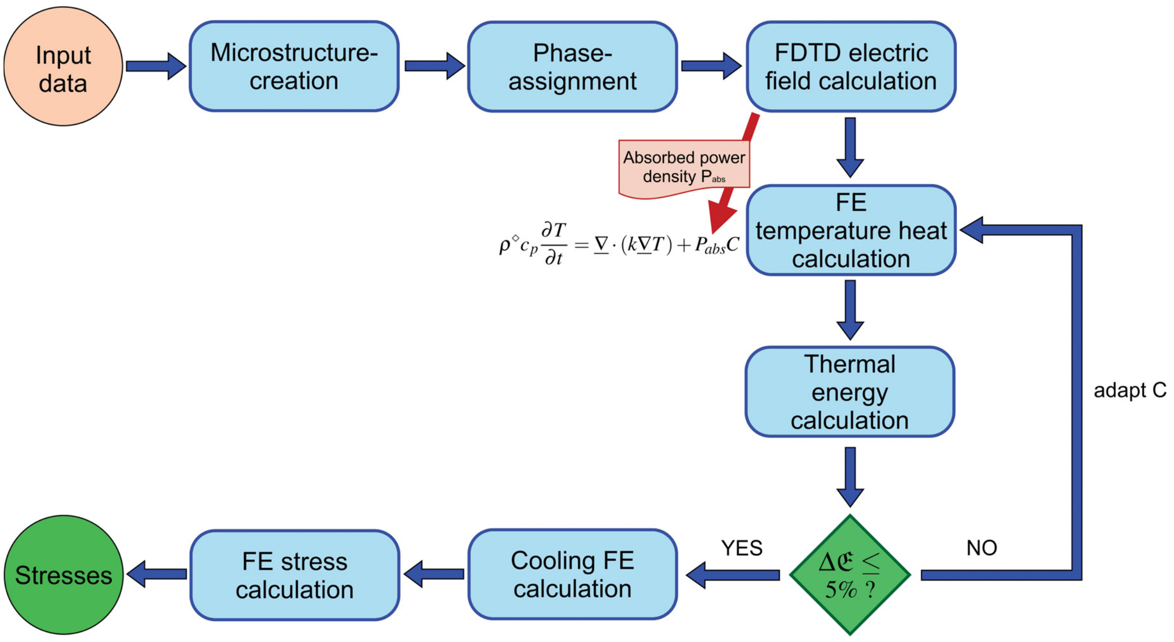

Input data
Microstructure-creation
Phase-assignment
FDTD electric field calculation
Absorbed power density P_abs
$\rho^{\diamond} c_p \frac{\partial T}{\partial t} = \underline{\nabla} \cdot (k \underline{\nabla} T) + P_{abs} C$
FE temperature heat calculation
Thermal energy calculation
adapt C
$\Delta\mathfrak{E} \leq 5\%$?
YES
NO
Cooling FE calculation
FE stress calculation
Stresses

Fig. 4.1 Simulation chain, blue arrows indicate sequential working paths and red arrow file transfer (Toifl et al., 2016a).

The simulation process starts with the collection and preprocessing of input data such as physical and thermo-mechanical material properties, numerical parameters, microstructure parameters and model dimensions (cf. figure 4.1). In the first numerical working package the microstructure is generated. The specific properties of the 2D and 3D microstructures are described in the respective chapters. Afterwards, the various grains are assigned to different phases according to a given random distribution function. The generated artificial microstructure represents the basic input for all further simulation modules. Within the next step the electromagnetic field inside the model rock is calculated by solving Maxwell's equations numerically applying a FDTD analysis as presented in section 3.1. The time averaged squared electric field $\overline{E^2}$ (or E_y^2 in the 2D case) is determined, which is used to derive the absorbed power density P_{abs} distribution as the output of this module (cf. equation 2.17). In the subsequent thermal FE calculation performed with the program *Abaqus* (Abaqus, 2014), the absorbed power density P_{abs} multiplied by a constant factor C is directly treated as heat source s entering the heat conduction equation (cf. equation 3.22) which is solved numerically (figure 4.1). This body heat flux is applied in each increment of the thermal analysis at the respective integration point by a *DFLUX* subroutine written in *FORTRAN 77* (Abaqus, 2014). This way, various microwave power levels as well as microwave irradiation times are considered. After the numerical evaluation of the transient temperature field during heating, the thermal energy inside the model $\mathfrak{E}_{th}$ is calculated by equation 4.2 and compared to the provided microwave energy minus 30% loss which is estimated with the currently used lab-setup. These losses account for the reflections back to the magnetron and the imperfect formation of the Gaussian beam. If the evaluated difference is greater than 5%, the constant factor C is adapted and the thermal heat transfer calculation is repeated (figure 4.1).

$$\mathfrak{E}_{th} = \sum_{inc=0}^{inc_{end}} \sum_{i=0}^{w} \rho_i^{\diamond inc}\, c_{p,i}^{inc}\, IVOL_i^{inc}\, \Delta T_i^{inc} + \sum_{inc=0}^{inc_{end}} \sum_{j=0}^{k} HFL_j^{inc}\, \Delta t_j^{inc}\, A_j^{inc} \tag{4.2}$$

In equation 4.2 variable *inc* is the current increment of the FE heat transfer calculation, *i* the integration point, $IVOL$ the integration point volume, ΔT the temperature difference to the preceding increment, HFL the heat flux component over the front surface (e.g. figure 4.5), Δt the time increment and A the surface which corresponds to the respective integration point on the front surface. The second double summation only affects all integration points on the front surface k.

Afterwards, the natural cooling of the hard rock model is calculated for a duration of 3600 s by switching off the body heat flux. The transient inhomogeneous temperature field is used as an input for the FE stress simulations. First, linear elastic calculations are performed followed by damage calculations using the CDP material model (cf. section 3.2.2). Finally, the stress and damage formation is analyzed in a statistical manner.

4.1.2 2D model

First, simulations of the microwave induced stresses and damage are performed on a two component artificial 2D microstructure. The aim of these 2D studies is to assess the influence of the heterogeneity on the propagation of a microwave beam, i.e. the amount of diffuse scattering as well as the heating and the formation of microwave induced stresses. The results have been published in Meisels et al. (2015) and Toifl et al. (2014). The preliminary 2D FDTD calculations are performed by Prof. Ronald Meisels from the Institute of Physics, Montanuniversitaet Leoben.

4.1.2.1 Microstructure

For the 2D simulations a two component rock model with circular microwave absorbing discs distributed randomly in a transparent matrix is built (figures 4.2 and 4.3). The geometrical configuration is generated by means of a user code written in *C* following the procedure described by Meisels and Kuchar (2007) where the random distribution is realized by the positional disorder of the discs on a square grid.

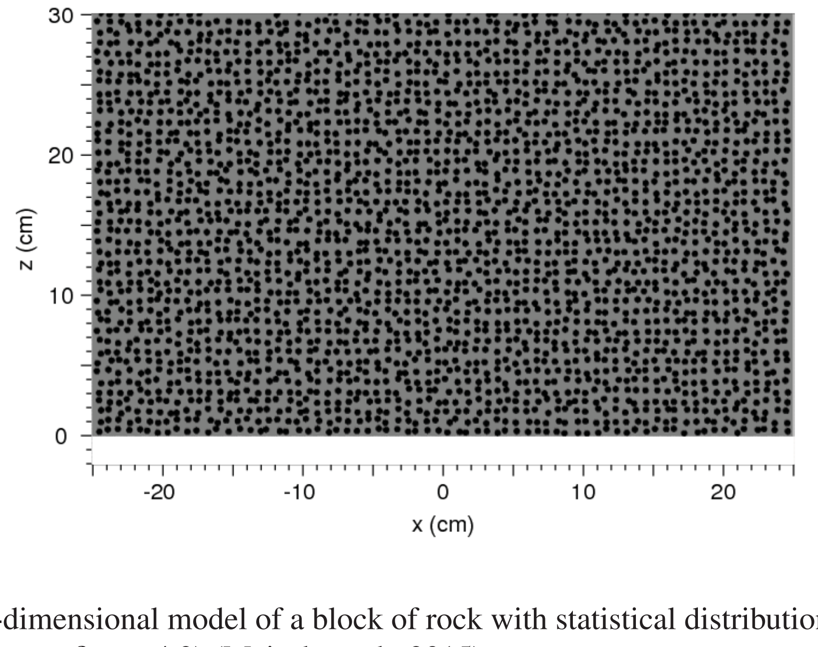


Fig. 4.2 Two-dimensional model of a block of rock with statistical distribution of discs in a matrix (details see figure 4.3) (Meisels et al., 2015).

The degree of the positional disorder is determined by the parameter δ. The deviation of the *i*-th element from the ideal position on a square lattice in the x- and z-direction, $\delta_{x,i}$ and $\delta_{z,i}$, is limited by δ_{max} ($\delta_{x,i}$, $\delta_{z,i} < \delta_{max}$) and has a uniform distribution (figure 4.3).

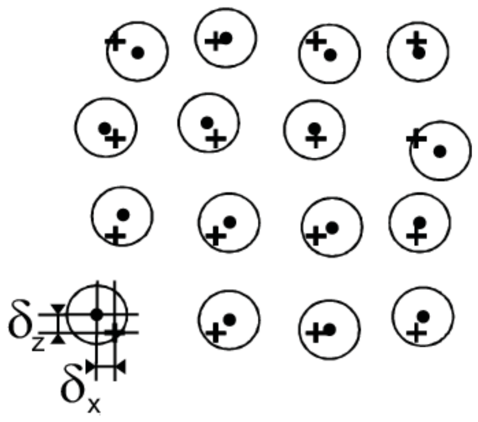


Fig. 4.3 Enlarged section of the two-dimensional model rock. The crosses represent the lattice points, the dots the actual centers of the discs. δ_x and δ_z denote the deviation from the ideal position on a lattice point (Meisels et al., 2015).

In the calculations δ is chosen to be 25% of the lattice constant a = 0.71 cm. The diameter of the discs is 0.467 cm corresponding to a filling factor f of 0.34. The variable f is the fraction of the area occupied by the discs relative to the total area. These values (parameters summarized in table 4.1) indicate that the arrangement of the discs is far away from percolation but that some of the discs can overlap by about 0.1 cm. This model is simplified but shows a qualitatively good estimate of the effects of scattering due to differences in the dielectric properties of inclusions and matrix in a rock.

Table 4.1 Parameters of the 2D model rock (Meisels et al., 2015).

Lattice constant a	0.71 cm
Disc diameter	0.467 cm
Filling factor f	0.34
Disorder, max δ_{max}	25% of lattice constant a
Size of 2D model rock	50×30 cm^2

4.1.2.2 Material data

In the 2D two component model it is assumed that the discs represent the microwave absorbing phase and the matrix is transparent for the microwaves. Based on these assumptions an averaged relative permittivity ϵ_r of a hard rock was taken from literature (cf. table 2.2). For the 2D study a value for $\epsilon_{r,eff}$ of $7.4+0.88i$ is chosen. The individual permittivities of the two constituents are determined by the Bruggeman's effective medium theory (Bruggeman, 1935). For the two component 2D model rock the individual permittivities as well as the effective one $\epsilon_{r,eff}$ have to fulfill equation 4.3 (Bruggeman, 1935).

$$f\frac{\epsilon_{r,d}-\epsilon_{r,eff}}{\epsilon_{r,d}+\epsilon_{r,eff}}+(1-f)\frac{\epsilon_{r,ma}-\epsilon_{r,eff}}{\epsilon_{r,ma}+\epsilon_{r,eff}}=0 \tag{4.3}$$

In equation 4.3 the index d corresponds to the discs, ma to the matrix and f is the filling factor defined in section 4.1.2. The imaginary part of the matrix permittivity is set to zero ($\epsilon''_{r,ma}=0$) corresponding to the assumption that this phase is transparent for microwaves. In order to increase the reflections at the interfaces between matrix and discs, a large difference between the real part of the matrix ($\epsilon'_{r,ma}$) and the discs ($\epsilon'_{r,d}$) is chosen. Hence, $\epsilon'_{r,ma}$ is set to 7.1 and the remaining complex part $\epsilon_{r,d}$ is derived by equation 4.3. As a result $\epsilon_{r,d}=7.690+2.787i$ is obtained. The resulting permittivity of the discs represents an upper limit case.

The temperature-dependent thermo-mechanical material properties (density, heat capacity, thermal conductivity, thermal expansion, Young's modulus, Poisson's ratio) of the two components (disc and matrix) were taken from literature. Therefore, it is assumed that the discs have the properties of plagioclase ($Ab_{56}An_{44}$, where Ab is albite and An anorthite) and the matrix those of quartz being constituents of hard rocks such as granite. These assumptions correlate with the microwave absorption behavior since quartz is a very poor absorbing mineral and plagioclase a rather good absorber (cf. table 2.2). The thermo-mechanical material properties used for the 2D simulations are summarized in table 4.2 (Meisels et al., 2015).

Table 4.2 Temperature-dependent thermo-mechanical properties of quartz and plagioclase (Meisels et al., 2015).

Component	**Property**	**T [°C]**	**Value**	**Source**
Quartz	c_p [J/kg K]	0	698	Goranson (1942)
		200	970	
		400	1130	
		800	1170	
		1200	1327	
	k [W/m K]	0	9.125	Birch and Clark (1940)
		50	7.515	
		100	6.445	
		150	5.735	
		200	5.190	
		250	4.710	
		300	4.335	
		350	4.020	
	α [1/K]	25	8.1×10^{-6}	Ackermann and Sorrell (1974)
		573	8.1×10^{-6}	
		574	0	
	E^* [GPa]	-	86.9	Atanasoff and Hart (1941)
	ν []	-	0.17	
	$\rho^\diamond$ [kg/m³]	-	2648	
Plagioclase $Ab_{56}An_{44}$	c_p [J/kg K]	20	750	Cermak and Rybach (1982)
		50	800	Benisek et al. (2013)
		200	945	
		400	1054	
		500	1090	
	k [$W/m\,K$]	-	1.46	Horai and Baldridge (1972)
	α [1/K]	0	3.5×10^{-6}	Skinner (1966)
		1000	3.5×10^{-6}	
	E^* [GPa]	-	53.3	Gebrande et al. (1982)
	ν []	-	0.29	
	$\rho^\diamond$ [kg/m³]	-	2703	

For the 2D CDP model averaged material properties of bulk hard rocks were retrieved from literature. These values are summarized in table 4.3. The viscosity parameter μ^* was found in a parameter study by evaluating the convergence rate and the amount of energy used for regularization (cf. section 3.2.2.3).

Table 4.3 Concrete damaged plasticity material parameters for the 2D model (Toifl et al., 2014).

Description	**Symbol**	**Value**	**Source**
Fracture toughness	K_{IC} [MPa$\sqrt{\text{m}}$]	1.48	Nasseri et al. (2005)
Tensile strength	σ_{t0} [MPa]	9	Hustrulid et al. (2001)
Compressive strength	σ_{c0} [MPa]	250	Hustrulid et al. (2001)
Dilation angle	ψ [°]	15	Busetti et al. (2012b)
Eccentricity	ϵ^* []	0.1	Busetti et al. (2012b)
σ_{b0}/σ_{c0}	σ_{b0}/σ_{c0} []	1.16	Busetti et al. (2012b)
K_c	K_c []	0.66	Busetti et al. (2012b)
Viscosity parameter	μ^* []	1×10^{-2}	

Since the fracture toughness K_{IC} is defined in the CDP model, a linear behavior between the stress in tension (σ_t) and displacement (u_t) during the degradation is assumed (cf. figure 4.4). For the interpretation of the remaining CDP material parameters the reader is referred to chapter 23.6.3 of the *Abaqus Analysis User's Guide* (Abaqus, 2014).

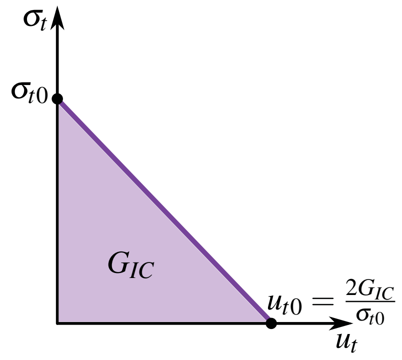


Fig. 4.4 Tension damage behavior in CDP model (Abaqus, 2014).

4.1.2.3 FDTD model

The FDTD simulations for the 2D case are performed with the commercial program *RSoft* (RSoft, 2014) since no MPI functionality is required. The size of the rectangular model rock is 50 cm in width and 30 cm in direction of microwave propagation, the size of the computational domain is 50 × 32 cm^2 as shown by the black frame in figure 4.2. The computational grid has a lattice constant of 0.05 cm. To find out whether this grid is sufficiently fine, a calculation was performed with the half of the value (0.025 cm). The results for E_y^2 along x = 0 cm deviate by less than 0.1% between the 0.05 cm case and 0.025 cm. Therefore, a value of 0.05 cm is considered sufficiently small for obtaining reliable results. The microwave source is positioned 1 cm in front of the sample (figure 4.2, x = 0, z = -1 cm). It emits a beam in positive z-direction with the polarization in y-direction and the value of the time averaged E_y^2 being = 1 (at x = 0, in dimensionless units). Part of the beam is reflected at the air / rock interface and at the discs / matrix interfaces in the interior which interferes with the emitted beam. At the boundary of the computational domain the radiation is entirely absorbed. This ensures that the wave pattern remains unaffected by reflections from these boundaries and that the effect of the disorder can clearly be observed. The model with absorbing boundary conditions is also relevant for a real environment, e.g. excavation of a mine or tunneling operation, because there the thickness would be almost infinite compared to the penetration depth.

4.1.2.4 FEM model

For the FE thermo-mechanical stress calculations a regular mesh containing four node bilinear elements is generated. To model the material inhomogeneities in the vicinity of the microwave irradiation spot in an appropriate manner, a finer discretization (node distance = 0.1 mm instead of 0.5 mm) is used in this region (cf. figure 4.5). The two types of meshes are joined together by means of tie constraints.

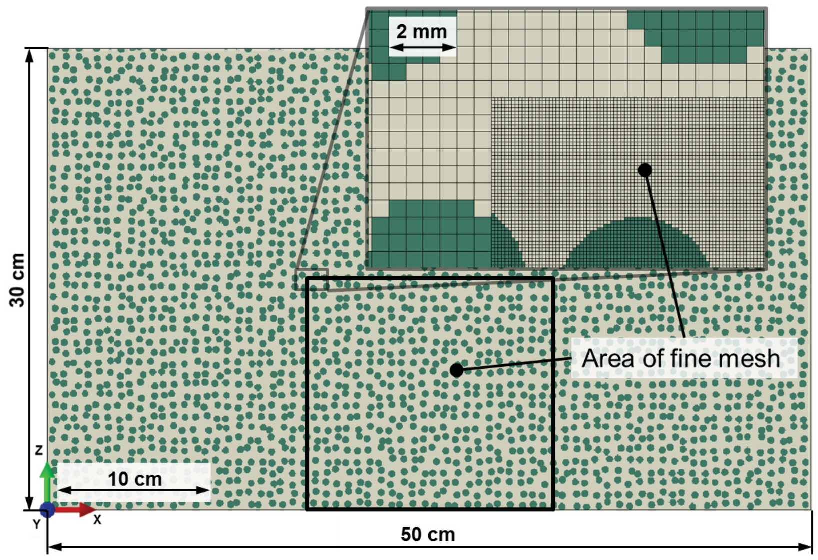


Fig. 4.5 2D finite element model including microstructure (Toifl et al., 2014).

On the front face (normal vector points in negative z-direction, see figure 4.5) in the thermal FE model, a thermal conductance of 20 $^{W}/_{m^2K}$ and an emissivity of 0.8 combined with an ambient temperature of 25°C is assumed. At all other faces an emissivity of 0.8 is used. The initial temperature of all nodes is set to 25°C. The inhomogenous rock sample is irradiated with a microwave source of P_{mw} = 25 kW. The total thermal energy within the model $\mathfrak{E}_{th}$ is compared to the total provided energy $\mathfrak{E}_{prov}$ (cf. equation 4.4) minus 30% accounting for losses.

$$\mathfrak{E}_{prov} = t\,\frac{P_{mw}}{w_0^2\pi}\,2w_0\;1 \tag{4.4}$$

In equation 4.4 the total energy $\mathfrak{E}_{prov}$ provided in the 2D model is calculated by dividing the total microwave power P_{mw} by the surface of the Gaussian beam at the source and scaling it to the front surface of the 2D model (in *Abaqus* an internal thickness of 1 is used for the numerical integration of the 2D elements).

In a subsequent mechanical FE model the same mesh again using linear elements with reduced integration as in the thermal analysis is employed. Furthermore, the same two steps (heating and cooling) are analyzed. The time varying temperature field is applied as an input to the stress model. Moreover, plane strain conditions are assumed. Finally, damage is assessed by means of a concrete damaged plasticity material model.

4.1.3 3D model

In the 3D case a more realistic microstructure containing polyhedral grains is used. Moreover, the anisotropic behavior of quartz grains as well as their phase transformation is considered. In order to extend the 2D model and to assess the micromechanical behavior of microwave induced stresses in greater detail, a two component 3D model is investigated. The results of this model have been published in Toifl et al. (2015a,b, 2016b). Moreover, a three component model is also used where a real granite rock is analyzed. To this end, measured material parameters (cf. Hartlieb et al. (2016)) are used and a parameter study with various irradiation times as well as microwave power levels is performed. These results are presented in Toifl et al. (2016a).

4.1.3.1 Model definition

Microstructure cube

For all 3D inhomogeneous models the same cube containing an artificial microstructure is used. This microstructure has been created by a Voronoi tessellation algorithm with randomly positioned seed points. This produces statistically distributed grain sizes. The phases are assigned according to the two or three component model. This approach was chosen in order to be able to capture the nature of the sharp grain boundaries. First, a cube is partitioned into polyhedra representing the grains of the material by a Voronoi tessellation algorithm provided by the open source software *Neper* (Quey et al., 2011). After that, the software performs some optimization loops (3 in the current case) in order to remove short edges of the polyhedra which would cause meshing problems. Subsequently, the grains are meshed with linear tetrahedral finite elements. At this point it is crucial that the desired number of elements per polyhedron is set to values above 100, otherwise the meshing algorithm becomes highly constrained and would result in a low quality of the FE mesh. Unfortunately, the meshing process of the polyhedra is very time consuming and the time effort strongly increases with the number of polyhedra. A microstructure of 30000 grains is assumed which leads to manageable tessellation / meshing times and number of finite elements (4060685), see figure 4.6.

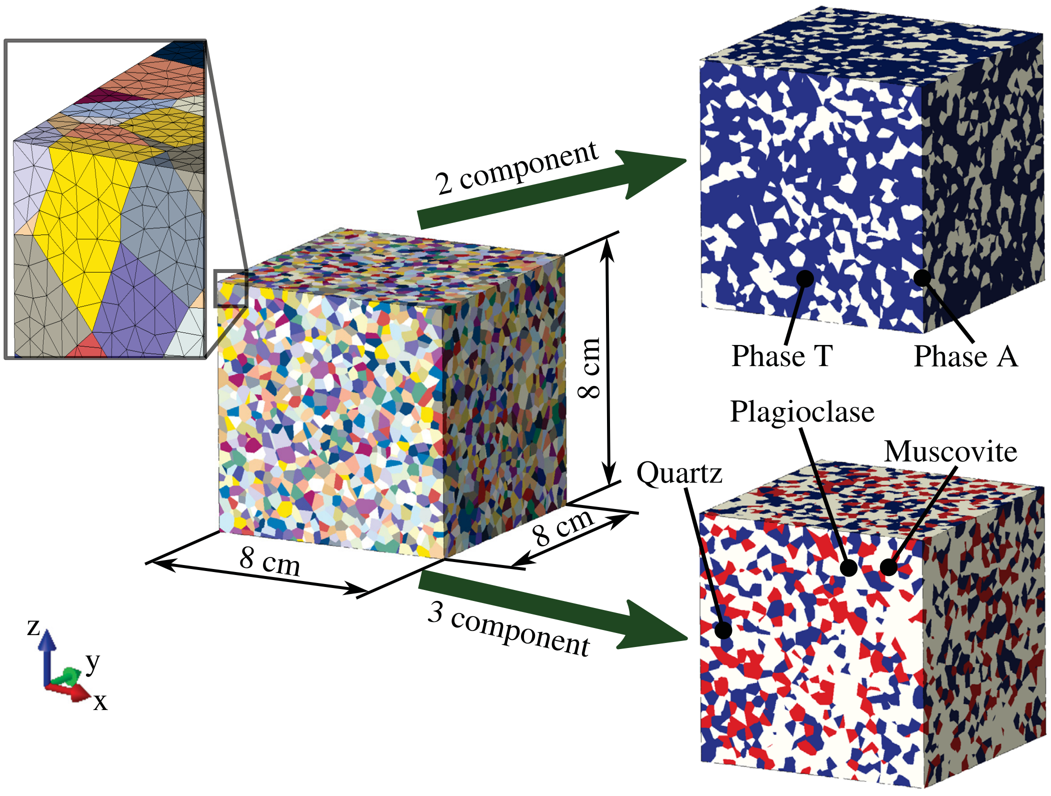


Fig. 4.6 Microstructure model including various grains and assigned phases in the two (phase T is blue and phase A beige) and three component model (quartz is blue, plagioclase beige and muscovite red). Upper left close-up shows the FE mesh (Toifl et al., 2016a,b).

In the next step the material phases are assigned to the different grains of the microstructure by a *C*++ script (figure 4.6). This is done in the two as well as three component case by considering the desired volume fractions.

Two component model

The 3D two component model is an extension of the 2D inhomogeneous model. Therefore, the same constant filling factor f of 0.34 (volume of microwave absorbing phase A in reference to total volume) is assumed. The two components are phase A (beige in figure 4.6) for the microwave absorbing component and phase T (blue in figure 4.6) for the transparent part. In order to assess different morphologies (different phase assignments) with the same filling factor f, two additional models with $f = 0.34$ have been generated. The new models (model B and C) are compared to the reference model (model A) in figure 4.7.

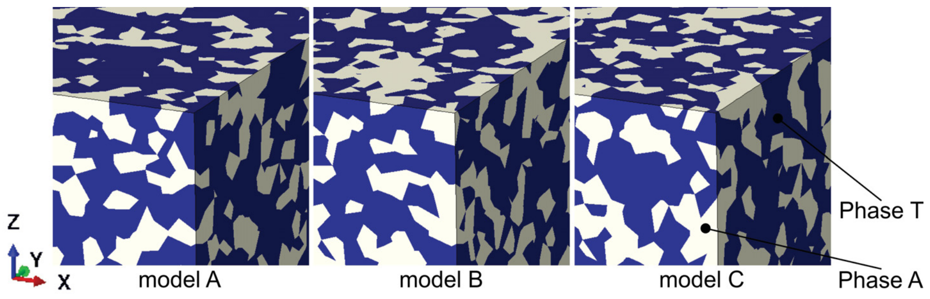


Fig. 4.7 Different two component 3D models with same filling factor but varying morphologies (Toifl et al., 2016b).

Significant variations in the phase distribution are visualized in figure 4.7. In addition to models A - C further models have been generated for three different filling factors ($f = 0.2$, $f = 0.34$, $f = 0.4$).

Three component model

The three component model is built to resemble granite which is analyzed by thermo-mechanical as well as dielectric measurements (cf. section 4.1.3.2) and during microwave irradiation experiments. The investigated granite mainly contains quartz, feldspar and micas. For the three component model the respective measured volume fractions of the constituents are used. The investigated granite samples contain 27 vol.% quartz (blue in figure 4.6), 53 vol.% feldspar (mainly plagioclase; beige in figure 4.6) and 20 vol.% micas (mainly muscovite; red in figure 4.6). The resulting phase distribution is visualized in figure 4.6.

Model rock

For obvious reasons it is impossible to finely mesh the entire rock so one standard technique is to embed a finely mesh cube in a surrounding homogenized bulk material with averaged material properties (e.g. the two component model in figure 4.8). The scaling of the cube depends on the desired average grain diameter, which is set to 3.2 mm in this work. This leads to a cube with an edge length of 8 cm containing the highly resolved microstructure. Moreover, the cube's edge length also results from the fact that at a distance of about 8 cm the temperature field has sufficiently decayed that almost constant conditions can be assumed. From there on it is admissible to set the boundaries of the microstructure domain.

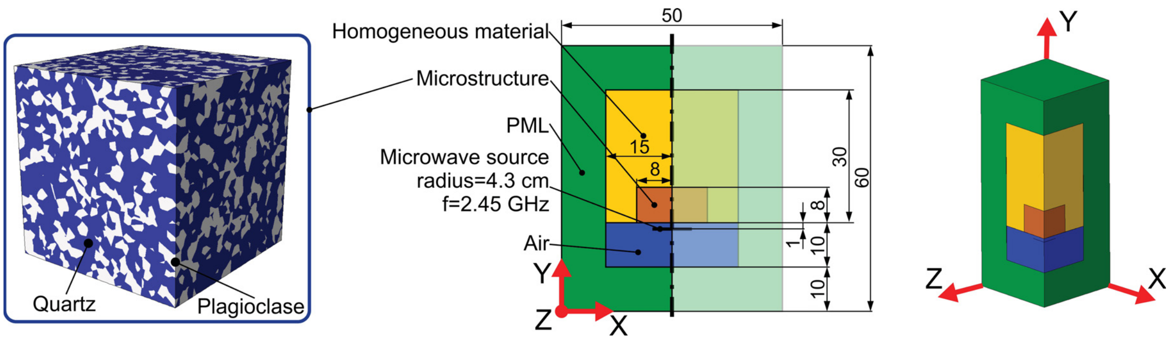


Fig. 4.8 Microstructure (left) and FDTD model (right, all dimensions in centimeters) of the two component case including isometric view. PML stands for perfectly matched layer and is used to be able to truncate the simulation model without causing reflections of the microwave (Toifl et al., 2016b).

A quarter symmetry of the whole numerical model is assumed in order to reduce the problem size and thus avoid excessive simulation times (figure 4.8). The two and three component models differ only in the dimensions of the material due to the higher penetration depth in the three component case (compare figure 4.8 and figure 4.24).

4.1.3.2 Material data

Two component model

In the two component model the same dielectric properties of the two phases and the same filling factor $f = 0.34$ as in the 2D case are chosen. The relative permittivity of the microwave transparent phase (phase T) is $\epsilon_{r,T} = 7.1 + 0i$ and that of the absorbing phase (phase A) $\epsilon_{r,A} = 7.690 + 2.787i$. Since Bruggeman's effective medium theory (Bruggeman, 1935) in the 3D case (equation 4.5) differs slightly from the 2D case, a different effective relative permittivity is obtained ($\epsilon_{r,eff} = 7.37 + 0.90i$).

$$f \frac{\epsilon_{r,A} - \epsilon_{r,eff}}{\epsilon_{r,A} + 2\epsilon_{r,eff}} + (1 - f) \frac{\epsilon_{r,T} - \epsilon_{r,eff}}{\epsilon_{r,T} + 2\epsilon_{r,eff}} = 0 \tag{4.5}$$

As in the 2D case the resulting permittivity of phase A represents an upper limit case. Therefore, the high imaginary part of $\epsilon_{r,A}$ is explained by a mixture of strong absorbing minerals (pyroxene and ilmenite) with plagioclase representing the absorbing phase A.

Similar to the 2D simulation the thermo-mechanical properties of the minerals were taken from literature. Therefore, the main constituent of phase A and of phase T, respectively, is chosen for the FE simulations. In this thesis, quartz represents the microwave transparent phase T and plagioclase, as the main component of phase A, the absorbing phase. These two minerals are typical constituents of hard rocks.

At typical pressure levels quartz transforms from trigonal quartz (α quartz) to hexagonal quartz (β quartz) at 573°C (Le Chatelier, 1889). The phase transition is accompanied by a change in symmetry and volume, where the β quartz has higher symmetry and volume than the α quartz (Moss, 1999). In order to consider the α to β phase transformation of quartz in the numerical model, material data in a range from room temperature up to about 800°C is required. Unlike in the 2D case, material data with more values near the phase transformation temperature as well as anisotropic material parameters are required. Conversely, the material response of plagioclase is assumed isotropic and the material data were taken from literature. The plagioclase material data for the 3D models are summarized in figures 4.14 - 4.17.

The elastic data as well as heat capacity and thermal expansion of quartz are taken from Carpenter et al. (1998). Their measurements of the lattice parameters form the basis for the calculation of the transformation strains. Experimental data are compared in Carpenter et al. (1998) with theoretical curves obtained from a Landau-type modeling of the temperature dependence of strains and elastic moduli (Pitteri, 2015). The anisotropic thermal conductance is used from Gibert and Mainprice (2009) where the thermal diffusivity of single quartz crystals is measured by a modified Ångström method up to a temperature of 800°C.

Specific heat capacity and thermal conductance of quartz

The thermal material behavior of quartz is depicted in figures 4.9 and 4.10 in a range from room temperature up to 800°C. The vertical line indicates the phase transformation temperature from α quartz to β quartz.

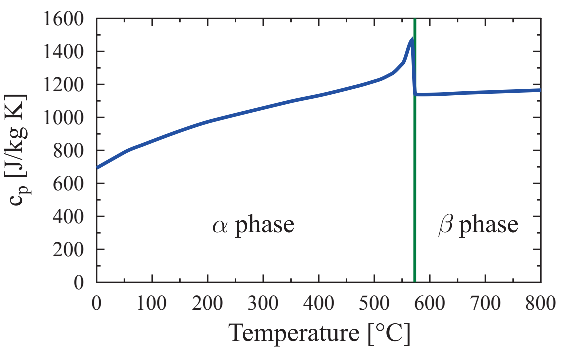


Fig. 4.9 Specific heat capacity c_p [$^{J}/_{kg\,K}$] of quartz as a function of temperature [°C]. Vertical line indicates phase transformation (Carpenter et al., 1998; Toifl et al., 2016b).

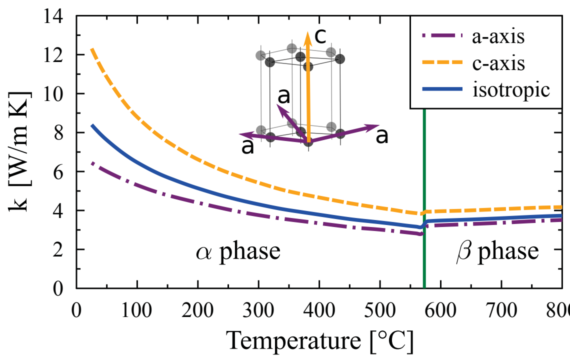


Fig. 4.10 Thermal conductance k [$^{W}/_{m\,K}$] of quartz in the isotropic and anisotropic case (Gibert and Mainprice, 2009; Okrusch and Matthes, 2005; Toifl et al., 2016b).

The heat capacity c_p is a scalar quantity and thus is represented by only one line in figure 4.9. The hotter the quartz gets, the more thermal energy is needed to further enhance the temperature. At the phase transformation significantly more energy (due to the latent heat) is needed to change the crystallographic system (peak in figure 4.9). In figure 4.10 the anisotropic behavior of the thermal conductance k as well as its isotropic equivalent mean are visualized. Only a slight change at the phase transformation temperature is noted. However, significant differences of the thermal conductance along the a- and c-axis, especially for the α quartz, are observed.

Uniaxial thermal expansion of quartz

Based on the thermal strains reported in Carpenter et al. (1998) the thermal expansion coefficient α with respect to a room temperature of 25°C is derived (figure 4.11). Additionally, the anisotropic thermal elongation of the quartz grains is considered.

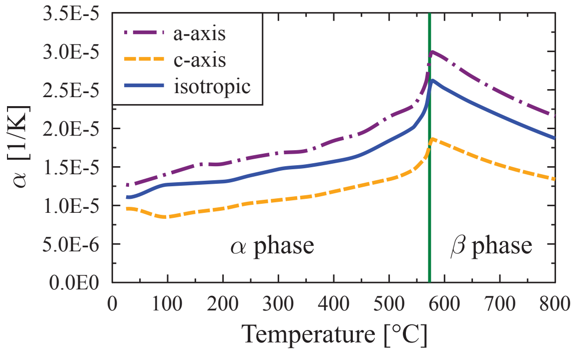


Fig. 4.11 Thermal expansion coefficient α [1/K] of quartz for the isotropic and anisotropic case (Carpenter et al., 1998; Toifl et al., 2016b).

In figure 4.11 it is observed that the thermal expansion increases rapidly when the temperature is close to the phase transformation temperature. This is due to the jump of the density from the α to the β phase. In the β quartz the thermal expansion coefficient α decreases since the thermal strains are almost constant in this phase. Furthermore, a strong difference between the thermal expansion along the a-axis of the crystal and the c-axis is observed. This gives rise to the conjecture that strong anisotropic effects should be seen within the stress field of the anisotropic model.

Elastic constants of quartz

Based on symmetry considerations of the crystal structure in quartz seven elastic constants have to be defined in order to fully describe the elastic behavior. Actually, in the β phase only six constants are required since C_{14} is zero in the hexagonal crystal configuration (figure 4.12), see Carpenter et al. (1998).

Figure 4.12 shows the strong temperature dependence of the different elastic constants, especially near the α - β phase transformation. Furthermore, the crystal orientation suddenly changes at the phase transformation (e.g. C_{14} becomes zero in β quartz and C_{11} the dominant coefficient). The isotropic material constants (Young's modulus E^* and Poisson number ν) are obtained analytically by means of a Voigt–Reuss–Hill averaging for the trigonal as well as hexagonal symmetry (Peselnick and Meister, 1965).

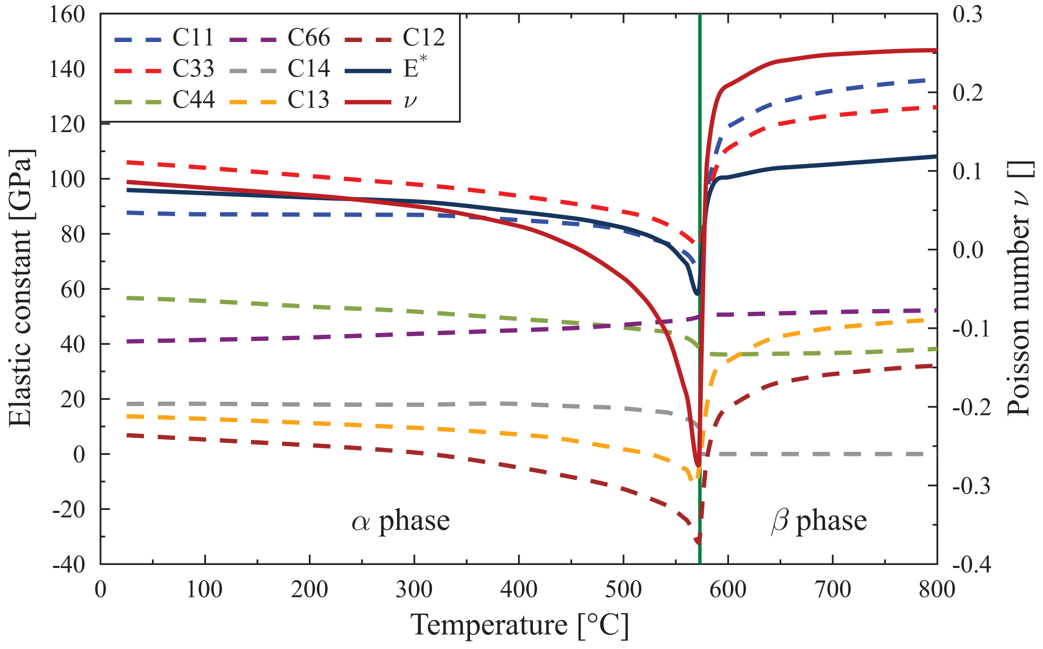


Fig. 4.12 Elastic constants C_{ij} [GPa] of quartz as a function of temperature [°C] (Carpenter et al., 1998). Isotropic elastic data is derived by a Voigt–Reuss–Hill averaging (Peselnick and Meister, 1965; Toifl et al., 2016b)

Three component model

As part of the FWF project various thermo-mechanical as well as dielectric measurements on granite samples (and other hard rocks) were conducted. These experiments were performed by Dr. Kaschnitz of the Austrian Foundry Research Institute (ÖGI, thermal conductivity, heat capacity and thermal expansion), Dipl.-Ing. Überbacher of the Seibersdorf Laboratories, Austria and Prof. Hutcheon of Microwave Properties North, Deep River, Ontario, Canada (permittivity measurements). They are summarized in Hartlieb et al. (2016). Details about the conducted measurements of the bulk materials can be found in Hartlieb et al. (2016). The properties of the single minerals were taken from literature or adapted according to a material averaging (Beardsmore and Cull, 2001; Bruggeman, 1935; Peselnick and Meister, 1965; Roy et al., 1981).

The investigated granite samples contain 27 vol.% quartz, 53 vol.% feldspar (mainly plagioclase) and 20 vol.% micas (mainly muscovite). Moreover, a maximum grain size between 2 and 3 mm is determined in the samples. For the conducted simulations plagioclase represents the feldspar group and muscovite the mica group.

Data of the complex dielectric constant of the bulk granite block were taken from measurements at a microwave frequency of 2.45 GHz as a function of temperature in a range between 20°C and 1000°C in steps of 100°C. Figure 4.13 visualizes the mean values of the real ($\epsilon'_{r,g}$) and imaginary ($\epsilon''_{r,g}$) parts.

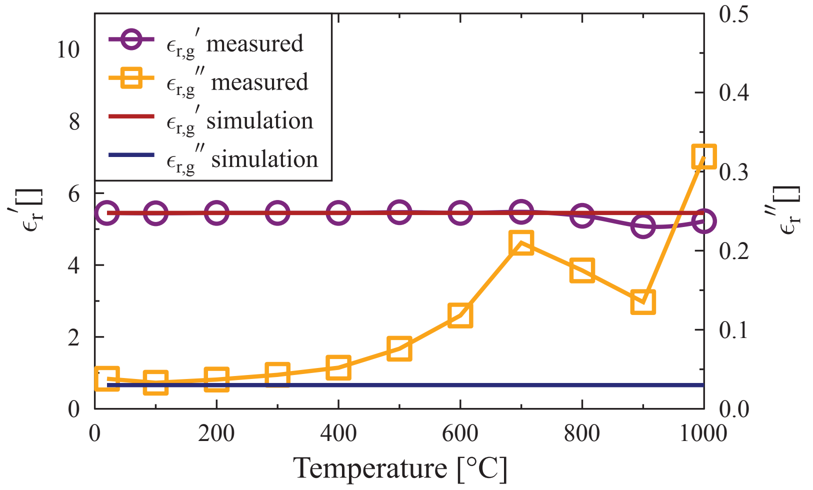


Fig. 4.13 Complex permittivity of granite (Hartlieb et al., 2016; Toifl et al., 2016a).

The real part of the permittivity of the bulk granite is almost independent of the temperature as observed in figure 4.13. In contrast the imaginary part increases significantly after 300°C. Since the presented inhomogeneous simulation strategy requires constant values of the complex dielectric property a lower limit case of $\epsilon_{r,g} = 5.45 + 0.03i$ (cf. figure 4.13) has been chosen for the simulations. The dielectric constants of the three minerals (quartz, plagioclase and muscovite) were taken from literature. Since these values vary in a wide range, the Bruggeman's effective medium theory (Bruggeman, 1935) is used to connect the bulk material data with those from the minerals. These four dielectric material values (dielectric constant of granite, quartz, plagioclase and muscovite) have to fulfill equation 4.6 (Bruggeman, 1935).

$$f_q \frac{\epsilon_{r,q} - \epsilon_{r,g}}{\epsilon_{r,q} + 2\epsilon_{r,g}} + f_p \frac{\epsilon_{r,p} - \epsilon_{r,g}}{\epsilon_{r,p} + 2\epsilon_{r,g}} + f_m \frac{\epsilon_{r,m} - \epsilon_{r,g}}{\epsilon_{r,m} + 2\epsilon_{r,g}} = 0 \tag{4.6}$$

The properties of quartz in equation 4.6 are represented by index q, those of plagioclase by index p, muscovite by index m and granite by index g. Variable f describes the volume fraction of the respective phase inside the granite ($f_q = 0.27$, $f_p = 0.53$, $f_m = 0.2$). Equation 4.6 combined with literature values summarized in table 2.2 results in $\epsilon_{r,q} = 4.28 + 0.0005i$ for quartz, $\epsilon_{r,p} = 6.57 + 0.0635i$ for plagioclase and $\epsilon_{r,m} = 4.43 + 0.0005i$ for muscovite.

Considering these values only plagioclase absorbs microwaves whereas quartz and muscovite are almost transparent for the microwaves.

Specific heat capacity of the three component model

The specific heat capacity of the granite samples was measured in a temperature range from 25°C to 1000°C (figure 4.14). The values for quartz and plagioclase were found in the literature whereas the c_p of muscovite is calculated using a Voigt-Reuss-Hill mixture rule (Peselnick and Meister, 1965).

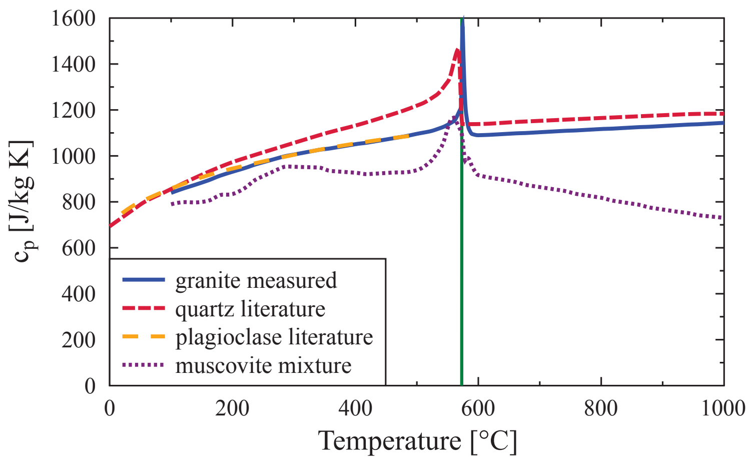


Fig. 4.14 Specific heat capacity c_p [J/kg K] of granite (Hartlieb et al., 2016), quartz (Carpenter et al., 1998), plagioclase (Benisek et al., 2013) and muscovite as a function of temperature [°C] (Toifl et al., 2016a).

As can be seen in figure 4.14 the specific heat capacity of the granite sample reveals a significant peak at a temperature of 573°C (green vertical line in figure 4.14). This peak results from the quartz phase transformation from trigonal quartz (α quartz) to hexagonal quartz (β quartz). The plagioclase values found in literature only range from 20°C to 500°C (Benisek et al., 2013). In order to countervail the differences in the specific heat values of the minerals, muscovite varies in a range between 800 J/kg K and 1000 J/kg K according to a Voigt-Reuss-Hill averaging.

Thermal conductance of the three component model

Data of the thermal conductance of granite were taken from measurements between 25°C and 1000°C whereas only constant values for plagioclase and values between 25°C and 800°C for quartz were found in literature (cf. figure 4.15). The property of muscovite is unknown and must be derived by a square-root mean averaging (Beardsmore and Cull, 2001; Roy et al., 1981).

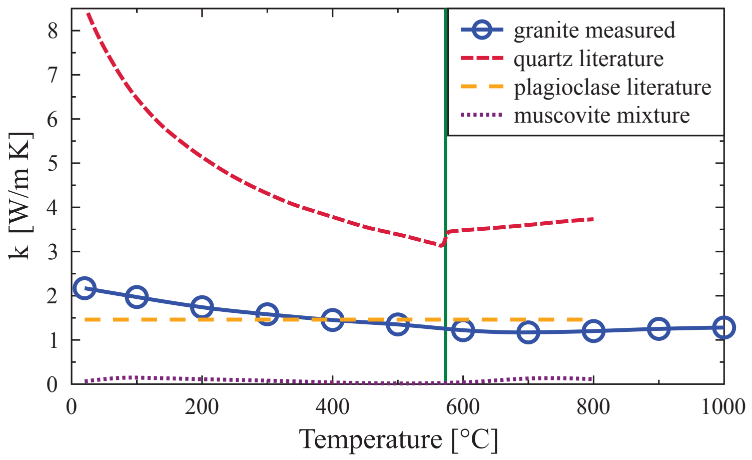


Fig. 4.15 Thermal conductance k [W/m K] of granite (Hartlieb et al., 2016), quartz (Gibert and Mainprice, 2009), plagioclase (Horai and Baldridge, 1972) and muscovite as a function of temperature [°C] (Toifl et al., 2016a).

Only minor changes of the thermal conductance due to the α to β quartz phase transformation of the measured granite is depicted in figure 4.15. The three minerals show a strong variation of the thermal conductance values. This fact can lead to strongly inhomogeneous thermal fields.

Uniaxial thermal strains of the three component model

Uniaxial thermal strains of granite were measured using a NETZSCH 402E, S/N 214 1 555 push rod dilatometer. Five granite samples of 6×6 mm^2 cross section and 25 mm length were heated from 20°C to 1000°C at 3 K/min and cooled down to room temperature again (with 10 K/min). The expansion coefficients of the constituents were taken from literature. The accuracy of the found literature values are verified by repeating the thermal strain experiments numerically and considering the microstructure (cf. section 4.1.3.3).

The measured grain samples reveal a significant change of the uniaxial thermal strain at the α to β quartz phase transformation temperature of 573°C (figure 4.16). Another rapid

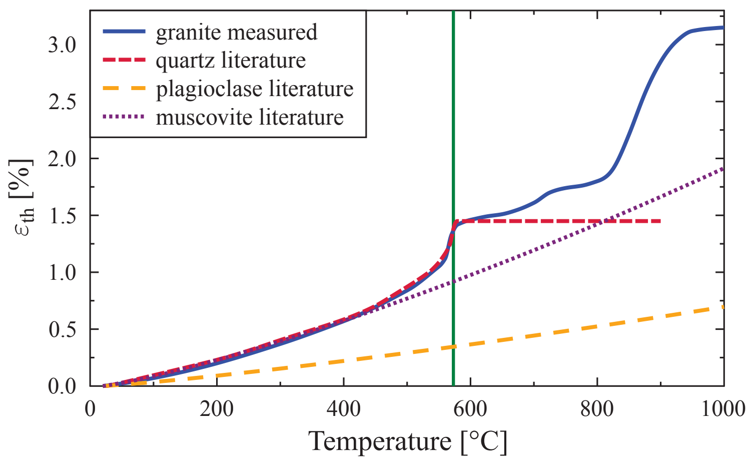


Fig. 4.16 Mean uniaxial thermal strain ε_{th} [%] of granite (Hartlieb et al., 2016), quartz (Carpenter et al., 1998), plagioclase (Skinner, 1966) and muscovite (Saxena et al., 1993) as a function of temperature [°C] (Toifl et al., 2016a).

change of the thermal strain is observed at a temperature of around 800°C. The reason for this increase of expansion is still under investigation (for details see section 4.1.3.3). No phase transformation is observed for plagioclase and muscovite (figure 4.16).

Elastic constants of the three component model

The elastic constants of the three minerals (quartz, plagioclase ($Ab_{47}An_{53}$) and muscovite) were retrieved from literature (figure 4.17). Young's modulus E^* and Poisson ratio ν of the remaining bulk granite is derived by applying the Voigt averaging scheme (Peselnick and Meister, 1965).

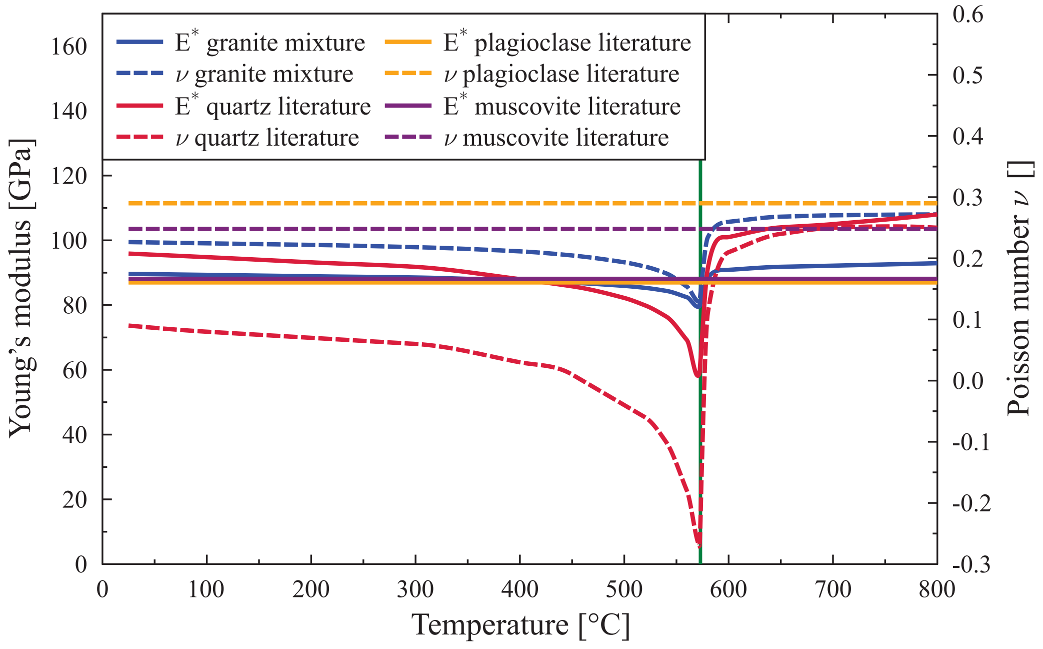


Fig. 4.17 Elastic parameters E^* and ν of granite, quartz (Carpenter et al., 1998), plagioclase ($Ab_{47}An_{53}$) (Hearmon, 1984) and muscovite (Vaughan and Guggenheim, 1986) as a function of temperature [°C] (Toifl et al., 2016a).

4.1.3.3 CDP calibration

In the thermal expansion experiments of granite samples not only the first heating but also a second heating was measured (cf. figure 4.18). These two thermal heating curves are used to calibrate the CDP three component 3D material model.

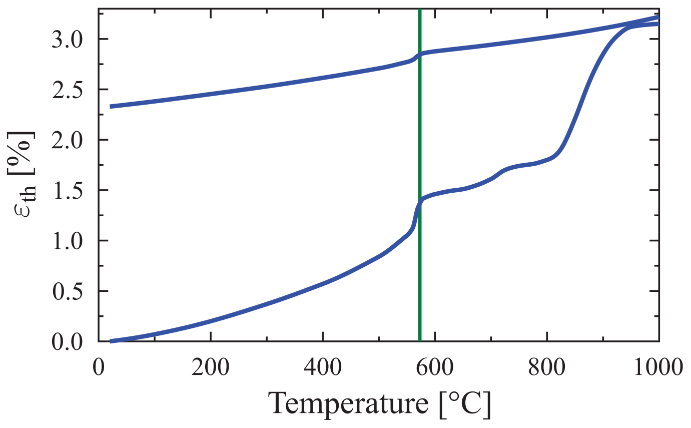


Fig. 4.18 Mean uniaxial thermal strain ε_{th} [%] of granite as a function of temperature [°C] (Hartlieb et al., 2016).

In figure 4.18 a significant variation between the first and second heating test of the same samples are observed. Furthermore, relatively large deviations between single measurements under the same conditions of the five granite samples are determined (cf. figure 3 in Hartlieb et al. (2016)). These strong variations can be explained by the relatively large grain diameters (2 - 3 mm) compared to the sample cross section (6×6 mm^2). If multiple quartz grains accumulate at a cross section, they dominate the thermal expansion behavior of the entire sample.

Besides the jump of thermal expansion at the well known α to β phase transformation at 573°C, another rapid change at temperatures around 800°C is observed. This second jump is governed by the phase transition of quartz at 870°C with a change to hexagonal tridymite, which is also accompanied by a significant reduction in density to 2.25 $^{\mathrm{g}}/_{\mathrm{cm}^3}$ (Okrusch and Matthes, 2005). In Heaney (1994), the author emphasizes that the β quartz tridymite transition only occurs if certain impurities are included in the quartz crystals. However, this jump in the thermal strains at temperature greater than 800°C is not visible in the second heating cycle. Unfortunately, the cooling curves are not available due to measurement issues, which would allow conclusions if the tridymite phase transformed back to β quartz or not. After the first thermal heating a permanent plastic strain of 2.33% is observed. Moreover, it can be seen that not only the second jump in the thermal expansion during reheating is missing but also the α to β phase transformation is reduced.

For the CDP material calibration the thermal expansion experiments are repeated numerically by considering the microstructure of the granite sample. To this end, the same sample dimensions $6 \times 6 \times 25$ mm^3 (cf. figure 4.19) and phase fractions (27 vol.% quartz, 53 vol.% plagioclase and 20 vol.% muscovite) as in the experiments are used. The microstructure is created by a Voronoi tessellation algorithm (Quey et al., 2011) and an averaged grain diameter of 2 mm resulting in 215 grains is assumed (details on the microstructure creation can be found in section 4.1.3.1). The nodes on the front surface (red points in figure 4.19) are constrained to the xy plane and the displacements in x- and y-direction of the middle points on the front and opposite plane (green stars in figure 4.19) are set to zero. Moreover, the middle point of two edges at the front surface are constrained in the direction of the edge (orange polygon in figure 4.19 constrained in y-direction and purple one in x-direction).

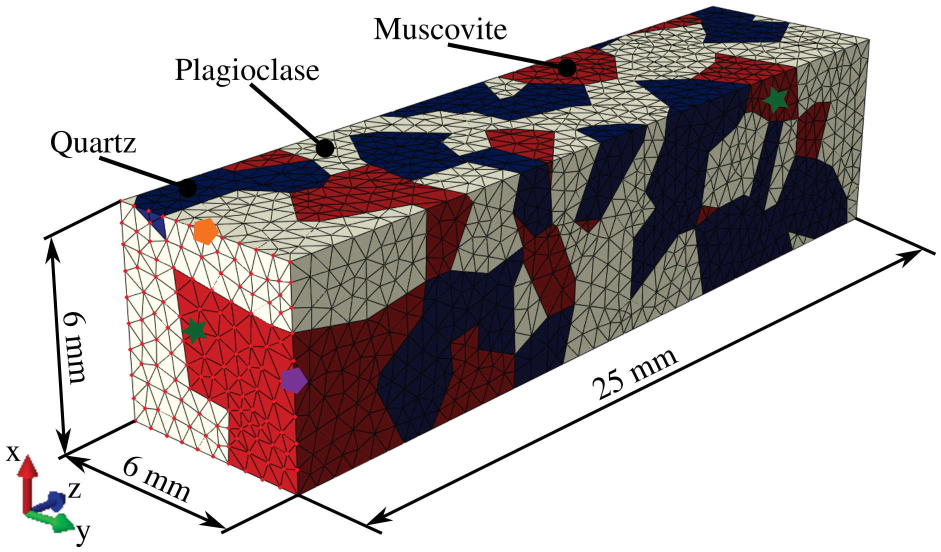


Fig. 4.19 Model for CDP material calibration including microstructure (quartz is blue, plagioclase beige and muscovite red).

The test consists of a heating step (19600 s $\widehat{=}$ 3 K/min) where all nodes are heated linearly from 20°C to 1000°C followed by a step where the temperature is held (600 s) and a cooling step (5880 s) from 1000°C to 20°C (linear). Afterwards, the second heating is carried out. Since the sample is small, all nodes are simultaneously set to the outer applied temperature. For the comparison with the measured thermal strains the maximum displacement of the free surface is measured and divided by the initial length. Moreover, it is assumed that the jump in the thermal strains at temperatures above 800°C is caused by the β quartz to tridymite phase transformation of quartz. Therefore, the thermal expansion of quartz is adapted during the calibration analysis for temperatures higher than 800°C in order to reach the final measured expansion of the granite sample. Since no rapid increase during the second heating is observed in the granite samples, no phase transformation of β quartz to tridymite is modeled. Most of the CDP parameters were taken from literature or set to the recommended values from *Abaqus* (Abaqus, 2014). The parameters with the respective source are summarized in table 4.4.

The fracture toughness K_{IC} is a bulk material parameter which is retrieved from Nasseri et al. (2005) for an averaged granite with a similar plagioclase content. Based on the fracture toughness and the elastic properties of granite for room temperature (E^* = 89.6 GPa and ν = 0.23), the critical energy release rate G_{IC} is calculated. Plane strain conditions are assumed (cf. equation 4.7).

$$G_{IC} = \frac{K_{IC}^2}{E^*}\left(1 - \nu^2\right) \tag{4.7}$$

Table 4.4 Concrete damaged plasticity material parameters for the 3D three component model.

CDP parameter	**Material**	**Value**	**Source**
K_{IC} [MPa$\sqrt{\text{m}}$]	Granite	2.23	Nasseri et al. (2005)
G_{IC} [J/m²]	Granite	52.68	
σ_{t0} [MPa]	Granite	9	Hustrulid et al. (2001)
	Quartz	25	Wang (2015)
	Plagioclase	23	Wang (2015)
	Muscovite	9.1	Wang (2015)
σ_{c0} [MPa]	Granite	85.45	Arzúa and Alejano (2013)
ϵ^* []	Granite	0.15	Abaqus (2014)
σ_{b0}/σ_{c0} []	Granite	2.34	Arzúa and Alejano (2013)
K_c []	Granite	0.56	Arzúa and Alejano (2013)
μ^* []	Granite	10^{-4}	

The compressive strength σ_{c0} is retrieved from Arzúa and Alejano (2013) who performed triaxial strength tests on an Amarelo Pais granite, which shows comparable volume fractions of the three minerals. The σ_{b0}/σ_{c0} value is derived from dividing the biaxial compression ($\sigma_{b0} \approx 200$ MPa) by the uniaxial compression strength ($\sigma_{c0} = 85.45$ MPa) (Arzúa and Alejano, 2013). The parameter K_c is determined by taking the friction angle ϕ from the Mohr-Coulomb model determined in Arzúa and Alejano (2013) ($\phi = 57.59°$) and using equation 4.8 (López-Almansa et al., 2014).

$$K_c = \frac{3 - \sin\phi}{3 + \sin\phi} \tag{4.8}$$

The dilation angle ψ is defined as a function of the maximum plastic shear strain γ^{pl} (cf. equation 3.45). For the used granite Arzúa and Alejano (2013) determined the three required parameters: $a^* = 30.95$, $b^* = 8.97$, $c^* = 0.654$. The function is shown in figure 4.20.

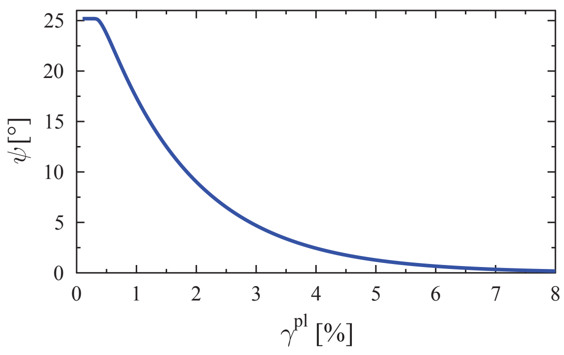


Fig. 4.20 Dilation angle ψ [°] as a function of the maximum plastic shear strain γ^{pl} [%] (Arzúa and Alejano, 2013).

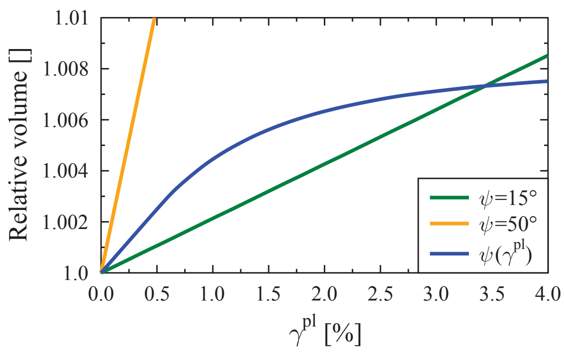


Fig. 4.21 Relative volume $(1+{}^{\Delta V}/_{V})$ of the unit cell under shear loading for various dilation angles ψ as a function of γ^{pl}.

In figure 4.21 the relative volume of a unit cell $(1+{}^{\Delta V}/_{V})$ with periodic boundary conditions under shear loading is visualized. In the case of constant dilation angle the volume increases linearly with increasing plastic strains. The slope of the line is determined by the dilation angle. Assuming $\psi = 15°$ the slope of the curve is significantly smaller than in the $\psi = 50°$ case. By applying the γ^{pl} dependent dilation angle proposed by Arzúa and Alejano (2013), the increase of the volume for higher plastic flow values becomes significantly less (cf. figure 4.21).

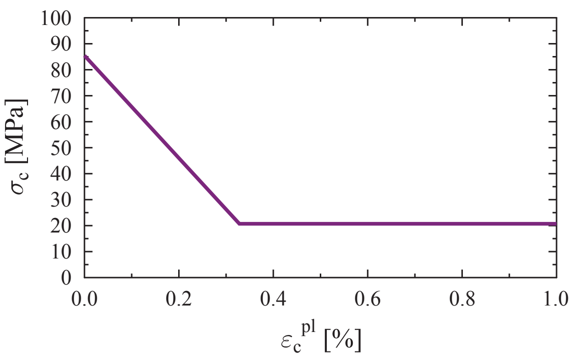


Fig. 4.22 Stress-strain damage behavior under compression of investigated granite (Arzúa and Alejano, 2013).

The tension damage behavior is visualized in figure 4.4. The parameters are summarized in table 4.5. For the compression damage behavior the material test data of Arzúa and Alejano (2013) are used (cf. figure 4.22). The resulting compression parameters are summarized in table 4.5.

Table 4.5 CDP parameters of bulk granite for 3D model (Arzúa and Alejano, 2013; Nasseri et al., 2005).

Load typ	**Component**	**Damage variable d**	**cracking / crushing strain $\tilde{\varepsilon}$**
Tension	Quartz	0.0 0.57	0.0 4.214×10^{-6}
	Plagioclase	0.0 0.57	0.0 4.581×10^{-6}
	Muscovite	0.0 0.57	0.0 1.1578×10^{-5}
	Granite	0.0 0.57	0.0 1.1707×10^{-5}
Compression	Granite	0.0 0.7576 0.7576	0.0 0.0033 0.1

For all material parameters with unknown values for the minerals the global granite values are used. The resulting CDP material model is further used to calculate the damage behavior of the granite samples under microwave irradiation.

4.1.3.4 FDTD model

Since the 3D FDTD models contain a huge number of degrees of freedom, the open source software *Meep* (Oskooi et al., 2010) has been used, which supports MPI and therefore parallel computing. However, *Meep* only provides the basic FDTD solver. In order to optimize the computation on a cluster, the numerical module "FDTD electric field calculation" in figure 4.1 has been adapted and refined (cf. figure 4.23).

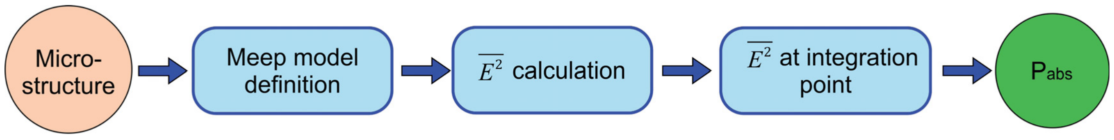

Micro-structure
Meep model definition
$\overline{E^2}$ calculation
$\overline{E^2}$ at integration point
P_{abs}

Fig. 4.23 3D *Meep* FDTD electromagnetic field calculation methodology.

In figure 4.23 the three main steps of the FDTD electric field calculation are visualized. Each of these modules represents a single *C++* program which can be found in appendix B. First, the dielectric properties are defined in the *Meep* model and then the electromagnetic field is calculated. In this module the electric field components at each time step of the investigated period are saved in a HDF5 file. In the next step these HDF5 files are read in and the time averaged squared electric field $\overline{E^2}$ is determined. The resulting $\overline{E^2}$ is saved once again in a HDF5 file and serves as the basis for the absorbed power density P_{abs} at the integration points of each finite element.

Meep **model definition**

First the model for the FDTD calculations has to be defined. In figure 4.24 the FDTD model for the three component 3D case ($60 \times 110 \times 60$ cm^3) is visualized.

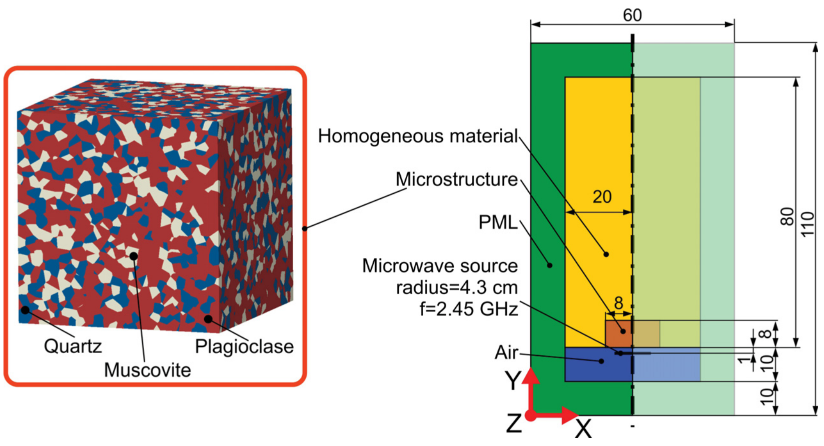


Fig. 4.24 FDTD three component 3D model; all dimensions in centimeters (Toifl et al., 2016a).

The two component 3D case including the isometric view is summarized in figure 4.8. In addition to the quarter rock model (microstructure cube plus homogeneous material), both models (two and three component) also include air in front of the material as well as perfectly matched layers (PML) at the domain limits. The PML acts as a perfectly absorbing boundary which allows to truncate the simulation space without causing reflections of the electromagnetic field (Oskooi and Johnson, 2011). The thickness of the layer is chosen to be of the magnitude of the wavelength of the electromagnetic field. In the air the wavelength of a beam with a frequency of 2.45 GHz is $\lambda_{air} = 12$ cm, in the two component material ($\epsilon_{r,eff} = 7.37 + 0.90i$) $\lambda_{2comp} =$ 4.5 cm and in the three component model ($\epsilon_{r,eff} = \epsilon_{r,g} = 5.45 + 0.03i$) $\lambda_{3comp} =$ 5.2 cm (cf. equation 2.15). Since most of the PML is located around the material phase, a thickness of 10 cm is chosen and verified to be sufficient by various numerical simulations. The difference in the y-dimension between the two ($y_{material} = 30$ cm) and three component model ($y_{material} = 80$ cm) is caused by the strongly varying dielectric properties. Since significantly bigger penetration depths are expected due to the small permittivity in the three component model, the dimensions have to be extended in order to avoid reflection at the back surfaces. Both described models represent the case where the material boundaries (except the front face) do not influence the microwave distribution. This condition can frequently be found in comminution processes (e.g. irradiation of a huge block of rock).

Similar to the 2D case a microwave beam with a typical technical frequency of 2.45 GHz with Gaussian shape and a waist radius of 4.3 cm is applied. It can be assumed that the Gaussian beam describes a planar wave (Jackson, 2011) at the source position which is parallel to the xz plane. The source plane is positioned 1 cm in front of the rock model (cf. figure 4.24) and emits a beam which propagates in positive y-direction, whereas it is polarized in z-direction. Since soft sources are used in *Meep*, the equivalent electric ($\underline{J}$) and magnetic currents ($\underline{K}^*$) – representing the source – are derived based on the "total-field / scattered-field" approach presented in section 3.1.3.3. Based on the assumption of a plane wave at the source position the desired electric $\underline{E}^\star = 1\ \underline{e_z}$ and magnetic fields $\underline{H}^\star = 1\ \underline{e_x}$ (in *Meep* dimensionless units) are defined (Jackson, 2011; Oskooi and Johnson, 2013). By considering equation 3.21 the equivalent electric ($\underline{J}$) and magnetic currents ($\underline{K}^*$) are derived (equation 4.9).

$$\underline{J} = \begin{pmatrix} 0 \\ 0 \\ -H_x^\star \end{pmatrix}, \underline{K}^* = \begin{pmatrix} -E_z^\star \\ 0 \\ 0 \end{pmatrix} \tag{4.9}$$

Since *Meep* provides only simple geometry (e.g. spheres, cuboids) for the definition of the permittivity in the FDTD model, a self-written *C++* script is used to define the spatial permittivity distribution manually. The algorithm identifying the grain (polyhedron) to which a certain FDTD point belongs to is visualized in figure 4.25. It can then be assigned its individual dielectric properties.

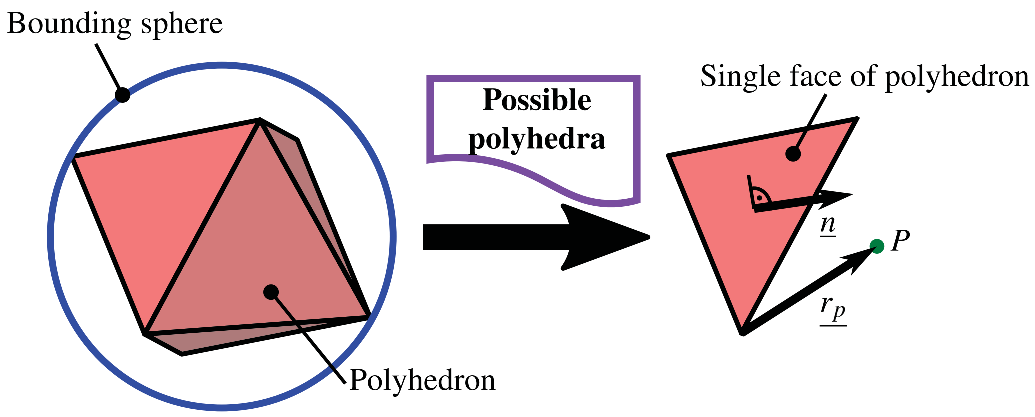


Fig. 4.25 Algorithm to identify the grain (polyhedron) a certain FDTD point belongs to (cf. Cyrus and Beck (1978); Ericson (2005)).

Before the *Meep* calculation starts, a bounding sphere (Ericson, 2005) for each polyhedron is calculated. Since *Neper* already provides the middle point of the polyhedron, the radius of a bounding sphere can be calculated by looping over all vertexes of the polyhedron to determine the greatest distance from the middle point. This distance represents the radius of the sphere. When the *Meep* calculation starts the solver loops over all polyhedra and determines which bounding spheres the FDTD grid point under investigation is located in. This is done by calculating the distance between the middle point and the grid point. If this distance is smaller than the radius the point must be inside the sphere. In the next step the possible polyhedra are analyzed more accurately by exploiting the convex behavior of the grains. Therefore, a loop over all faces of a polyhedron is built where a vector $\underline{r_p}$ from any point of the face to the investigated grid point P is constructed (cf. figure 4.25). If the scalar product between this vector $\underline{r_p}$ and the normal vector $\underline{n}$ of the face pointing inward is positive for all faces of the polyhedron, the point lies in the respective grain (figure 4.25, cf. Cyrus and Beck (1978)). Finally, the permittivity of the phase of the respective polyhedron can be set to the grid point.

In the 3D FDTD simulations a Courant number S of 0.5 is chosen. In a parametric analysis the influence of the grid constant on the accuracy of the resulting thermal field combined with the computation effort has been investigated and a grid constant $\Delta x = \Delta y = \Delta z$ of 1 mm has been derived as the most appropriate value for the current models. The electric field components are saved in a HDF5 file when the electric field has stabilized after the 24th period. The 24th period has been determined iteratively.

Time averaged squared electric field

After the electric field at each time step of the 24th period has been saved the time averaged squared electric field $\overline{E^2}$ is derived. This decoupling of the FDTD electric field calculation and the time integration of the electric field are chosen in order to speed up the calculations. $\overline{E^2}$ in each FDTD grid point is calculated according to equation 4.10.

$$\overline{E^2} = \sqrt{\left(\mathfrak{T}^{-1}\int_{23\mathfrak{T}}^{24\mathfrak{T}} E_x^2\,dt\right)^2 + \left(\mathfrak{T}^{-1}\int_{23\mathfrak{T}}^{24\mathfrak{T}} E_y^2\,dt\right)^2 + \left(\mathfrak{T}^{-1}\int_{23\mathfrak{T}}^{24\mathfrak{T}} E_z^2\,dt\right)^2} \tag{4.10}$$

In equation 4.10 $\mathfrak{T}$ is the period defined as $1/f^*$. Since the electric field components (E_x, E_y, E_z) are not available as a function of time the integration is performed numerically. To this end, a trapezoidal integration rule (cf. Atkinson (1989)) is used. In equation 4.11 the time averaged E_x value is calculated exemplarily.

$$\mathfrak{T}^{-1}\int_{23\mathfrak{T}}^{24\mathfrak{T}} E_x^2\,dt = \frac{1}{n-1}\left(\frac{E_{x,1}^2}{2}+\frac{E_{x,n}^2}{2}+\sum_{i=2}^{n-1}E_{x,i}^2\right) \tag{4.11}$$

The variable n in equation 4.11 describes the number of time increments (and therefore output points) within the 24th period. The final equation 4.12 to determine the time averaged squared electric field $\overline{E^2}$ is depicted below.

$$\overline{E^2} = \sqrt{\begin{aligned}&\left(\frac{1}{n-1}\left(\frac{(E_{x,1})^2}{2}+\frac{(E_{x,n})^2}{2}+\sum_{i=2}^{n-1}(E_{x,i})^2\right)\right)^2+\\ &\left(\frac{1}{n-1}\left(\frac{(E_{y,1})^2}{2}+\frac{(E_{y,n})^2}{2}+\sum_{i=2}^{n-1}(E_{y,i})^2\right)\right)^2+\\ &\left(\frac{1}{n-1}\left(\frac{(E_{z,1})^2}{2}+\frac{(E_{z,n})^2}{2}+\sum_{i=2}^{n-1}(E_{z,i})^2\right)\right)^2\end{aligned}} \tag{4.12}$$

In order to scale the electric field output to the source value at the Gaussian axis the same FDTD model without any dielectric material is calculated. Equation 4.12 is also applied to this model to calculate the time averaged squared electric field at the source position $\overline{E_{0air}^2}$. Based on a parameter study it is concluded that a numerical integration of only half a period as well as considering only every second time step is sufficient. With this reduced integration scheme the calculations can be accelerated without causing significant inaccuracies.

Electric field at integration points

$\overline{E^2}$ has to be determined at the integration point positions of the finite element mesh in order to link the electromagnetic with the thermal field (by the absorbed power density). First, the positions of the integration points are read out during a dummy heat transfer analysis. The *DFLUX* subroutine is used to read the integration points out in a text file (Abaqus, 2014). In the next step $\overline{E^2}$ of each integration point is derived by searching finite difference grid points close to the respective point. Afterwards, the corresponding phase of the point has to be determined by the same algorithm as described in "*Meep* model definition". Finally, the absorbed power density P_{abs} at the integration point is derived by equation 2.17 and the resulting value is written to a text file.

4.1.3.5 FEM model

The dimensions of the 3D FE model equal the quarter model rock and are visualized in figures 4.26 and 4.27 (e.g. figures 4.24 and 4.8 homogeneous material plus microstructure). The cube containing the microstructure has already been meshed by the open-source program *Neper* (Quey et al., 2011). In the homogeneous part a global element size ranging from 0.25 cm up to 1 cm is assumed which leads to 281988 linear hexahedral elements in the two component case and 871640 in the three component model. The entire model contains 4342673 elements, 1000173 nodes (two component model) and 4932325 elements, 1604713 nodes (three component model).

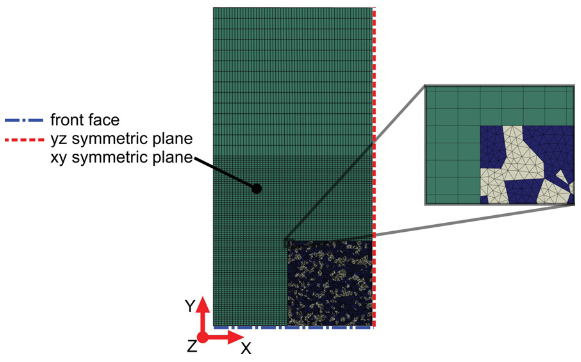


Fig. 4.26 FE mesh of the two component model with linear tetrahedral and hexahedral elements (Toifl et al., 2016b).

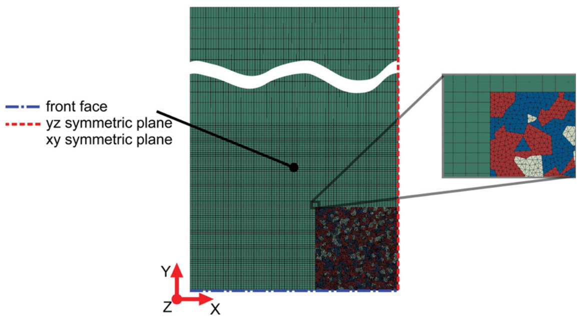


Fig. 4.27 FE mesh of the three component model with linear tetrahedral and hexahedral elements (Toifl et al., 2016a).

The two domains (microstructure cube and homogeneous part) are joined together by means of tie constraints of the respective surfaces. In the anisotropic model (anisotropic thermal conductance, thermal expansion and elastic constants), the local Cartesian coordinate system of each quartz grain (phase T) is randomly defined. Since no feedback of the displacement field on the heat flux is assumed, the thermomechanical simulation is divided in an uncoupled manner into a heat transfer and a subsequent stress analysis (Abaqus, 2014).

In the heat transfer model the absorbed power density (P_{abs}) is read in at each integration point (4 integration points in each tetrahedral finite element and 8 in each hexahedral element). The source term of the heat conduction equation (body heat flux s) is calculated by multiplying the absorbed power density with the iteratively determined constant C. Therefore, the thermal energy inside the model $\mathfrak{E}_{th}$ (cf. equation 4.2) is compared to the total provided energy $\mathfrak{E}_{prov}$ (equation 4.13) minus 30% losses.

$$\mathfrak{E}_{prov} = t\,\frac{P_{mw}}{4} \tag{4.13}$$

The factor 4 in equation 4.13 accounts for the quarter symmetry of the FE model. In the two component case a microwave source of P_{mw} = 25 kW is used whereas in the three component case varying power levels are investigated (cf. tables 4.6 and 4.7).

Table 4.6 Four different irradiation times at two microwave power levels used for the three component model (Toifl et al., 2016a).

Irradiation time [s]	**Microwave power P_{mw} [kW]**
15	25
30	30
60	
72	

On the front face of the material model (blue dash-dotted line in figures 4.26 and 4.27) a thermal conductance of 20 $^{W}/_{m^2\,K}$ and an emissivity of 0.8 combined with an ambient temperature of 25°C in the two component and 20°C in the three component case (according to the thermal expansion experiments) are assumed. At all other faces, i.e. the cutting planes where the model is truncated, an emissivity of 0.8 is used. The two component model is heated for 15 s or 25 s, respectively, followed by a natural cool down (i.e. the body heat flux is switched off) for 3600 s. In the three component case varying irradiation times under constant microwave power (table 4.6) as well as under constant energy are investigated (table 4.7).

Table 4.7 Irradiation times and microwave power levels used for equal microwave energy $\mathfrak{E}_{mw} = 1.8$ MJ analysis.

Irradiation time	**Microwave power P_{mw}**
0.1 s	18 MW
1 s	1.8 MW
15 s	120 kW
30 s	60 kW
72 s	25 kW
100 s	18 kW

In the mechanical model the same FE mesh as in the thermal analysis but elements with reduced integration are used. Furthermore, the same two steps (heating and cooling) are analyzed. As an input the time varying temperature field calculated in the previous analysis is applied. On the xy cutting plane (figures 4.26 and 4.27) of the quarter model the displacements in z-direction, and on the yz plane (red dashed line in figures 4.26 and 4.27) the displacements in x-directions are set to zero as the symmetric boundary conditions for the stress calculation. Moreover, the node located at the Gaussian axis on the front face of the model is fixed in y-direction. First linear elastic calculations are performed followed by simulations including the CDP model that allow to assess the induced damage.

4.2 Homogeneous models

4.2.1 Methodology

The 3D homogeneous model considers strong coupling between the thermal and electromagnetic fields. This is achieved by a staggered algorithm. However, there is still only a weak coupling between the displacements and the thermal field (figure 4.28).

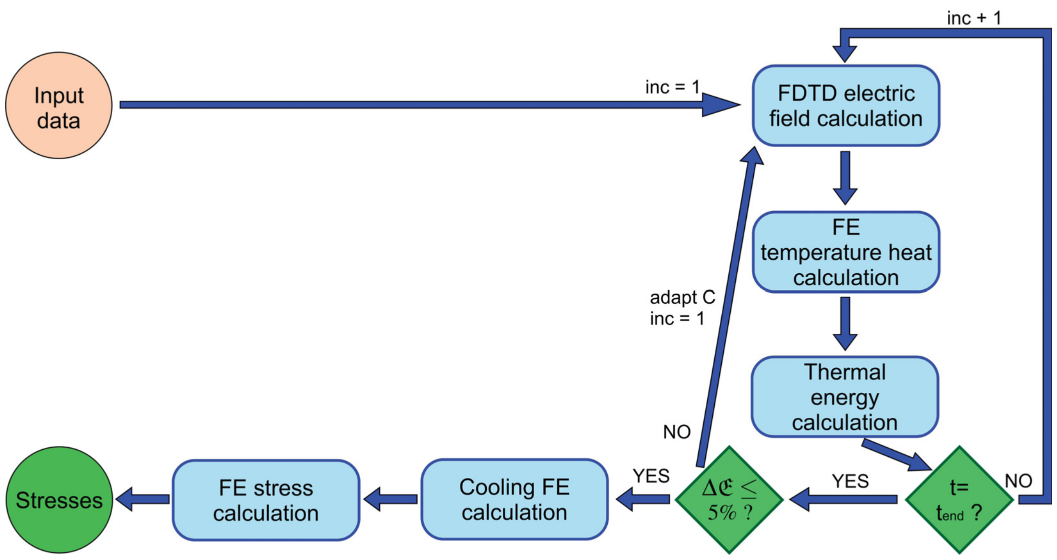

Input data
inc = 1
FDTD electric field calculation
inc + 1
FE temperature heat calculation
Thermal energy calculation
adapt C
inc = 1
NO
YES
$\Delta\mathfrak{C} \leq 5\%$?
YES
t= t_{end} ?
NO
Cooling FE calculation
FE stress calculation
Stresses

Fig. 4.28 Homogeneous model with strongly coupled (SCM) FDTD - FEM simulations.

The simulation methodology of the strongly coupled homogeneous case (SCM) starts with the FDTD electromagnetic field calculation (figure 4.28). In the first step the permittivity values at room temperature are used and the absorbed power density P_{abs} is calculated as an output of the model (cf. section 4.1.3.4). In the subsequent thermal FE simulation only one increment of the FE thermal analysis is calculated with a predefined maximum temperature change per increment (in this study $\Delta T = 50°\text{C}$). Interruption of the heat transfer calculation after one increment is achieved by an *URDFIL* subroutine (Abaqus, 2014). After each increment the thermal field is used as an input for the subsequent electromagnetic field calculation in order to obtain the spatial distribution of the complex permittivity. Therefore, the regular hexahedron mesh is exploited and the FE shape functions are used to interpolate the nodal temperatures. This procedure is repeated until the final irradiation time is reached. In the next step the thermal energy of the heat transfer model is evaluated and compared to the provided microwave energy minus 30% losses (cf. section 4.1.3.5). If the difference is greater than 5% (of the microwave energy) the constant factor C is adapted and the electromagnetic as well as the thermal calculations are repeated. Afterwards, the cooling of the hard rock model is calculated for a duration of 3600 s. Finally, the transient inhomogeneous temperature field is used as an input for the FE stress simulation.

The strong coupling between FDTD and FEM (SCM) is implemented by a global *Python* script which calls other programs (*Python*, *C++* and *Abaqus*) and provides the input data needed by the subsequent calculations. This global script also creates new input files based on the calculation history. The whole procedure is very flexible so that automatically new dimensions or materials can be accounted for (cf. appendix D).

4.2.2 Material data basalt

For the strongly coupled homogeneous simulations (SCM) basalt, as a representative of a rather homogeneous hard rock, is used. The temperature-dependent permittivity was measured and has been published in Hartlieb et al. (2016).

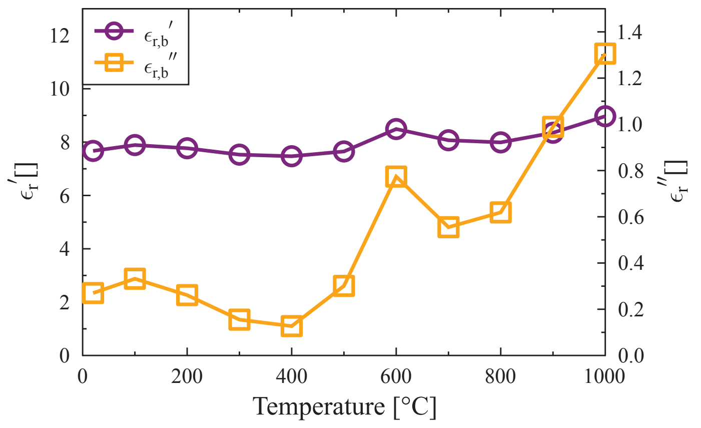


Fig. 4.29 Complex permittivity of basalt as a function of temperature [°C] (Hartlieb et al., 2016).

Figure 4.29 reveals that the real part of the permittivity of basalt is almost constant between 20°C and 500°C whereas the imaginary part slightly increases at first followed by a rapid decrease between 100°C and 400°C. The $\epsilon''_{r,b}$ value at 100°C is a factor of 1.6 higher than the one at 400°C. The thermo-mechanical parameters have been published in Hartlieb et al. (2016) and summarized in appendix E.

Chapter 5

Results

In this chapter the electromagnetic, thermal and mechanical results of the numerical models described in chapter 4 are presented.

5.1 Inhomogeneous models

5.1.1 2D model

The 2D results have been published in Meisels et al. (2015) and Toifl et al. (2014).

5.1.1.1 Electromagnetic results

In the paper Meisels et al. (2015) various combinations of permittivity values have been investigated. However, in this thesis only the case described in section 4.1.2.2 is presented. Figure 5.1 visualizes the time averaged squared electric field E_y^2 after the 24th period when the electromagnetic field has stabilized. The electric field values are normalized to the field value at the source position $E_{y,0air}^2$.

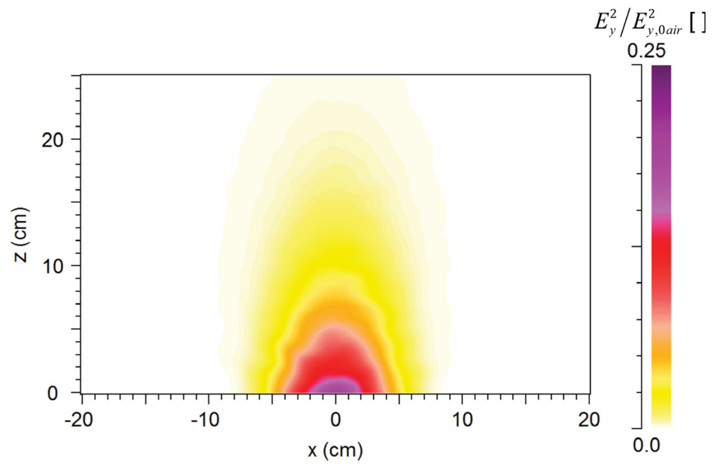


Fig. 5.1 Time averaged squared electric field E_y^2 scaled by the reference value at the center of the source $E_{y,0air}^2$ of the 2D inhomogeneous model (Meisels et al., 2015).

A certain qualitative deviation of the E_y^2 field from an ideal Gaussian beam is observed in figure 5.1. In order to assess this phenomenon in detail, figure 5.2 visualizes the difference between the homogeneous and inhomogeneous model.

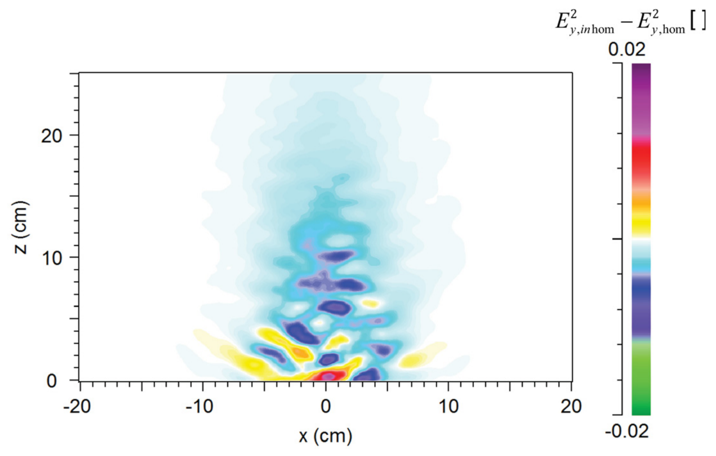


Fig. 5.2 Difference in the electric field value between the inhomogeneous and homogeneous 2D model (Meisels et al., 2015).

In figure 5.2 the strongest positive deviations from the homogeneous case are observed near the air / rock interface (at z = 0 cm). Further inside the material the deviations are negative. The reflections of the microwaves on the disc / matrix boundary are governed by the dissimilar real parts of the permittivity. The absorption of microwaves in the discs ($\epsilon''_{r,d} \neq 0$) leads to higher E_y^2 in the first part of the beam within the rock. Moreover, sideward scattering of the electric field within the first centimeters of the rock is observed.

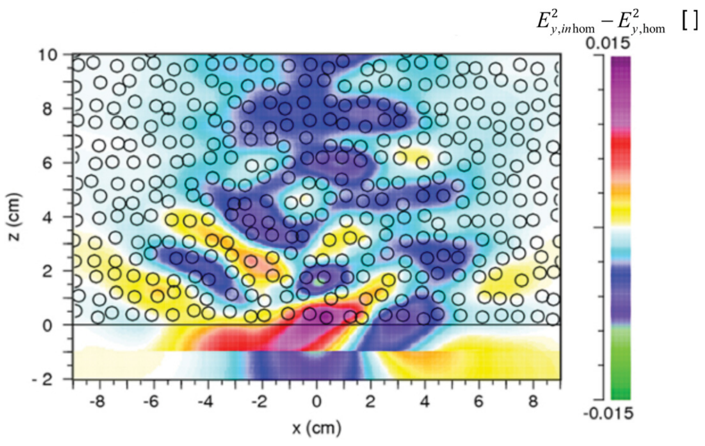


Fig. 5.3 Comparison of disc arrangement (circles) and ΔE_y^2 distribution (color pattern). Enlarged section of figure 5.2 near the microwave source (Meisels et al., 2015).

In order to identify the origin of the stripe-like pattern in the electric field, this pattern is compared with the arrangement of the discs (figure 5.3) and with variations of the density of the discs (figure 5.4). For calculating the variation of the density the disc distribution is smoothed over about 1.5 cm and contours of equal density are drawn. Neither the disc arrangement nor the density variation shows a correlation with the details of the ΔE_y^2 pattern, with the exception of few coincidences in the very first part (up to 10 cm) of the model rock as visible in figure 5.4. In these areas the strong absorption causes negative ΔE_y^2 values (blue / green) at higher density ($>$ 0.34 contours) and positive ΔE_y^2 values (yellow / red) at lower density values ($<$ 0.34 contours).

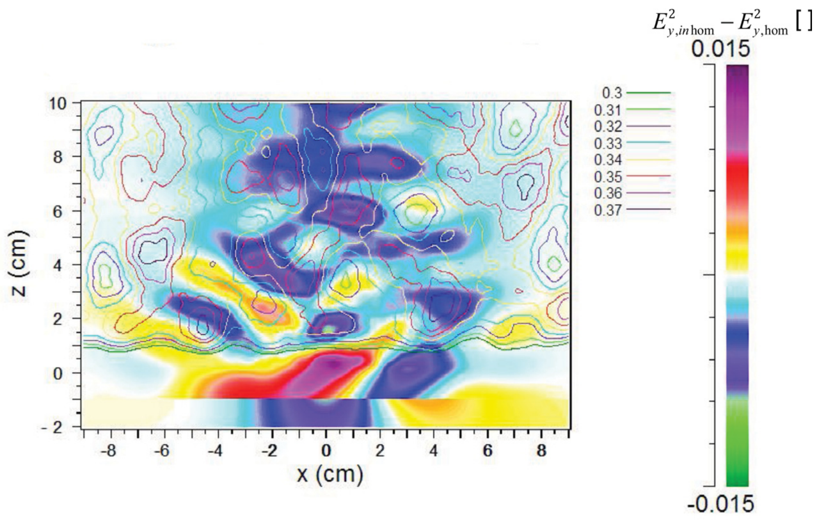


Fig. 5.4 Comparison of density variations of the disc distribution (contour lines) and the ΔE_y^2 distribution as in figure 5.2. Near the boundary of the model rock (x = 0) the values of the density approach zero. Contours with values smaller than 0.3 are not shown in this figure (Meisels et al., 2015).

The stripe-like structure is not exactly regular. Nevertheless, an average period of the pattern can be determined which is about half of the wavelength in the rock ($\lambda_{2comp} = 4.5$ cm). It increases with longer wavelength. This leads to the conclusion that the stripe-like structure is an interference effect dominated by the effective permittivity with the deviations from regularity, and the finer structure depending on details of the interference of partial waves between the discs. Finally, an absorbed power density P_{abs} is derived (figure 5.5).

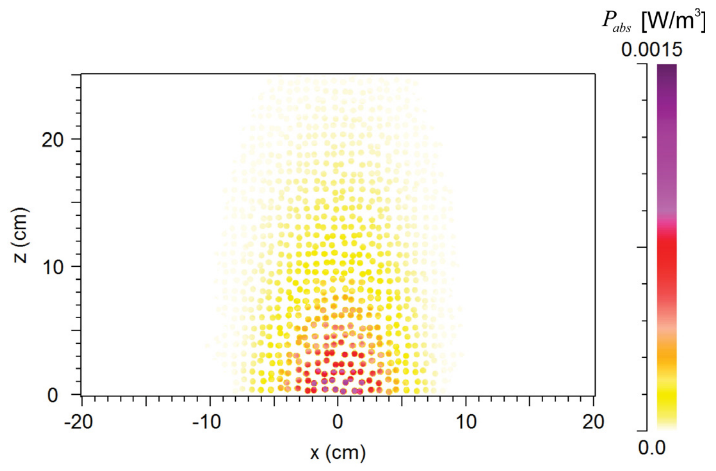


Fig. 5.5 Absorbed power density P_{abs} inside the 2D rock sample (Meisels et al., 2015).

According to the assumption that the matrix is transparent for microwaves ($\epsilon''_{r,ma} = 0$), absorption only occurs in the discs. The discs with the highest values of the normalized absorbed power density are located near the front surface of the 2D model.

5.1.1.2 Thermal results

Based on the absorbed power density (cf. figure 5.5) a heat transfer analysis is performed. Figure 5.6 visualizes the temperature field after a microwave irradiation with 25 kW for 15 s.

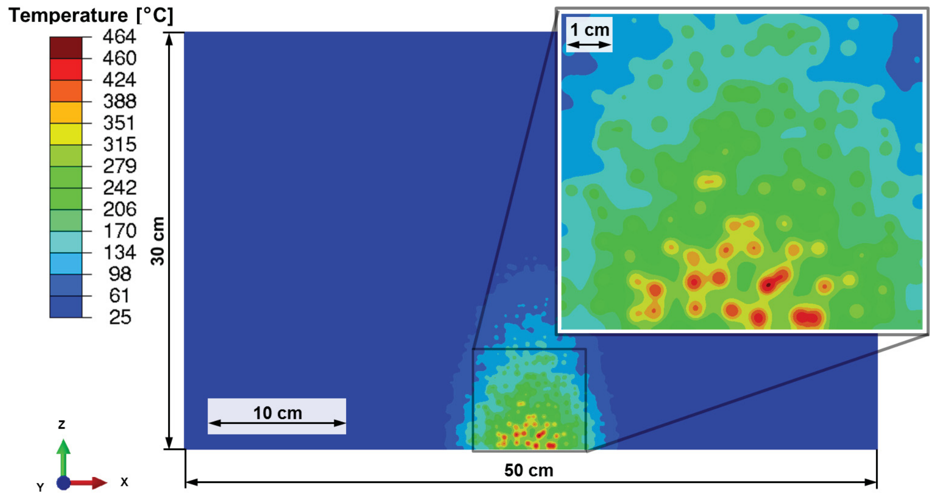


Fig. 5.6 Temperature distribution in °C after 15 s of microwave irradiation (25 kW) in the 2D model rock (Meisels et al., 2015).

The maximum temperature (464°C) in the rock sample is obtained inside an inclusion (two overlapping discs) in a distance of 1 cm from the front surface (figure 5.6). Furthermore, selective heating of the discs due to the selective absorption and variation of the thermal properties can be observed, which leads to significant thermal gradients between the two constituents. 30 s after switching off the microwave source the temperature differences between the discs have equalized and a surface temperature of 310°C is observed agreeing with experimental results.

5.1.1.3 Linear elastic stress results

With the time varying temperature field as input, a subsequent stress analysis is conducted assuming plane strain conditions and a heterogeneous material as in the microwave analysis. No effect due to a sudden change in volume is expected since the temperature does not exceed 573°C - the temperature of the transformation from α quartz to β quartz. As the material can be considered as brittle, the maximum principal stress is an appropriate measure for assessing damage initiation.

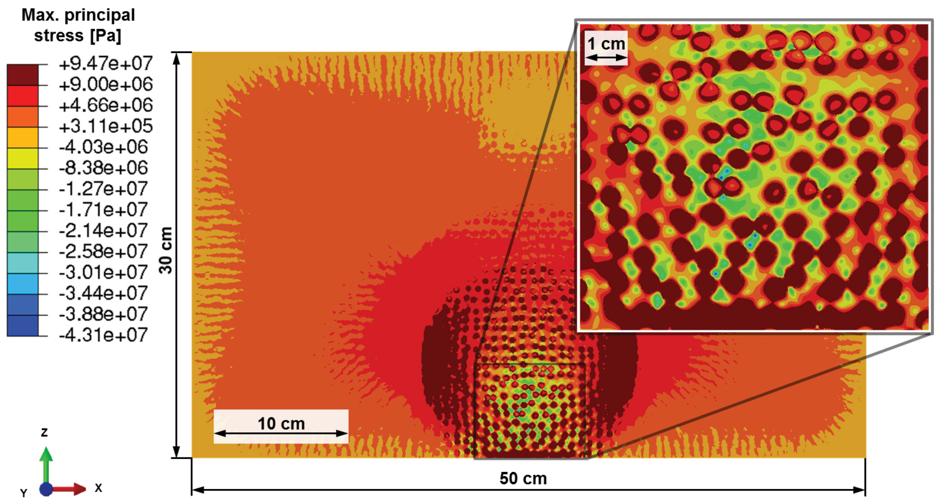


Fig. 5.7 Maximum principal stresses in Pa in the model rock after 15 s of microwave irradiation calculated from the temperature distribution of figure 5.6. Dark red indicates values larger than tensile strength. (Meisels et al., 2015).

High values of maximum principal stresses are observed in areas on the front face as well as right and left of the microwave heated region exceeding the tensile strength of a typical hard rock of about 9 MPa (Hustrulid et al., 2001) (figure 5.7). These stresses are caused by the high thermal gradients combined with the strong mismatch in the thermal expansion of the two phases. Furthermore, critical tensile stresses are obtained in large areas inside the discs particularly along their circumference. Most of the matrix visualized in detail of figure 5.7 is under compression due to strong selective heating and significantly higher thermal expansion of the matrix compared to the discs (cf. table 4.2). The matrix acts as a crack arrest caused by the high compressive stresses.

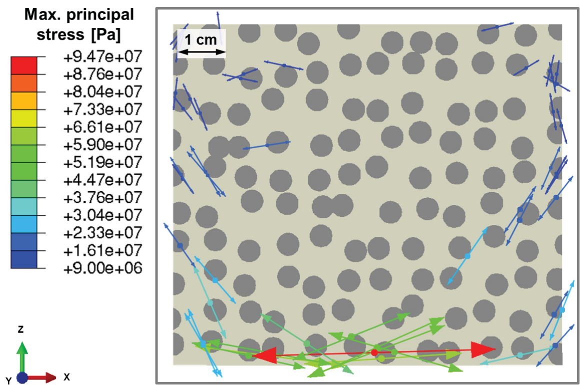


Fig. 5.8 Vector plot of the maximum principal stress (greater than tensile strength of 9 MPa) after 15 s of microwave irradiation in the magnified area shown in figure 5.7; the arrows indicate the direction of the normal to a potential crack plane (Meisels et al., 2015).

Driven by the high maximum principal stresses near the irradiation spot, cracks may initiate at the circumference of the discs as well as in the matrix. Most of the cracks will form in radial direction of the discs (figure 5.8). Based on the highly loaded matrix in this area, damage propagation will take place there as well. The stresses will redistribute once crack propagation has started.

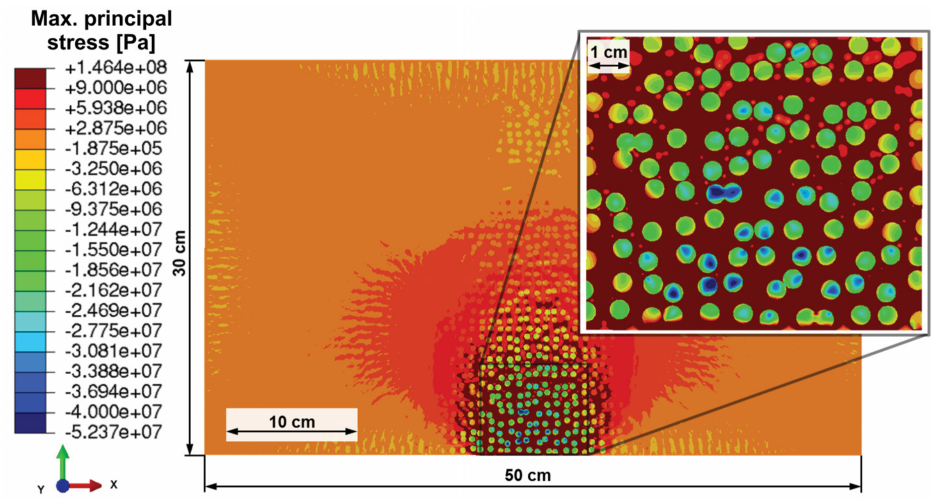


Fig. 5.9 Maximum principal stresses in Pa in a comparative model rock with permuted material definition (discs are composed of quartz and matrix of plagioclase) after 15 s of microwave irradiation.

A comparative analysis in which a permuted material definition is assumed (discs are composed of quartz and matrix of plagioclase) yields a different behavior (figure 5.9). Now high values of maximum principal stresses are observed in large areas inside the matrix. Interface cracks in the matrix may initiate between the various discs as well as between the inclusions and the boundary surface. Initial cracks can potentially grow within the highly loaded matrix entailing severe rock fragmentation.

5.1.1.4 Damage results

By applying a concrete damaged plasticity model the microwave induced damage can be quantified. Figure 5.10 shows the scalar stiffness degradation variable *d* (cf. equation 3.28) at the end of the heating step (in the 2D CDP material only tension damage is considered).

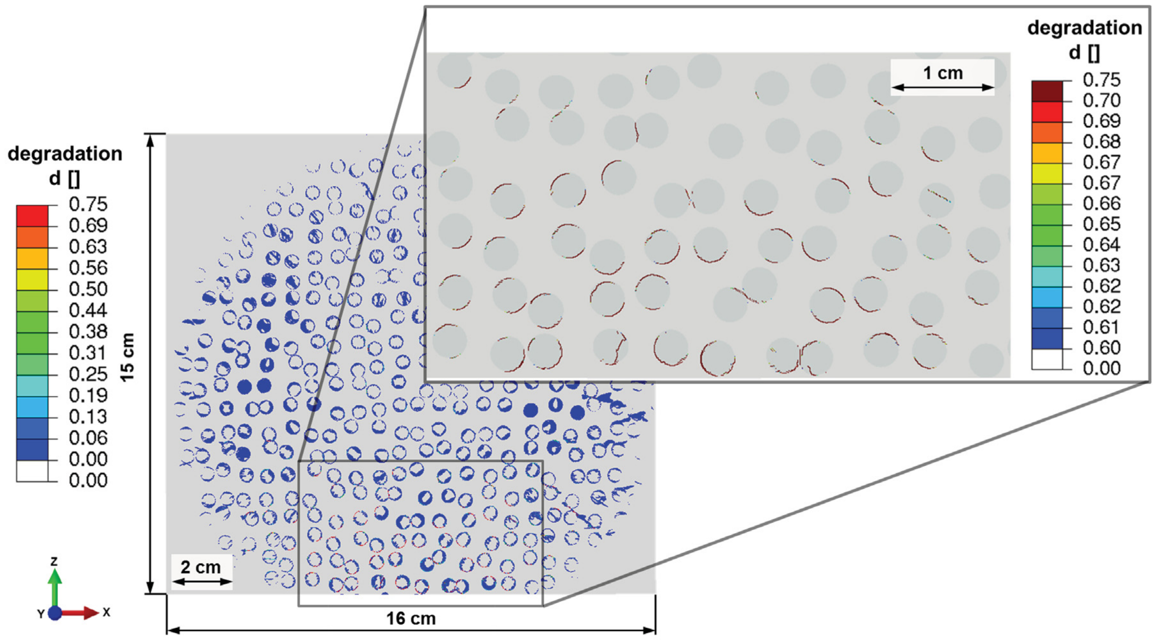


Fig. 5.10 Scalar degradation *d* of material stiffness after 15 s of microwave irradiation; left picture corresponds to the refined area in figure 4.5 (Toifl et al., 2014).

After 15 s of microwave irradiation damage initiation and subcritical cracks (blue areas) are observed in the discs around the irradiation spot (approximately in an area of 16 cm x 15 cm). Near the front surface the discs (circles in magnified area of figure 5.10) contain critical cracks which are able to propagate in an instable way (dark red lines in detail of figure 5.10). Indeed, most of these critical cracks are arranged around the circumference of the discs in agreement with the vector plot of the maximum principal stresses (cf. figure 5.8).

Strong crack propagation in the cooling step is observed in figure 5.11. The simulation aborted after 15.15 s of cooling (maximum temperature 343°C) due to strong local discontinuities caused by the crack propagation. Critical cracks propagate far outside the microwave irradiated area mainly following the disc / matrix boundaries. Only few discs (circles in

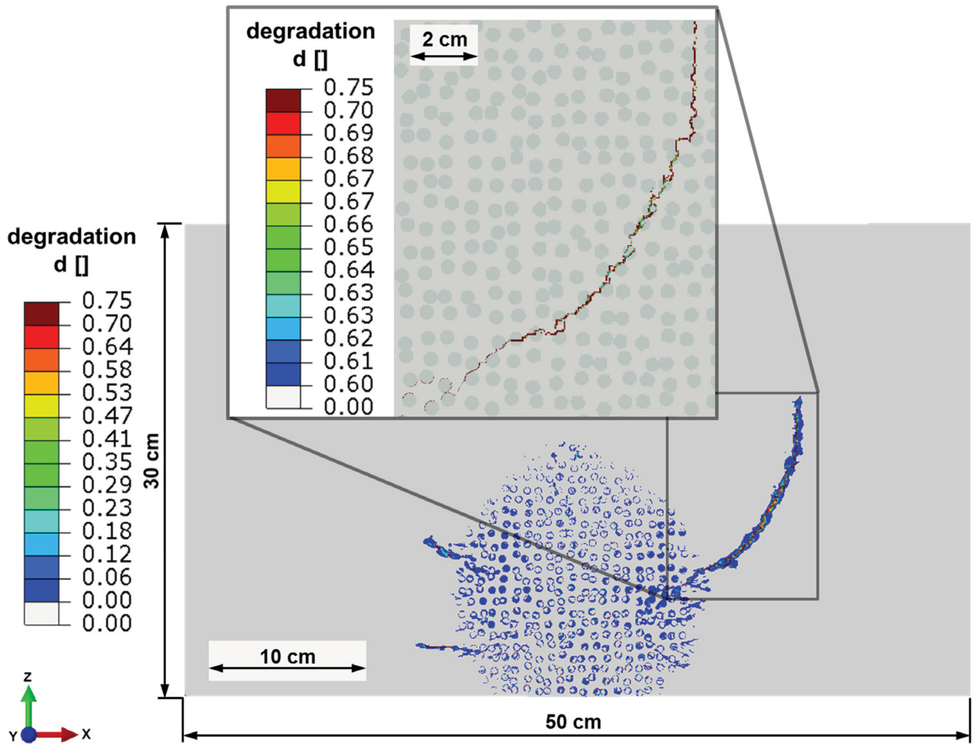


Fig. 5.11 Scalar degradation d of material stiffness after 15.15 s of cooling. Cracks mainly following the disc / matrix boundaries (Toifl et al., 2014).

detail of figure 5.11) are cracked in radial direction as well. Further crack propagation is expected until the ambient temperature is reached.

5.1.2 3D model of the two component system

The major results of the 3D two component models have been published in Toifl et al. (2015a,b, 2016b). Toifl et al. (2016b) presents a detailed analysis of the stress formation on a microstructure level including anisotropic quartz grain behavior and α to β quartz phase transformation.

5.1.2.1 Electromagnetic results

The electromagnetic field inside the microstructure of the reference model (model A in figure 4.7) is calculated. To this end, the time averaged squared electric field ($\overline{E^2}$) is evaluated after the 24th period, when the electric field has stabilized. This period has been found iteratively. In figure 5.12 the $\overline{E^2}$ field scaled by the reference value at the center point of the source ($\overline{E^2_{0air}}$) in air is illustrated.

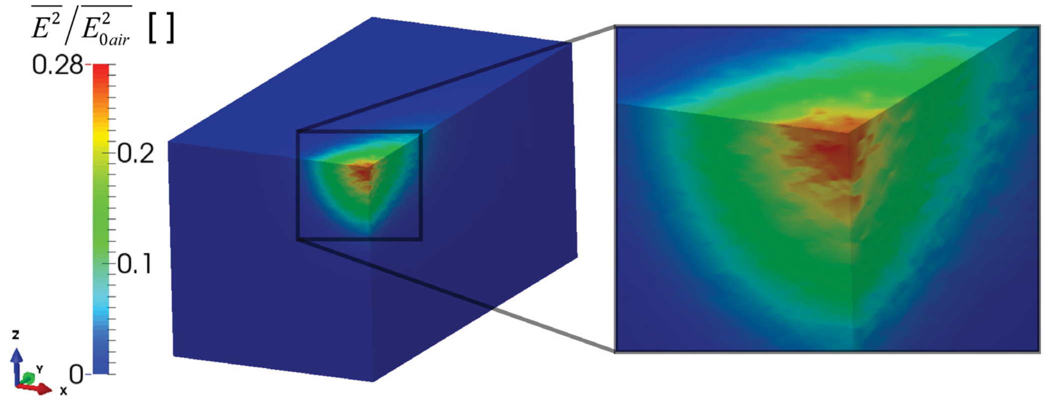
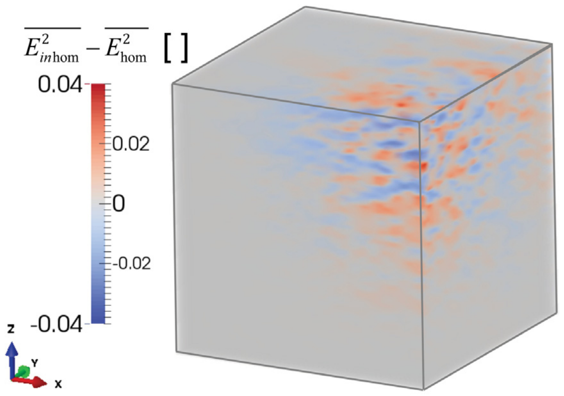


Fig. 5.12 Time averaged squared electric field ($\overline{E^2}$) after the 24th period scaled by the reference value at the center of the source position ($\overline{E^2_{0air}}$) in air (Toifl et al., 2016b).

By analyzing figure 5.12 a significant deviation from an ideal Gaussian beam, as it would appear in a homogeneous material, is observed. Furthermore, a stripe-like pattern occurs which would not be visible in the homogeneous case. Since the imaginary part of the effective permittivity is considerable, the microwave beam does not penetrate deeply into the material. Figure 5.13 shows the difference between the inhomogeneous and homogeneous electric field at each point of the FDTD grid inside the material.



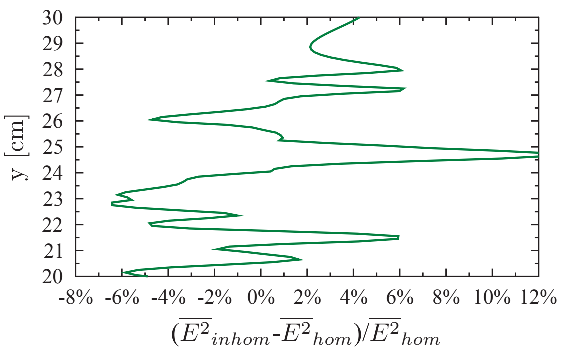


Fig. 5.13 Difference between the electric field of the inhomogeneous and homogeneous model. Red shaded areas indicate higher electric field values in the inhomogeneous model and blue lower values (Toifl et al., 2016b).

Fig. 5.14 Cut along the Gaussian axis (x = 25 cm, z = 25 cm) showing the relative difference in the electric field of the inhomogeneous to the homogeneous model in percent (Toifl et al., 2016b).

In figure 5.13 it can be seen that the electric field in the inhomogeneous model deviates quite significantly from the main beam in the homogeneous case. However, on average the values of $\overline{E^2}$ in the microstructure are slightly higher along the center of the Gaussian beam than in the homogeneous case as indicated by a cut along the middle axis of the Gaussian beam, "Gaussian axis" (in y-direction at x = z = 25 cm), see figure 5.14. A relative difference of up to 12% is observed. These deviations are caused by the differences of the real parts of the permittivity at the phase interfaces and by the non-zero imaginary part of phase A. From the results of the FDTD analysis, the absorbed power density P_{abs} distribution can be worked out (figure 5.15), which serves as input for the subsequent FE simulations.

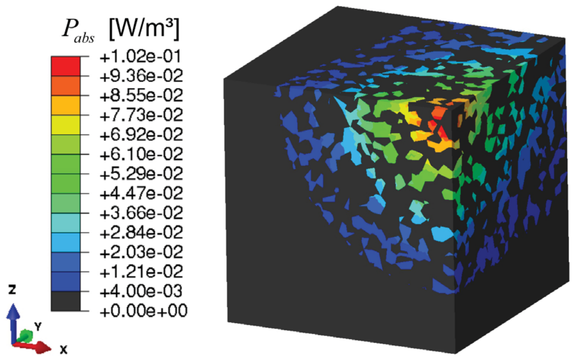


Fig. 5.15 Absorbed power density (P_{abs}) in $^{\mathrm{W}}/_{\mathrm{m}^3}$ (Toifl et al., 2016b).

According to the assumption that phase T is transparent for microwaves, absorption only occurs in phase A. As illustrated in figure 5.15, most of the energy is absorbed by few grains of phase A near the axis of the Gaussian beam.

5.1.2.2 Thermal results

Temperature field in the reference model

As described in section 4.1.3.5, the body heat flux, which is derived from the absorbed power density, is applied at the integration points of each finite element through the subroutine *DFLUX* (Abaqus, 2014). Heating the reference model with a microwave source of 25 kW and assumed losses of 30% for 15 s results in a considerably inhomogeneous temperature field (figure 5.16).

In figure 5.16 a maximum temperature of 547°C is observed in a depth of 1 cm along the axis of the Gaussian beam inside a phase A grain. Moreover, strong selective heating due to the selective absorption (cf. figure 5.15) combined with the variation of the thermal properties of the two constituents is obtained. The strong localization of areas with high temperatures leads to significant thermal gradients between the two material phases and, therefore, high thermal stresses are expected. However, the maximum temperature in phase T

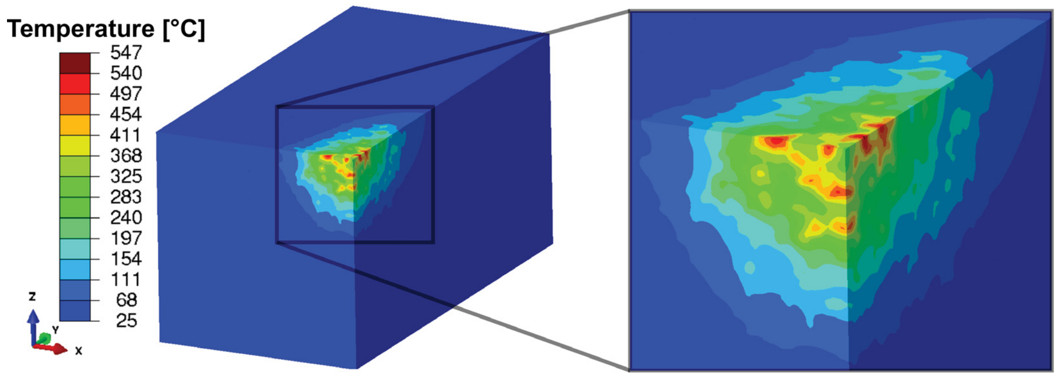


Fig. 5.16 Temperature field in °C after 15 s of microwave irradiation (25 kW minus 30% losses) of the reference morphology (Toifl et al., 2016b).

(quartz) is 482°C, which is significantly below the α to β phase transformation temperature of 573°C.

Temperature field in the case of a phase transformation

Since no α to β phase transformation in quartz has been observed in the reference model so far, a new model with the same morphology but 25 s irradiation time is considered. In figure 5.17 the temperature distribution of the 25 s model is compared to the one after 15 s of microwave irradiation.

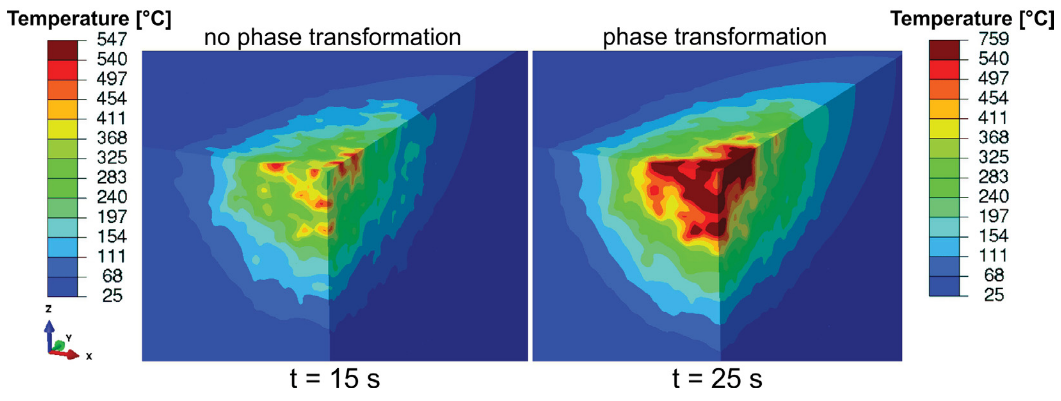


Fig. 5.17 Comparison of the temperature distribution in °C between 15 s and 25 s (incl. phase transformation) irradiation time (Toifl et al., 2016b).

By comparing the same microstructure model with 15 s and 25 s of irradiation time, respectively, significantly higher temperatures in the 25 s model (maximum of 759°C instead of 547°C) are observed (figure 5.17). Furthermore, the depth and the radius of the area above 70°C are larger due to the longer irradiation time. After 25 s of microwave heating a maximum temperature of 693°C is observed inside phase T (quartz, figure 5.18). As a consequence the α quartz transforms to β quartz in areas near the hottest phase A grains around the Gaussian beam axis. For better visibility the phase A elements have been switched off in figure 5.18.

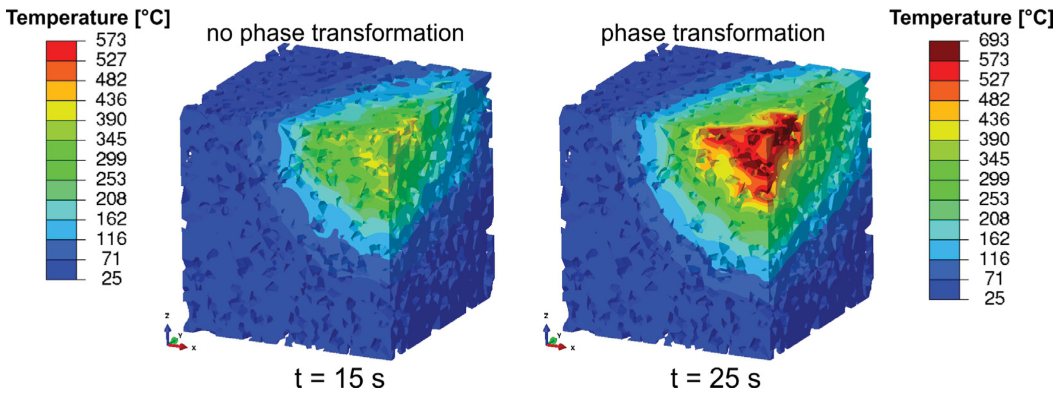


Fig. 5.18 Comparison of the phase T (quartz) temperatures in °C between the model of 15 s and 25 s microwave irradiation. The phase A elements are not displayed (Toifl et al., 2016b).

Temperature field for different morphologies

The dependence of the resulting microwave induced temperature field on the morphology is assessed in figure 5.19. Model A corresponds to the reference model investigated in figure 5.16.

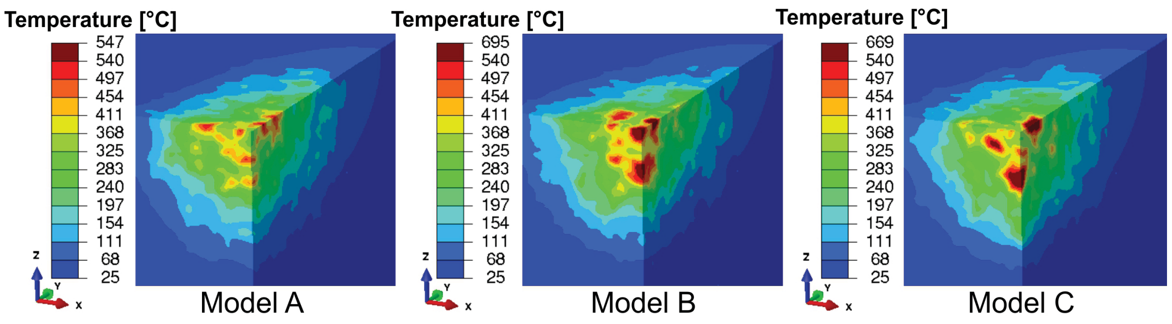


Fig. 5.19 Comparison of the temperature fields in °C between different morphologies.

Although all of the three models contain the same filling factor $f = 0.34$, significant variations in the thermal fields are obtained (figure 5.19). The highest temperature of 695°C is observed in model B in a huge phase A grain which is directly located at the Gaussian axis at the front surface. The peak temperatures differ by as much as 150°C between the three investigated morphologies.

5.1.2.3 Linear elastic stress results

Using the transient temperature field (obtained in section 5.1.2.2) as an input as well as the boundary conditions described in section 4.1.3.5, a stress analysis is conducted assuming an elastic material behavior of the two constituents. Since hard rocks can be considered as quasi-brittle, the maximum principal stress is an appropriate measure for assessing damage initiation. However, crushing due to compressive stresses as well as ductile failure is also assessed in the reference model.

Stress field of the reference model

High maximum principal stresses are observed in the reference model in phase A grains near the front face of the rock model inside the main Gaussian beam (figure 5.20). In this model the stresses are significantly higher than the tensile strength of 9 MPa and, therefore, damage initiation is expected. The tensile strength of 9 MPa is exceeded down to a depth of 10 cm (in direction of the microwave propagation, light blue areas in figure 5.20) indicating that initial surface cracks are likely to propagate in depth direction. After 4.2 s of microwave irradiation the maximum principal stresses at the phase boundaries and in small areas around the main irradiated spot already reach the material strength.

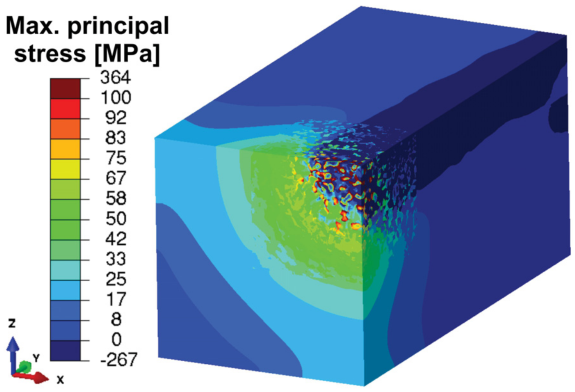


Fig. 5.20 Maximum principal stresses in MPa after 15 s of microwave heating of the inhomogeneous reference model (Toifl et al., 2016b).

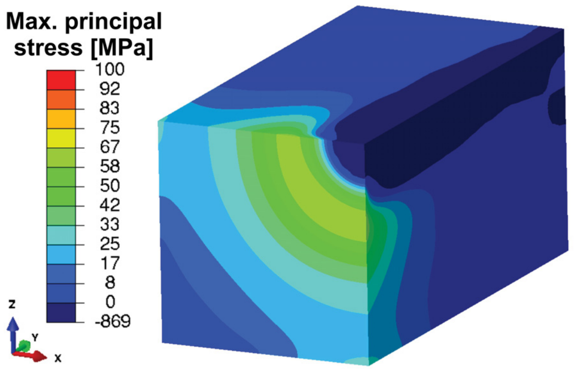


Fig. 5.21 Maximum principal stresses in MPa after 15 s of microwave heating of the homogeneous rock model (Toifl et al., 2016b).

In order to investigate the influence of the microstructure on the formation of the stress distribution, a comparative analysis including a homogeneous material definition is performed. By comparing figure 5.21 with the reference microstructure model (figure 5.20) a significant difference in the distribution of the stresses is found. In the homogeneous case the highest principal stresses are observed around the main Gaussian beam (figure 5.21), whereas substantially higher stresses are found inside the main beam in phase A grains of the inhomogeneous model (figure 5.20). For the inhomogeneous case a submodel with only a few grains around the Gaussian axis at the front face (radius $\approx$ 3 cm) including elements with quadratic shape functions is built in order to assess the stress localization in more detail (figure 5.22). The submodel is stressed by the global displacements (at the interfaces between submodel and global model) and heated according to the transient temperature field.

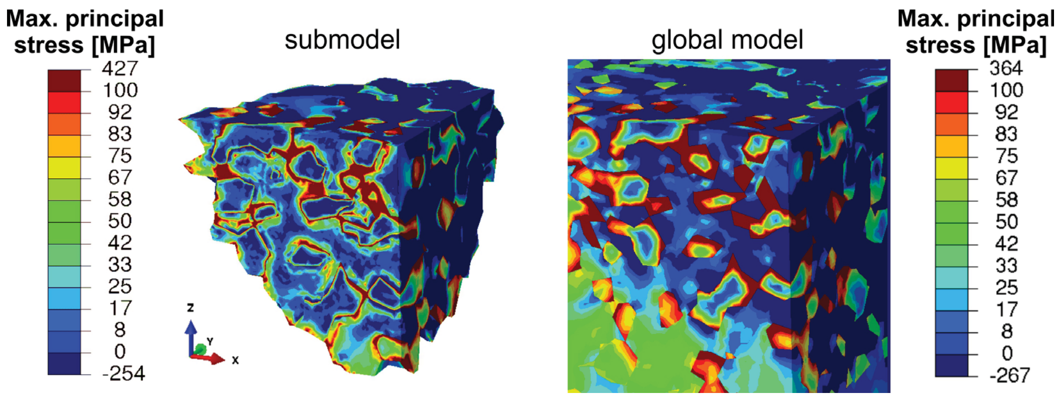


Fig. 5.22 Maximum principal stresses in MPa after 15 s of microwave heating of the inhomogeneous reference model near the Gaussian axis.

As can be seen in figure 5.22, the highest maximum principal stresses are strongly localized at the phase boundaries. Moreover, very high stresses occur if strongly absorbing phases (phase A) are close to each other. Then the transparent phase is under strong tension as well. Additionally, higher maximum principal stresses than in the global model are observed in the submodel.

In addition to brittle failure under tension crushing due to compressive stresses has to be investigated. By comparing the minimum principal stresses (figure 5.23) with the uniaxial compressive strength of hard rocks, which is about 250 MPa (Hustrulid et al., 2001), the areas of damage due to compression can be identified.

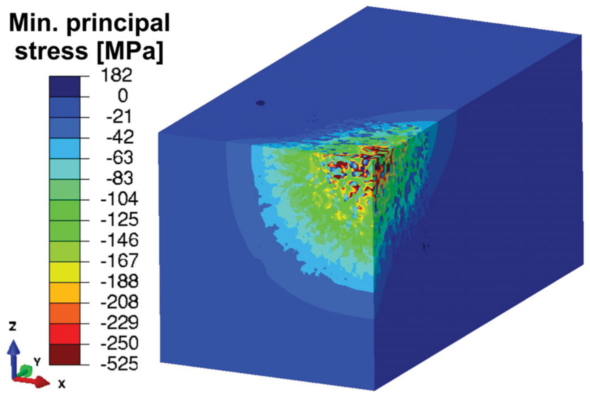


Fig. 5.23 Minimum principal stresses in MPa after 15 s of microwave heating of the reference morphology (Toifl et al., 2016b).

Figure 5.23 shows that the minimum principal stresses exceed the material limit in the vicinity of the Gaussian axis in phase T grains near their phase boundaries. In this area crushing is expected.

The maximum principal stress is an appropriate measure as long as brittle behavior of the constituents is assumed. However, it has to be pointed out that in realistic rock materials different damage mechanisms might occur especially at elevated temperatures where non-negligible amounts of plasticity are expected. In that case an alternative failure criterion would allow more accurate predictions of the failure behavior of the rock. This is demonstrated on the example of the Tresca stress distribution function displayed in figure 5.24.

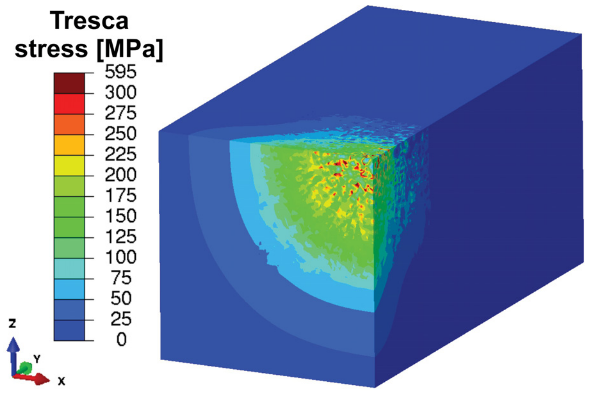


Fig. 5.24 Tresca stresses in MPa after 15 s of microwave heating of the reference morphology (Toifl et al., 2016b).

The Tresca stresses exceed the yield strength in a wide area around the main irradiation spot as shown in figure 5.24. The highest values are located near the phase boundaries in both constituents close to the Gaussian axis. After 15 s of microwave irradiation plastic deformation is expected if ductile material behavior is assumed.

Stress results for different morphologies

The maximum principal stresses of two different morphologies are compared with the reference model (figure 5.27 first line). Since higher maximum temperatures are observed in model B and C (figure 5.19), higher maximum principal stresses can be expected. Comparing the maximum principal stresses along the Gaussian axis of the three different morphologies with the homogenous model allows to estimate the effects of the different phase assignments (figure 5.25). Moreover, the temperature distribution (figure 5.26) can be correlated to the stress formation (figure 5.25).

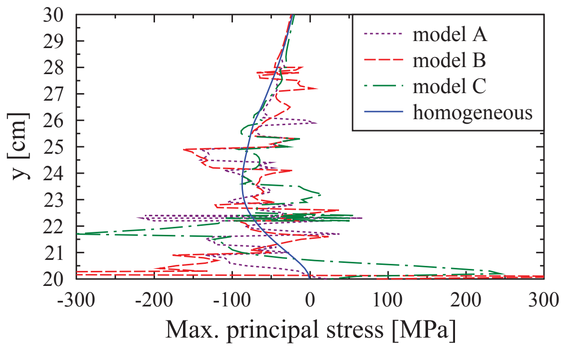


Fig. 5.25 Maximum principal stress profile along the y-direction of the Gaussian beam (Toifl et al., 2016b).

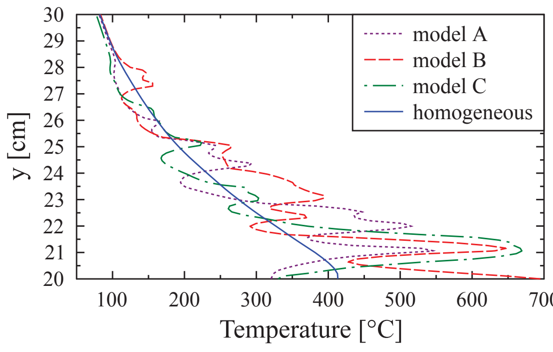


Fig. 5.26 Temperature profile along the y-direction of the Gaussian beam (Toifl et al., 2016b).

Considerable variations in the maximum principal stresses between the three models are observed (figure 5.25). Especially within the first centimeters of the material a strong deviation from the homogeneous model is determined. In model B and C significantly higher stresses than in the reference model (model A) are obtained (cf. figure 5.27 first line). The influence of the microstructure on the selective heating of a rock and the resulting formation of stresses due to microwave absorption is analyzed by illustrating the temperature profile along the Gaussian axis (figure 5.26).

As can be seen in figure 5.26 the temperatures of the inhomogeneous material models strongly deviates from the homogeneous one. Furthermore, strong thermal gradients appear, thus causing high thermal stresses. The largest deviation from the reference model A is observed in model B where the temperature near the surface is very high. Moreover, model C yields higher temperatures than model A.

Stress results for different phase distributions

In addition to the variation in the morphologies different filling factors f (volume fraction of absorbing phase) are investigated. In figure 5.27 three different filling factors ($f = 0.34$, $f = 0.4$, $f = 0.2$) and varying morphologies are visualized.

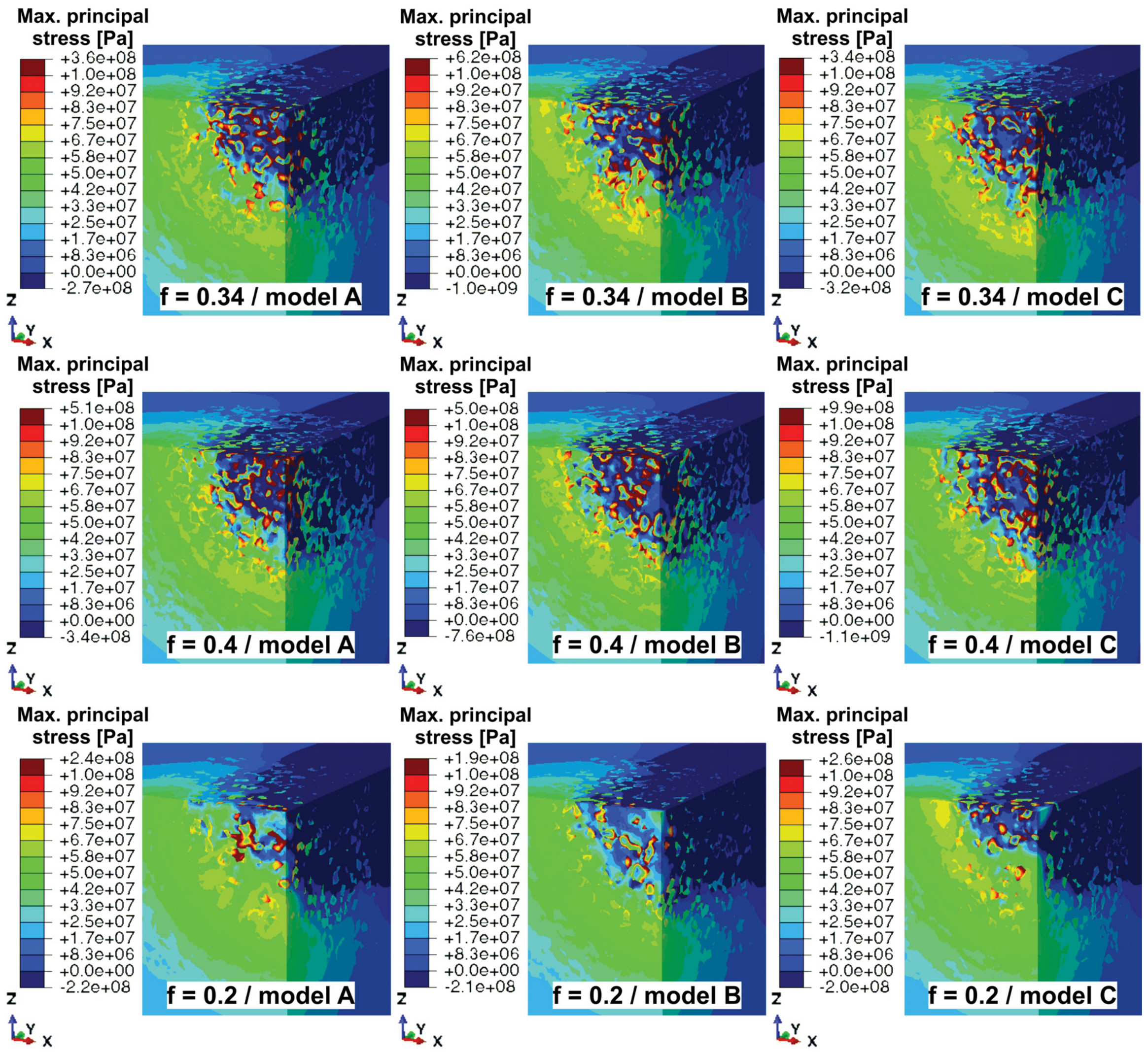


Fig. 5.27 Comparison of the maximum principal stresses in Pa between different phase distributions and morphologies.

The maximum principal stress field varies strongly between the different filling factors. In the $f = 0.4$ case larger maximum principal stresses and also a broader region of high stresses are observed compared to the $f = 0.34$ models. If fewer absorbing particles are available ($f = 0.2$), significantly lower stresses are found. They are concentrated at the grain boundaries of some phase A grains. In order to investigate the differences in the maximum principal stress fields in more detail, the volume fraction of elements in the microstructure containing stresses higher than the tensile strength (9 MPa) is plotted for all nine models (figure 5.28).

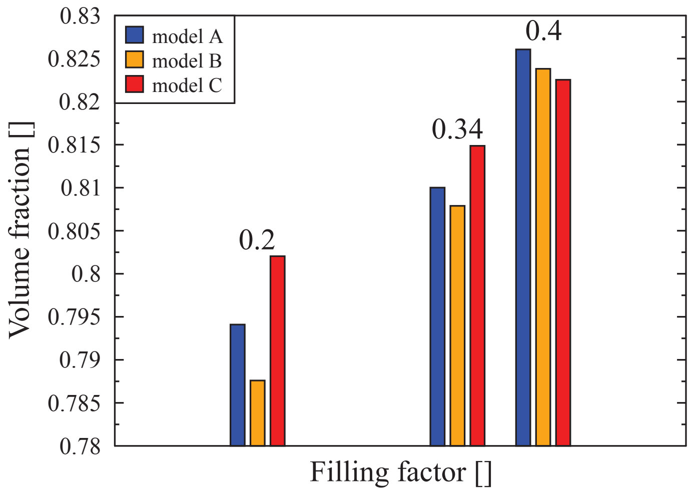


Fig. 5.28 Volume fraction of elements in the microstructure which contains maximum principal stresses higher than the tensile strength (9 MPa).

In figure 5.28 the nine models are grouped according to their filling factor (first triple belongs to $f = 0.2$, the second to $f = 0.34$ and the third to $f = 0.4$). The fraction containing stresses higher than the tensile strength in the microstructure cube increases with higher values of the filling factor. Additionally, the deviation between the different morphologies with the same filling factor is larger for $f = 0.2$ than for $f = 0.4$.

Stress field after phase transformation

The influence of longer microwave irradiation times followed by α to β phase transformation of quartz is assessed by comparing the reference model with 15 s microwave treatment with a model with 25 s of microwave exposure (figure 5.29). Significantly higher maximum principal stresses are observed in phase A grains near the Gaussian axis after 25 s of irradiation (cf. figure 5.29). Moreover, a larger area of high principal stress appears (t = 25 s). Also higher compressive stresses are formed in phase T as a consequence of the higher tensile stresses in phase A.

In order to investigate the influence of phase transformation and longer irradiation time in detail, a statistical analysis of the stresses at the integration points of the finite elements located in the microstructure cube is performed. For this purpose the total stress range is

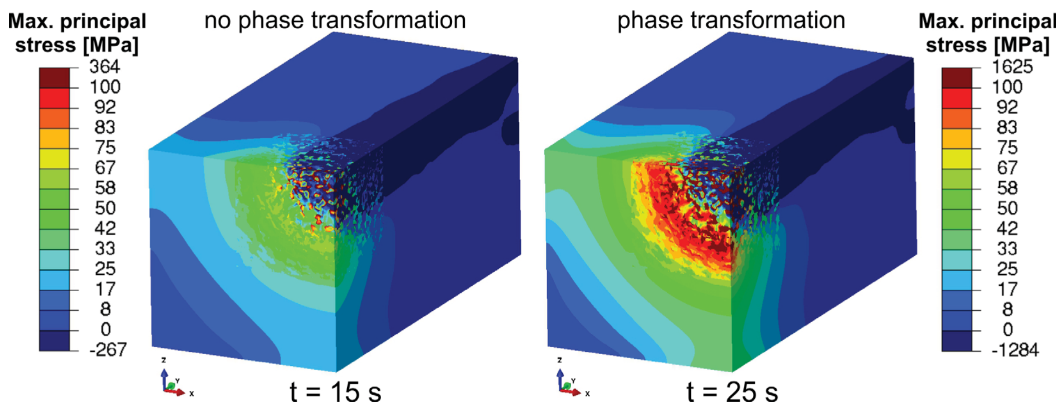


Fig. 5.29 Comparison of the maximum principal stresses in MPa between 15 s and 25 s (incl. phase transformation) irradiation time (Toifl et al., 2016b).

divided into classes with a width of 1 MPa. A *Python* script loops over all integration points and incrementally increases the frequency density, expressed by the integration volume divided by the total volume of the corresponding phase, of the class which the respective stress value belongs to. In figure 5.30 the frequency density distribution is plotted for the cases of 15 s as well as 25 s irradiation time, separately evaluated for both phases. The function values of the graph are found by dividing the accumulated volume fraction in each class by the width of the class (Steland, 2013).

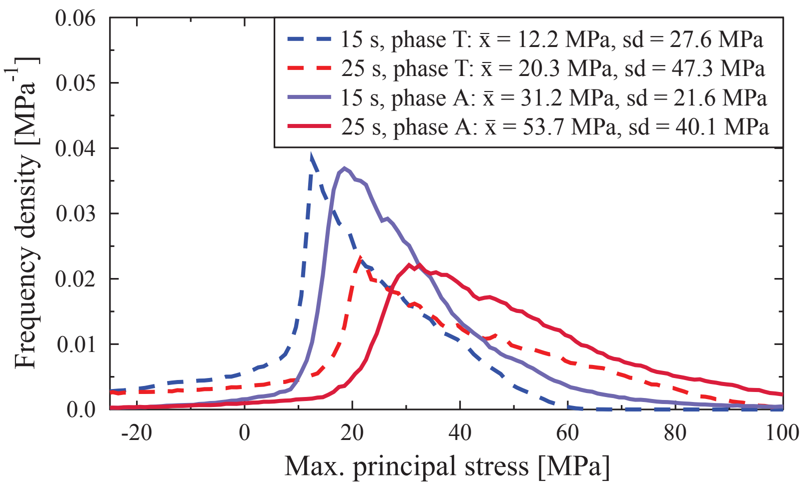


Fig. 5.30 Frequency density of the maximum principal stresses of phase T and phase A scaled to the respective volumes corresponding to the stress values. The legend includes the arithmetic mean $\bar{x}$ as well as the standard deviation *sd* (Toifl et al., 2016b).

In figure 5.30 a strong difference in the distribution of the maximum principal stresses between the model of 15 s and 25 s of microwave irradiation can be seen. In the 25 s case a significant shift of the frequency distribution to higher stresses for phase T (arithmetic mean of 20.3 MPa) as well as phase A ($\bar{x}$ = 53.7 MPa) compared to the 15 s model (phase T: $\bar{x}$ = 12.2 MPa, phase A: $\bar{x}$ = 31.2 MPa) are observed. Moreover, a larger deviation occurs between the phase T and phase A stress distribution in the model including phase transformation (25 s). However, the density distributions of the maximum principal stresses after 25 s of microwave irradiation are significantly broader (phase T: *sd* = 47.3 MPa, phase A: *sd* = 40.1 MPa) than in the 15 s case (phase T: *sd* = 27.6 MPa, phase A: *sd* = 21.6 MPa). Here *sd* denotes the standard deviation.

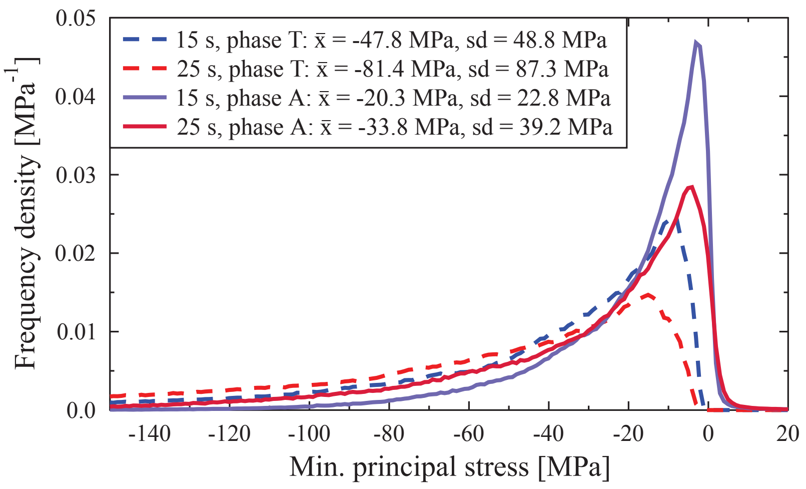


Fig. 5.31 Frequency density of the minimum principal stresses of phase T and phase A scaled to the respective volumes corresponding to the stress values (Toifl et al., 2016b).

By visualizing the frequency density distribution of the minimum principal stresses (figure 5.31), a significant difference in the shape of the graphs compared to the maximum principal stresses (figure 5.30) is observed. The larger part of phase A exhibits stresses close to zero minimum principal stresses. Phase T contains significantly more volume under high compressive stresses than phase A. Similar to the maximum principal stresses (figure 5.30) the curves are shifted to higher stresses in the 25 s irradiation case.

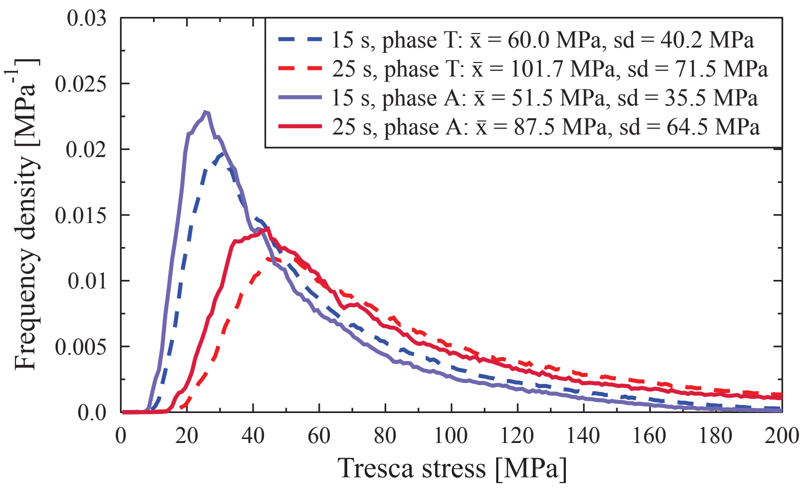


Fig. 5.32 Frequency density of the Tresca stresses of phase T and phase A scaled to the respective volumes corresponding to the stress values (Toifl et al., 2016b).

At higher stress levels the Tresca stress distribution of both constituents is almost coincident (figure 5.32). This is due to the fact that very high Tresca stresses are observed in phase T as well as in phase A near their boundary phases. A significant shift to higher stresses is observed after 25 s of microwave irradiation.

Influence of anisotropic material behavior

The influence of the anisotropic behavior of the quartz grains (phase T) after 25 s of microwave heating of the reference model is investigated. To this end the stress distribution in the microstructure cube is calculated for both the isotropic as well as the anisotropic case (figure 5.33).

In figure 5.33 higher maximum principal stresses are observed in the anisotropic model in phase A grains over a wider range than in the isotropic model. Furthermore, due to the anisotropic behavior of the quartz grains tensile stresses occur in phase T (quartz) near the Gaussian beam that do not appear in the isotropic case. Especially near the Gaussian axis the quartz grains are instantly changing their elastic constants due to phase transformation. In addition, higher temperatures are obtained in the anisotropic model which lead to higher thermal stresses. To investigate these differences in more detail a statistical analysis, as outlined in figures 5.30 - 5.32, is performed and the difference between the anisotropic and isotropic model is displayed (figure 5.34).

Figure 5.34 reveals lower compressive stresses in phase T (quartz) and therefore higher tensile stresses up to a level of 20 MPa compared to the isotropic case. Furthermore, according

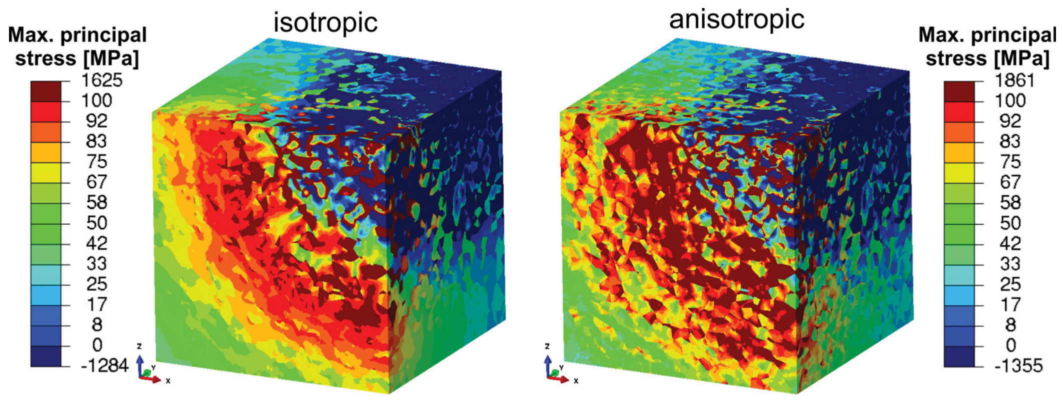


Fig. 5.33 Comparison of maximum principal stresses in MPa between the isotropic and anisotropic model after 25 s of microwave heating. Only the cube containing the microstructure is visualized (Toifl et al., 2016b).

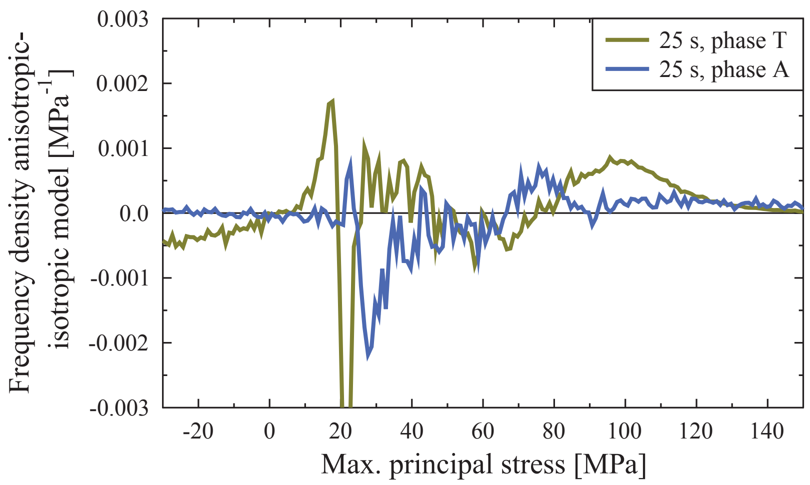


Fig. 5.34 Difference of the frequency density of the maximum principal stresses between the anisotropic and the isotropic model after 25 s of microwave irradiation (Toifl et al., 2016b).

to the anisotropic model in phase T significantly larger areas are subjected to stresses greater than 80 MPa. Moreover, a slightly higher fraction of phase A containing tensile stresses greater than 75 MPa is predicted by the model with anisotropic quartz behavior.

5.1.2.4 Damage results

In the 3D two component concrete damaged plasticity model the same CDP parameters as in the 2D case are used (cf. table 4.3). In order to reduce the computational cost, the CDP model is first only applied in the microstructure cube and the remaining bulk material is modeled as linear elastic. In figure 5.35 the stiffness degradation variable d is displayed at different irradiation times of the reference inhomogeneous model during a microwave irradiation of 15 s. In the 3D two component model the CDP material model only considers damage under tension.

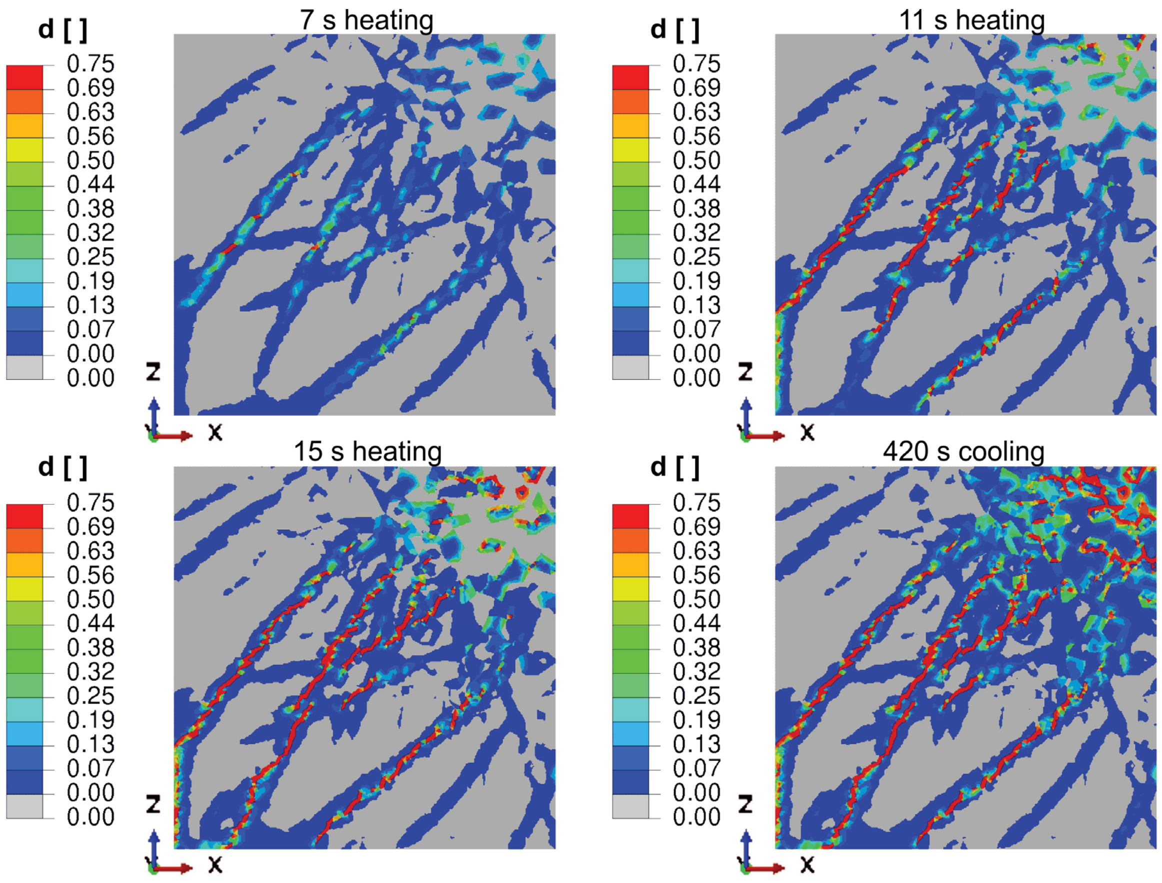


Fig. 5.35 Scalar stiffness degradation variable d in the reference model at different times during a microwave irradiation of 15 s.

During the first seconds of microwave irradiation damage and subcritical cracks (blue areas in figure 5.35) are initiated around phase boundaries and in the vicinity of the main heated area. Later, critical cracks (red in figure 5.35) propagate outside radially. After 15 s of heating also fully damaged material points are observed at the phase boundaries near the Gaussian axis. During cooling a strong damage formation is observed in these areas. The damaged material points ($d \geq 0.6$) are plotted in figure 5.36 and are compared with those of a homogeneous model.

Figure 5.36 shows a strong difference between the damage behavior of the inhomogeneous versus the homogeneous case. In the homogeneous models damage is determined only around the main heated area, whereas in the microstructure model damage can also be seen near the Gaussian beam. However, the damage around the middle axis does not penetrate deep into

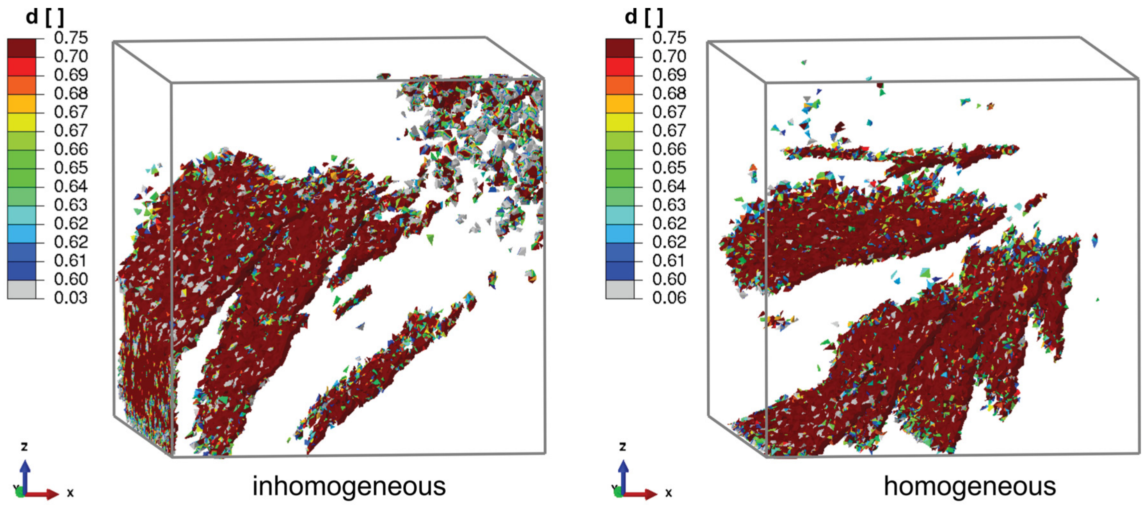


Fig. 5.36 Finite elements with $d \geq 0.6$ after 15 s of microwave irradiation of the inhomogeneous model compared with the homogeneous one. Microwaves propagate along positive y-axis.

the material (figure 5.36). By correlating the phase distribution of the inhomogeneous model with the elements containing $d \geq 0.6$ (figure 5.37), further conclusions can be drawn.

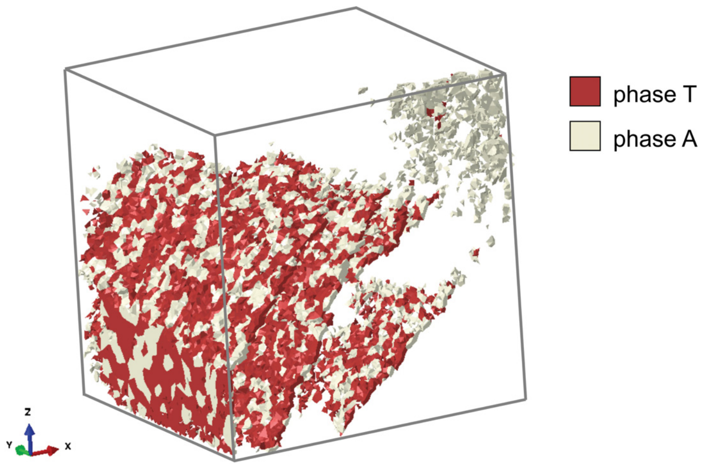


Fig. 5.37 Finite elements with $d \geq 0.6$ after 15 s of microwave irradiation of the inhomogeneous model including the phase distribution.

Damage near the Gaussian axis in the inhomogeneous model occurs almost exclusively in the strongly absorbing phase A. In the area around the main heated rock the damage is determined in both constituents. The formation of damage on the microstructure scale can be assessed in more detail by visualizing the volume fraction of elements (within the microstructure) with scalar degradation values $d \geq 0.75$ as a function of time for both constituents (figures 5.38 and 5.39).

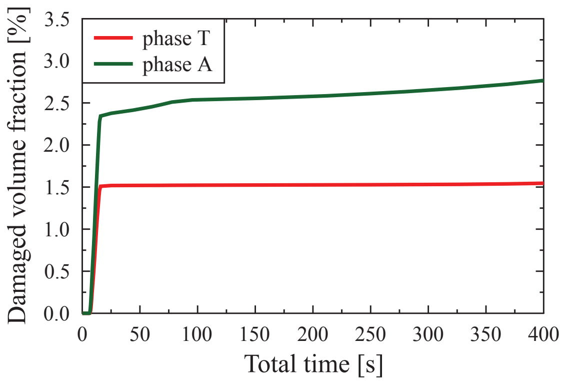


Fig. 5.38 Formation of volume fraction (within the microstructure) of damaged material points $d \geq 0.75$ over the total time (heating plus cooling) for both constituents.

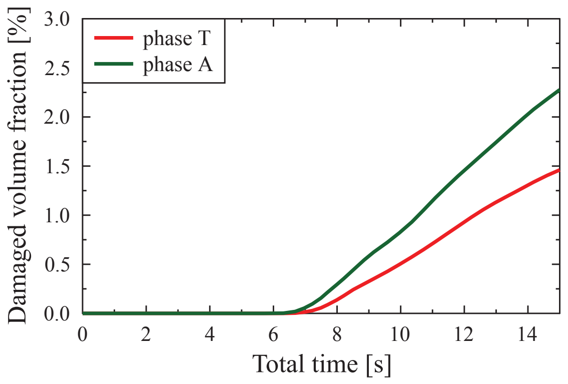


Fig. 5.39 Formation of volume fraction (within the microstructure) of damaged material points $d \geq 0.75$ over the heating time for both constituents.

In figures 5.38 and 5.39 it is observed that the highly absorbing phase A is more severely damaged over the whole irradiation and cooling time. Damage initiation starts after an irradiation duration of about 7 s (where a maximum temperature of 340°C is observed). During cooling a slight increase in damaged finite elements is observed in figure 5.38. The CDP material model is also applied to the 25 s irradiation model which includes the α to β phase transformation (figure 5.40).

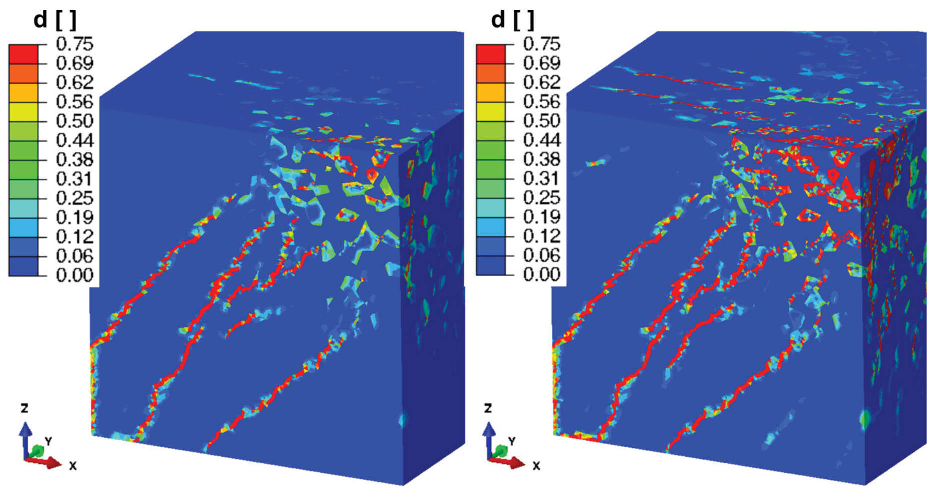


Fig. 5.40 Scalar stiffness degradation variable d in the reference model after 15 s and 25 s (α to β phase transformation) of microwave irradiation.

Due to the longer microwave irradiation time combined with the α to β phase transformation significantly more damage is observed after 25 s of microwave irradiation than in the 15 s case (figure 5.40). Moreover, the damage penetrates deeper into the depth of the material near the Gaussian axis. This tendency becomes especially apparent if only the elements with $d \geq 0.7$ are plotted (figure 5.41).

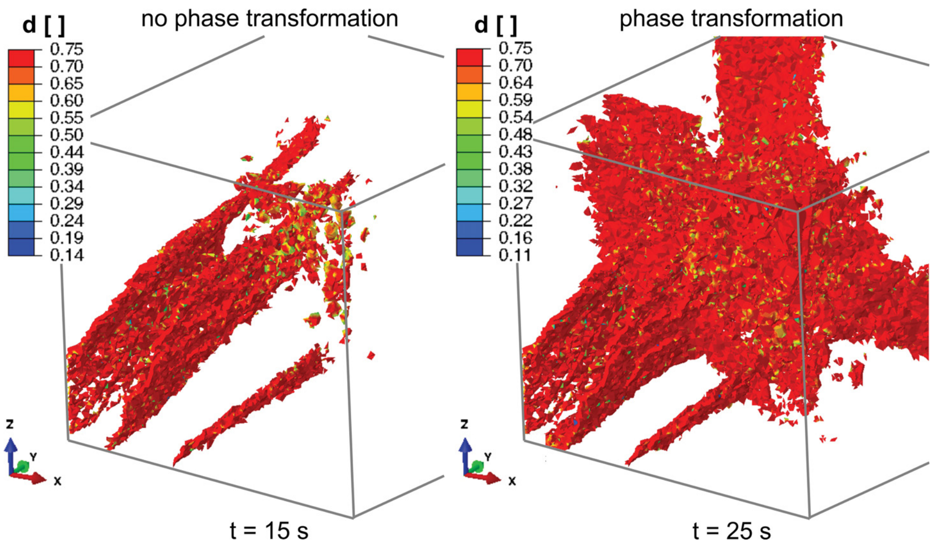


Fig. 5.41 Finite elements with $d \geq 0.7$ after 15 s and 25 s (α to β phase transformation) of microwave irradiation of the inhomogeneous model.

In addition to the damaged elements after 15 s of irradiation two major disc shaped radial damage patterns are observed in the 25 s case. Also more elements near the main Gaussian axis are damaged. The global damage behavior of the two models (15 s and 25 s) is assessed in a statistical analysis (figure 5.42).

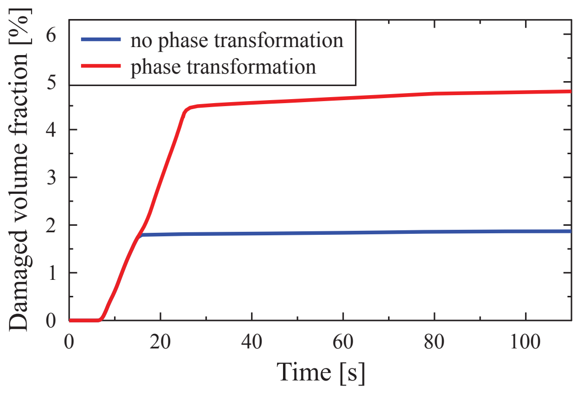


Fig. 5.42 Volume fraction of microstructure elements with scalar degradation values $d \geq 0.75$ over irradiation times for 15 s and 25 s (α to β phase transformation).

Overproportional more volume contains damage values $d \geq 0.75$ in the 25 s irradiation case than in the 15 s model (figure 5.42). This is caused by the additional volumetric strain of the α to β phase transformation in the 25 s case. In all of the investigated 3D two component

CDP models the damage propagates in radial direction outward until the microstructure / bulk material interface is reached where it is obviously deflected. In order to investigate the effect of these boundaries, an extended model (25 x 30 x 25 cm^3 compared to 15 x 30 x 15 cm^3) is created and the CDP material definition is extended to the bulk (figure 5.43). In order to preclude any artifacts originating from the tie constrains, that are necessary to connect two differently meshed regions, a new meshing strategy has been pursued. The two regions are now connected by a transition zone of gradually increasing tetrahedral elements without the need of any tie constraint.

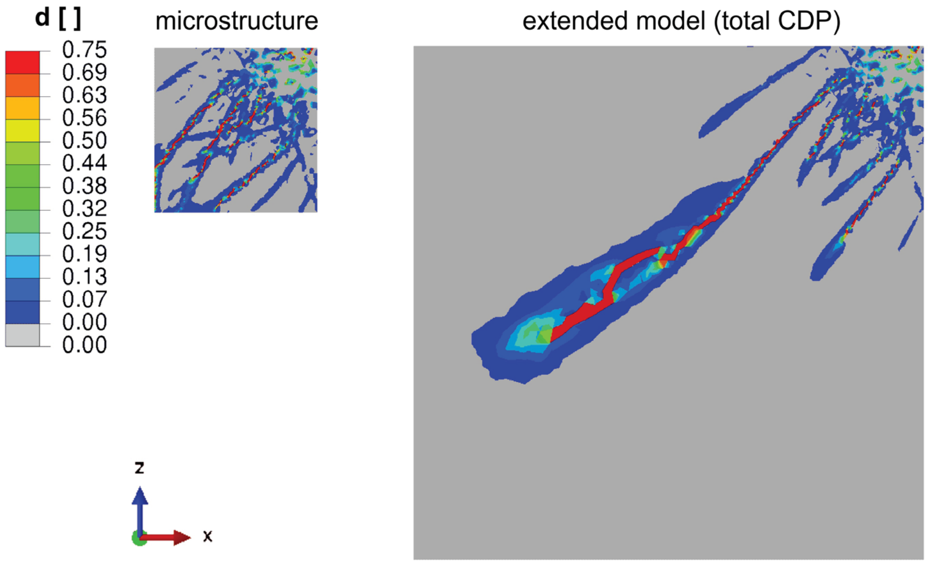


Fig. 5.43 Scalar stiffness degradation variable d in an extended model including CDP in the bulk material area compared to the reference model after 12 s of microwave irradiation.

In figure 5.43 damage clearly propagates outside the main irradiated area. By contrast, the remaining damage pattern is slightly reduced compared to the reference model.

5.1.3 3D model of the three component system

With the 3D three component model varying parameter analyses applied to a specific granite block are conducted (cf. section 4.1.3). Preliminary results of the 3D three component model have been published in Toifl et al. (2016a).

5.1.3.1 Electromagnetic results

The obtained electromagnetic field is the basic input for all following thermal field calculations. In figure 5.44 the time averaged squared electric field ($\overline{E^2}$) scaled with the reference value at the center of the source position in air ($\overline{E^2_{0air}}$) is illustrated after the 24th period, when the electric field has stabilized.

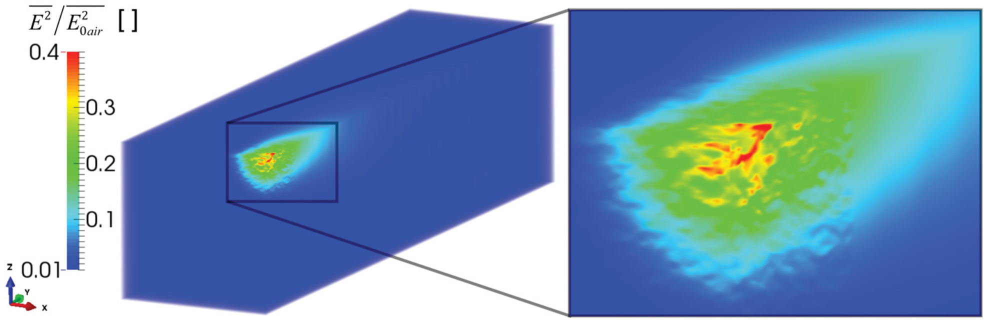


Fig. 5.44 Time averaged squared electric field ($\overline{E^2}$) after the 24th period scaled with the reference value at the center of the source position in air ($\overline{E^2_{0air}}$) for the three component granite model (Toifl et al., 2016a).

A significant deviation from an ideal Gaussian beam is observed in figure 5.44. Moreover, a stripe-like pattern occurs which is not visible in a homogeneous case. Since the permittivity of the granite is relatively small, the microwave beam penetrates significantly into the material. In figure 5.45 the difference between the inhomogeneous and homogeneous electric field is depicted at each point of the FDTD grid inside the microstructure cube.

Figure 5.45 reveals that the electric field in the inhomogeneous model deviates strongly from the main beam in the homogeneous case. Especially some small areas with significantly varying electric field values can be observed (dark red and dark blue in figure 5.45). A similar behavior is obtained in figure 5.46 where the relative difference of the time averaged squared electric field between the inhomogeneous and a homogeneous model is plotted. A maximum difference of up to 40% is visible. These strong deviations are mainly caused by the large differences in the real parts of the permittivity, especially between quartz ($\epsilon'_{r,q} = 4.28$) and plagioclase ($\epsilon'_{r,p} = 6.57$).

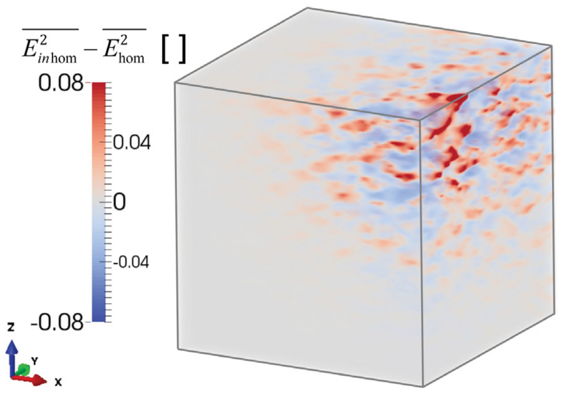


Fig. 5.45 Difference between the electric field of the inhomogeneous and the homogeneous model in the three component granite. Red shaded areas indicate higher electric field values in the inhomogeneous model and blue lower values (Toifl et al., 2016a).

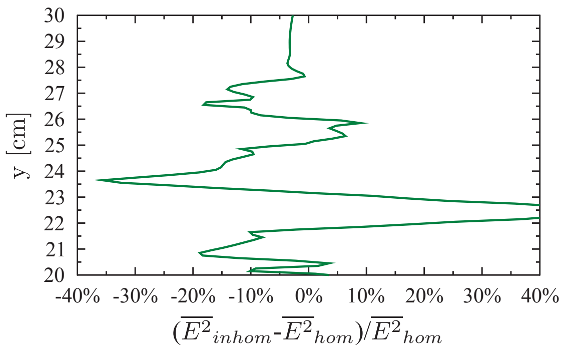


Fig. 5.46 Cut along the Gaussian axis (x = 30 cm, z = 30 cm) showing the relative difference of the inhomogeneous to the homogeneous model in three component granite (Toifl et al., 2016a).

From the resulting electric field inside the rock model, the absorbed power density distribution can be worked out (figure 5.47), which serves as an input for the subsequent FE simulations. Since the imaginary part of the dielectric constant of quartz and mica is nearly zero ($\epsilon''_{r,q} = \epsilon''_{r,m} = 0.0005$), most of the microwave power is absorbed in the plagioclase phase. Very high values of the absorbed power density are obtained at the boundaries of plagioclase grains near the axis of the Gaussian beam (figure 5.47).

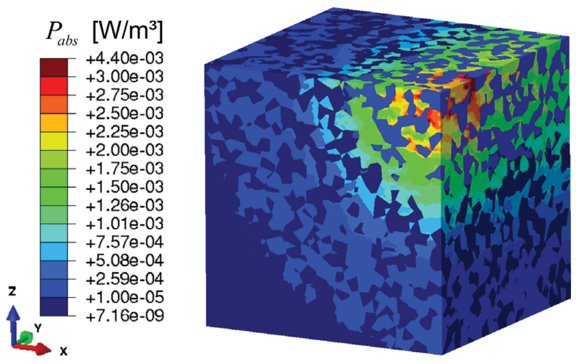


Fig. 5.47 Absorbed power density (P_{abs}) in W/m³ of the microstructure cube of the three component granite model (Toifl et al., 2016a).

5.1.3.2 Thermal results

Based on the four different irradiation times combined with two power levels summarized in table 4.6 the thermal fields are evaluated. The temperature fields at the end of the microwave treatment with 25 kW (minus 30% of losses) are summarized in figure 5.48.

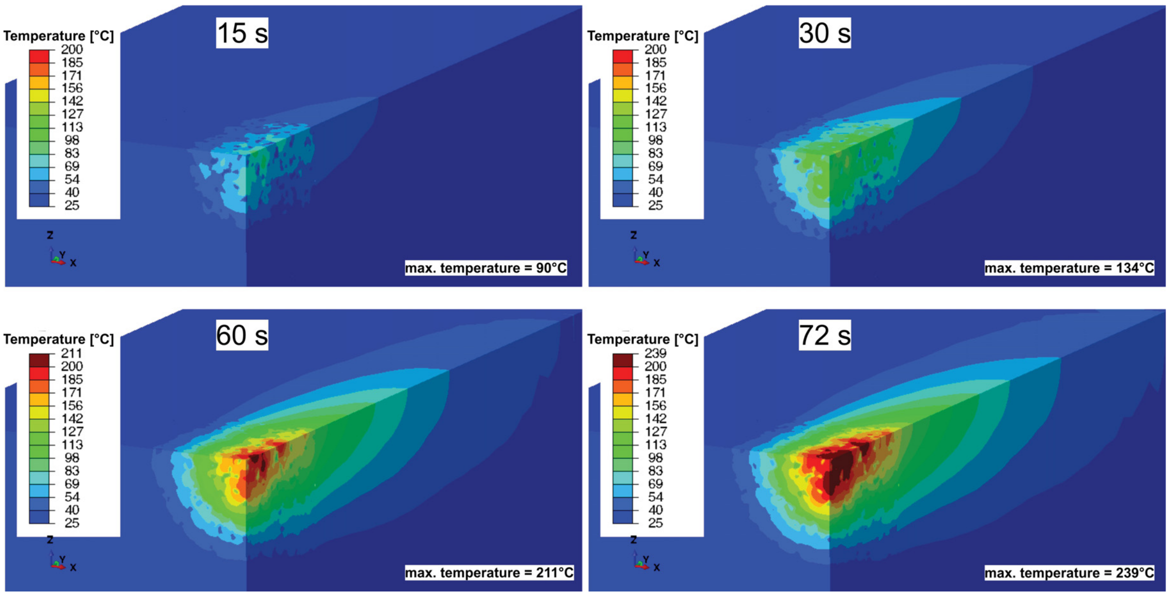


Fig. 5.48 Temperature field in °C after microwave irradiation with 25 kW (minus 30% losses) of granite models for durations given in the figure (Toifl et al., 2016a).

Strong variations in the temperature fields between the different microwave irradiation times are observed in figure 5.48. After 15 s of irradiation a maximum temperature of only 90°C is reached, whereas after 72 s a maximum temperature of 239°C is determined. The maximum temperature of each model is observed in a depth of about 1 cm. Moreover, the heated area also extends rapidly with increasing microwave treatment durations. The volume fraction subjected to temperatures higher than 40°C (light blue area in figure 5.48) grows from 0.40% in the 15 s case to 4.01% after 72 s of irradiation. Strong localized heating is obtained due to the selective absorption (cf. figure 5.47) combined with the variation of the thermal properties of the three constituents. However, with increasing irradiation times the temperature field becomes homogenized due to the heat conduction, which smoothens the thermal gradients with longer irradiation times.

By irradiating the granite models with a microwave source of 30 kW (minus 30% losses) almost the same conclusion as in the 25 kW case can be drawn (figure 5.49). A maximum temperature of 103°C is determined after 15 s of irradiation whereas after 72 s 279°C is reached. The absolute values of the temperatures are higher than in the 25 kW case

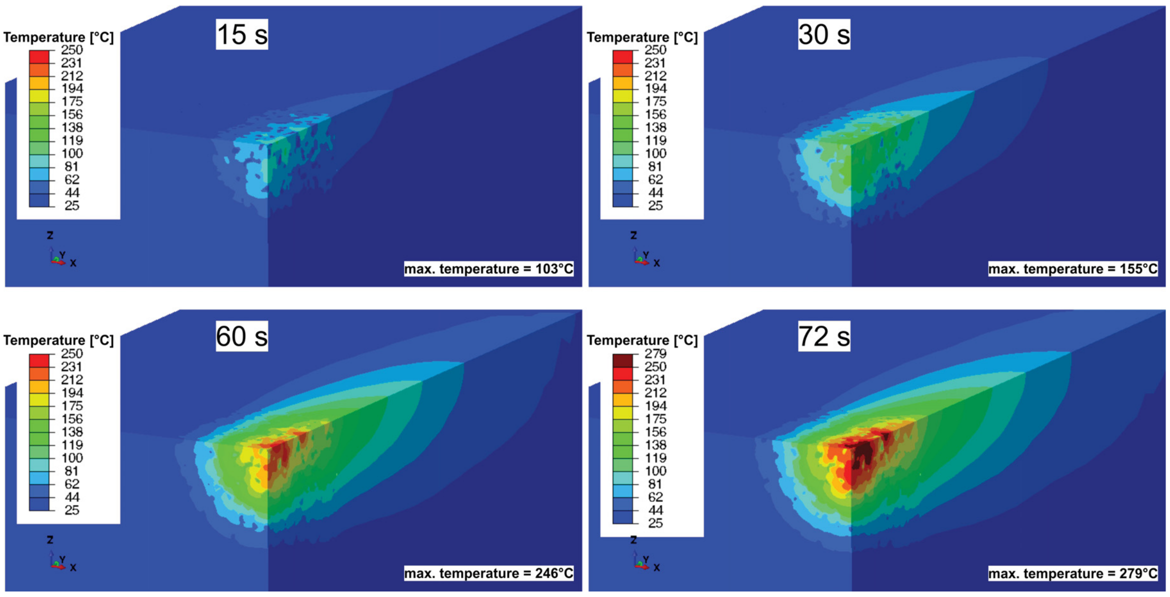


Fig. 5.49 Temperature field in °C after microwave irradiation with 30 kW (minus 30% losses) of granite models for durations given in the figure (Toifl et al., 2016a).

(figure 5.48). However, in none of the observed models the α to β phase transformation temperature of 573°C is reached. A volume fraction exposed to temperatures above 40°C of 0.54% is determined after 15 s microwave treatment and 5.11% after 72 s. These values are also higher than in the models with the 25 kW source (figure 5.48).

In order to investigate these variations in the temperature fields in greater detail, a statistical analysis of the temperatures at the integration points of the finite elements located in the microstructure cube is performed. For this purpose the total temperature range is divided into classes with a width of 5°C (cf. statistical analysis in section 5.1.2.3). In figure 5.50 the frequency density distribution is plotted for all investigated microwave irradiation parameters. The function values of the graphs are derived by dividing the volume fraction in each class by the width (5°C) of the class (Steland, 2013).

As expected, in figure 5.50 higher temperatures are observed with increasing microwave irradiation times and powers. The arithmetic mean values of the frequency distribution are shifted to higher temperatures due to a larger amount of provided microwave energy (15 s, 25 kW: $\bar{x}$ = 33.21°C compared to 72 s, 25 kW: $\bar{x}$ = 59.41°C). Moreover, the distribution of the temperature gets broader with increasing power levels and irradiation times (15 s, 25 kW: sd = 9.66°C compared to 72 s, 25 kW: sd = 41.94°C). The shift between the frequency densities of the 25 kW to the 30 kW case becomes more significant with increasing irradiation durations (figure 5.50). This is due to the fact that the difference of the provided microwave

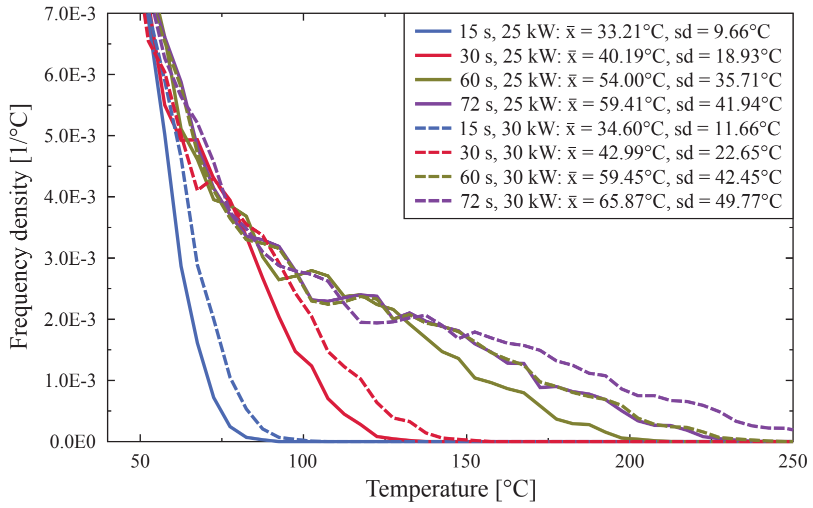


Fig. 5.50 Frequency density of the temperature of the microstructure cube scaled to the respective volumes corresponding to the temperature values. The legend includes the arithmetic mean $\bar{x}$ as well as the standard deviation *sd* (Toifl et al., 2016a).

energy between 25 kW and 30 kW is only 100 kJ after 15 s of microwave treatment as opposed to 360 kJ after 72 s. Furthermore, figure 5.50 reveals almost identical temperature frequency densities for the 72 s, 25 kW simulation and the 60 s, 30 kW case. This coincidence of the graphs is caused by equal amounts of the supplied microwave energy in both cases ($\mathfrak{E}_{mw} = 1.8$ MJ). However, the maximum temperature is slightly different between the two models (72 s, 25 kW: $T_{max} = 239$°C and 60 s, 30 kW: $T_{max} = 246$°C, see figure 5.48 and figure 5.49). This variation is caused by the weaker thermal conductance in the 60 s irradiation case (figure 5.51).

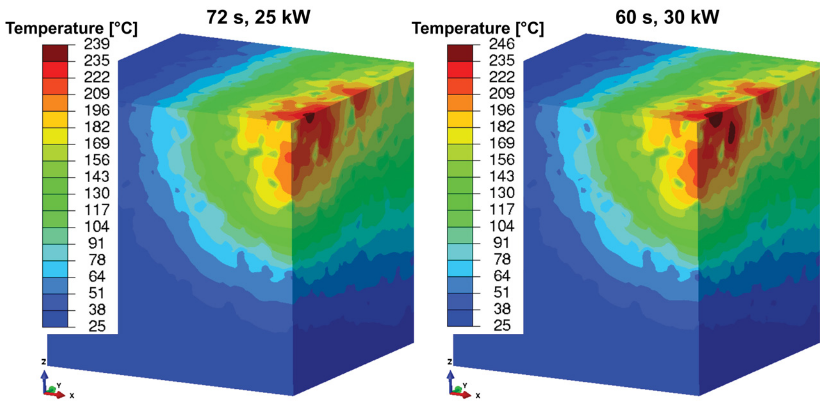


Fig. 5.51 Temperature field in °C in the microstructure cube after microwave irradiation with 72 s, 25 kW and 60 s, 30 kW, respectively (Toifl et al., 2016a).

Small variations are observed in the temperature field inside the microstructure of the two models with a microwave energy of 1.8 MJ (figure 5.51). Not only the magnitude of the maximum temperature is different but also the distribution varies. After 60 s of microwave irradiation with a source of 30 kW stronger temperature localization is observed. In order to investigate even more strongly varying combinations of microwave power and irradiation time additional simulations are performed for $\mathfrak{E}_{mw}$ = 1.8 MJ (cf. table 4.7). In figure 5.52 the temperature fields at the end of irradiation are presented.

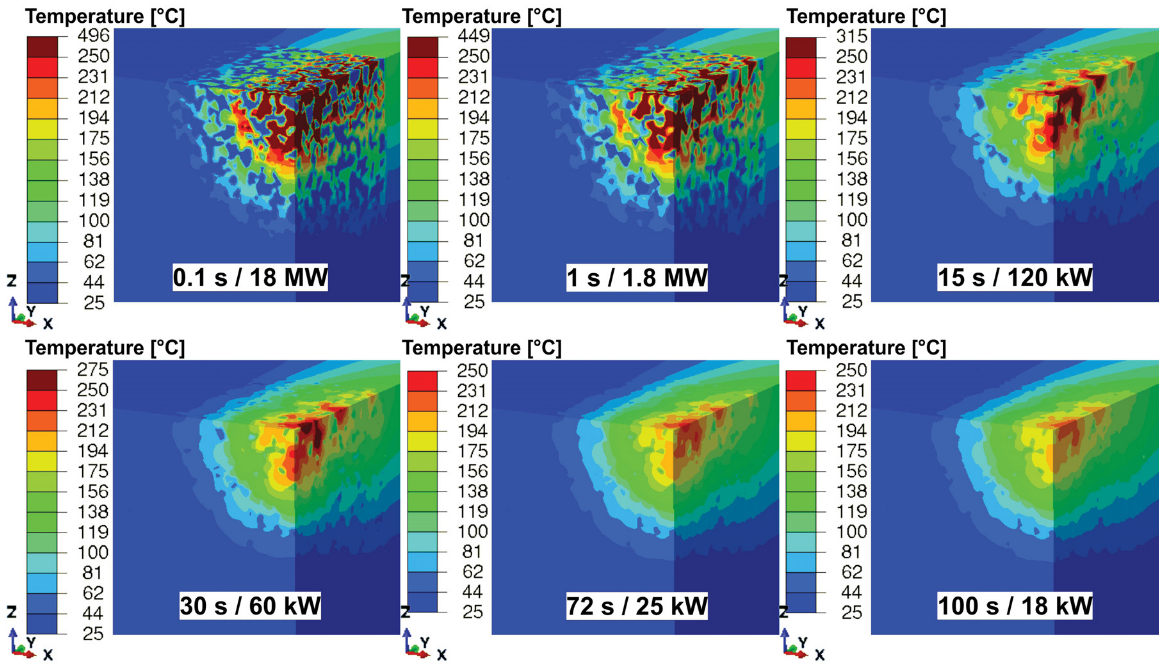


Fig. 5.52 Temperature field in °C after microwave irradiation with the microwave power and irradiation time given in the figure for the same energy $\mathfrak{E}_{mw}$ = 1.8 MJ.

The temperature fields show significantly larger differences for the simulations performed in figure 5.52 compared to figure 5.51. A maximum temperature of 496°C is observed after 0.1 s of microwave irradiation and 18 MW compared to 227°C after 100 s with 18 kW. Moreover, the 0.1 s, 18 MW case exhibits strong selective heating. The temperature fields of the different models with the same energy is assessed by a statistical analysis (figure 5.53).

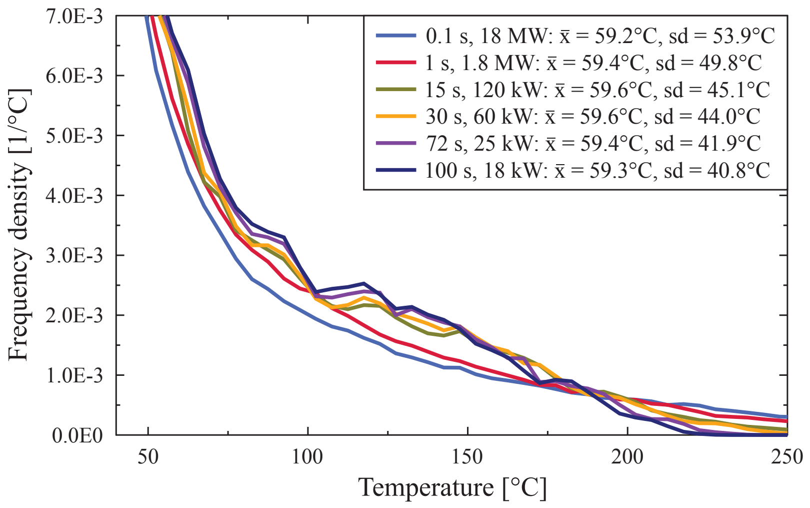


Fig. 5.53 Frequency density of the temperature of the microstructure cube scaled to the respective volumes corresponding to the temperature values for the models with constant microwave energy. The legend includes the arithmetic mean $\bar{x}$ as well as the standard deviation *sd*.

In figure 5.53 the influence of the irradiation time under constant microwave energy on the thermal field is clearly visible. With longer duration the thermal conductance has more time to smoothen the temperature field. Therefore, significantly more volume is never exposed to exceeding 180°C if the time duration is increased. The mean values of the thermal frequency density stay almost constant over all models but the standard deviation significantly increases with decreasing irradiation times (*sd* = 53.9°C in the 0.1 s, 18 MW case compared to *sd* = 40.8°C with 100 s, 18 kW).

5.1.3.3 Linear elastic stress results

First, the linear elastic stress results of the models summarized in table 4.6 are presented. Figure 5.54 visualizes the maximum principal stresses at the end of the irradiation time in the 25 kW case.

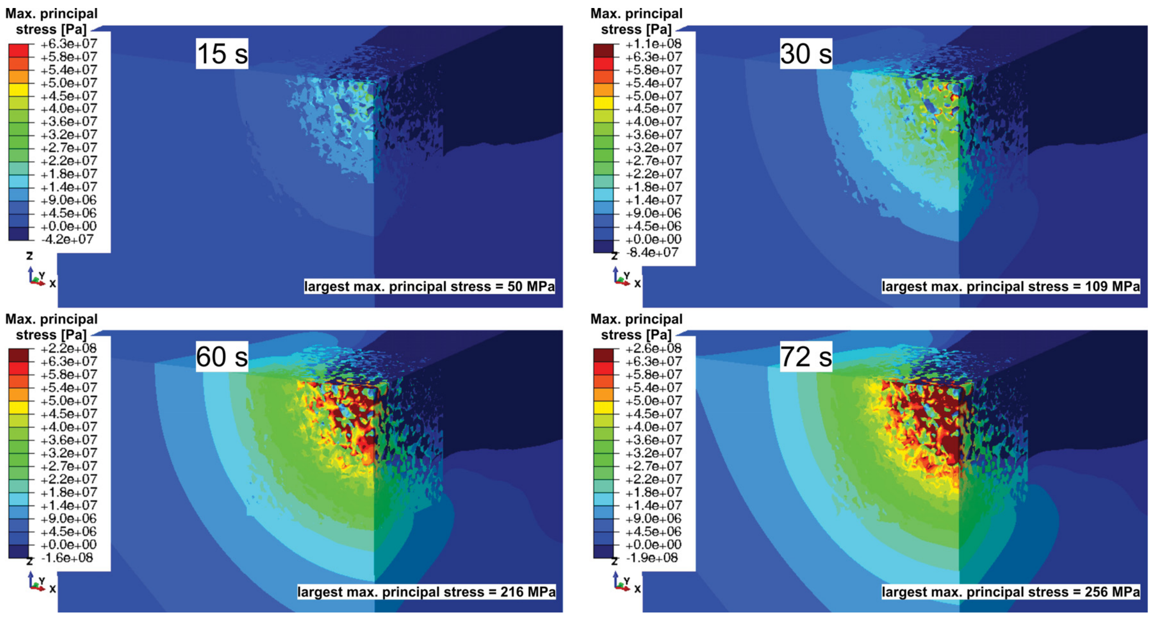

Max. principal stress [Pa]
15 s
largest max. principal stress = 50 MPa
30 s
largest max. principal stress = 109 MPa
60 s
largest max. principal stress = 216 MPa
72 s
largest max. principal stress = 256 MPa

Fig. 5.54 Maximum principal stress field in Pa after microwave irradiation with 25 kW (minus 30% losses) of granite models for durations given in the figure (Toifl et al., 2016a).

The maximum principal stress varies strongly between the different irradiation times (figure 5.54). After 15 s of irradiation only a small area with stresses higher than 9 MPa (light blue area in figure 5.54) and a maximum stress of 50 MPa are observed. With increasing durations the volume exposed to high maximum principal stresses expands significantly and much higher magnitudes are determined (largest maximum principal stress after 72 s = 256 MPa). The largest stresses are observed at the boundaries of the plagioclase grains, which represent the high absorbing phase in the models. In the 15 s case localized crack initiations near the front surface are expected since the area of stresses higher than 9 MPa penetrates only some centimeters into the depth of the material. After 72 s of irradiation the strength limit is exceeded down to a depth of about 10 cm (in the direction of microwave propagation).

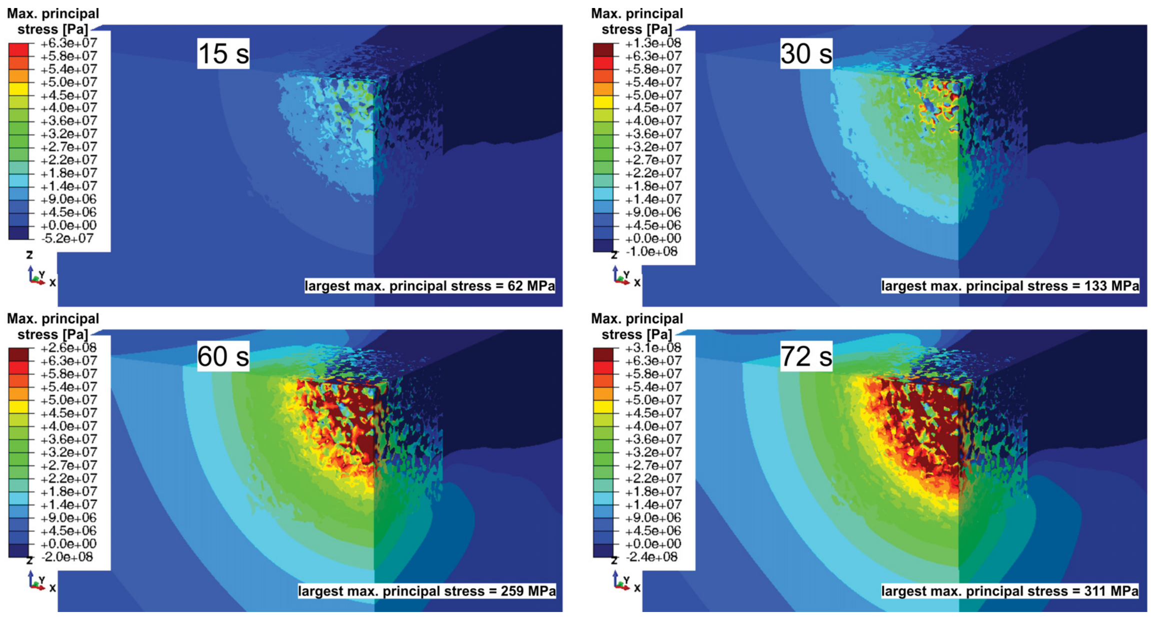


Fig. 5.55 Maximum principal stress field in Pa after microwave irradiation with 30 kW (minus 30% losses) of granite models for durations given in the figure (Toifl et al., 2016a).

By applying 30 kW instead of 25 kW higher maximum principal stresses are obtained (figure 5.55). After 15 s of irradiation the highest stress reaches 62 MPa, whereas after 72 s 311 MPa are observed. In order to investigate the differences in the stress fields between the various configurations in greater detail, a statistical analysis as described in section 5.1.2.3 is performed (figure 5.56). Here a class width of 1 MPa has been chosen.

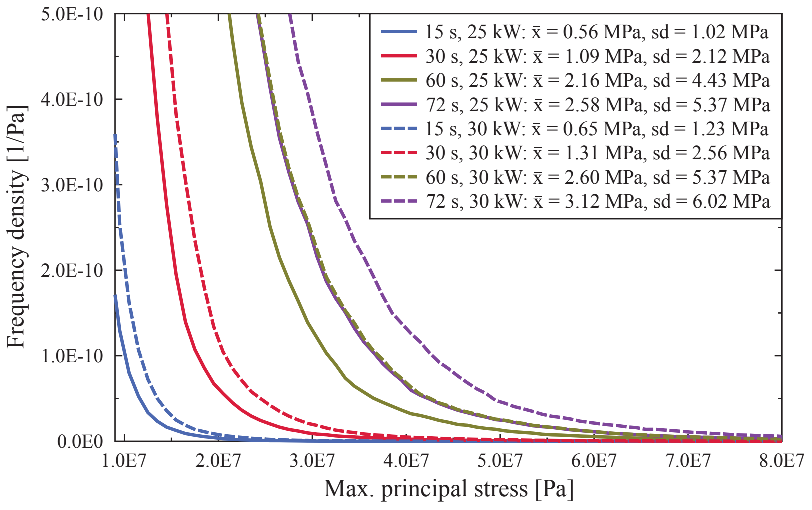


Fig. 5.56 Frequency density of the maximum principal stress of the total granite model. The legend includes the arithmetic mean $\bar{x}$ as well as the standard deviation *sd* (Toifl et al., 2016a).

Figure 5.56 reveals that the maximum principal stress distribution is shifted to higher values with increasing microwave irradiation times and provided powers (15 s, 25 kW: $\bar{x}$ = 0.56 MPa; 72 s, 25 kW: $\bar{x}$ = 2.58 MPa; 15 s, 30 kW: $\bar{x}$ = 0.65 MPa; 72 s, 30 kW: $\bar{x}$ = 3.12 MPa). Besides the shift of the arithmetic mean values the frequency densities also become broader (15 s, 25 kW: *sd* = 1.02 MPa; 72 s, 25 kW: *sd* = 5.37 MPa; 15 s, 30 kW: *sd* = 1.23 MPa; 72 s, 30 kW: *sd* = 6.02 MPa). The graphs for 72 s, 25 kW and 60 s, 30 kW are almost coincident. However, the largest maximum principal stress values vary slightly (256 MPa after 72 s, 25 kW, 259 MPa after 60 s, 30 kW). These differences are investigated in detail by plotting the microstructure cube in both cases (figure 5.57).

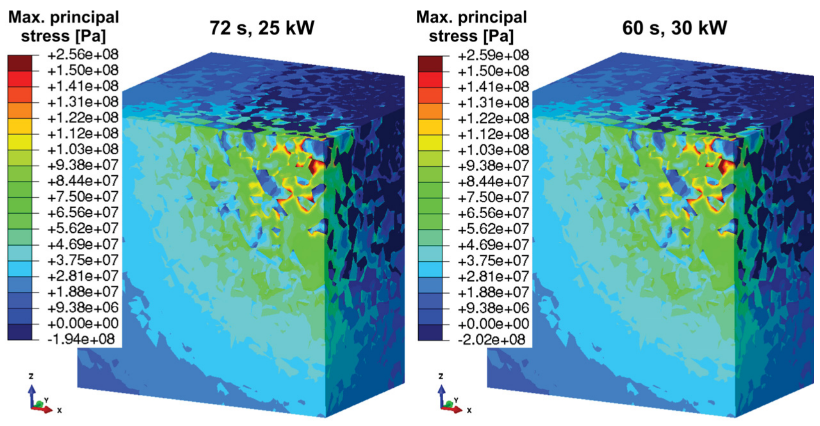


Fig. 5.57 Maximum principal stress field in Pa in the microstructure cube after microwave irradiation with 72 s, 25 kW and 60 s, 30 kW (Toifl et al., 2016a).

Although differences in the temperature fields between 72 s, 25 kW and 60 s, 30 kW are observed (figure 5.51), no significant variations in the stress field are detectable (figure 5.57). Only the largest maximum principal stress is slightly higher in the 60 s, 30 kW case. After 72 s microwave irradiation with 25 kW 4.60% of the total volume experience stresses higher than the material's strength (9 MPa), whereas after 60 s with 30 kW a slightly higher value of 4.63% is determined. In figure 5.58 the maximum principal stresses of the models with the same provided energy of $\mathfrak{E}_{mw} = 1.8$ MJ are visualized.

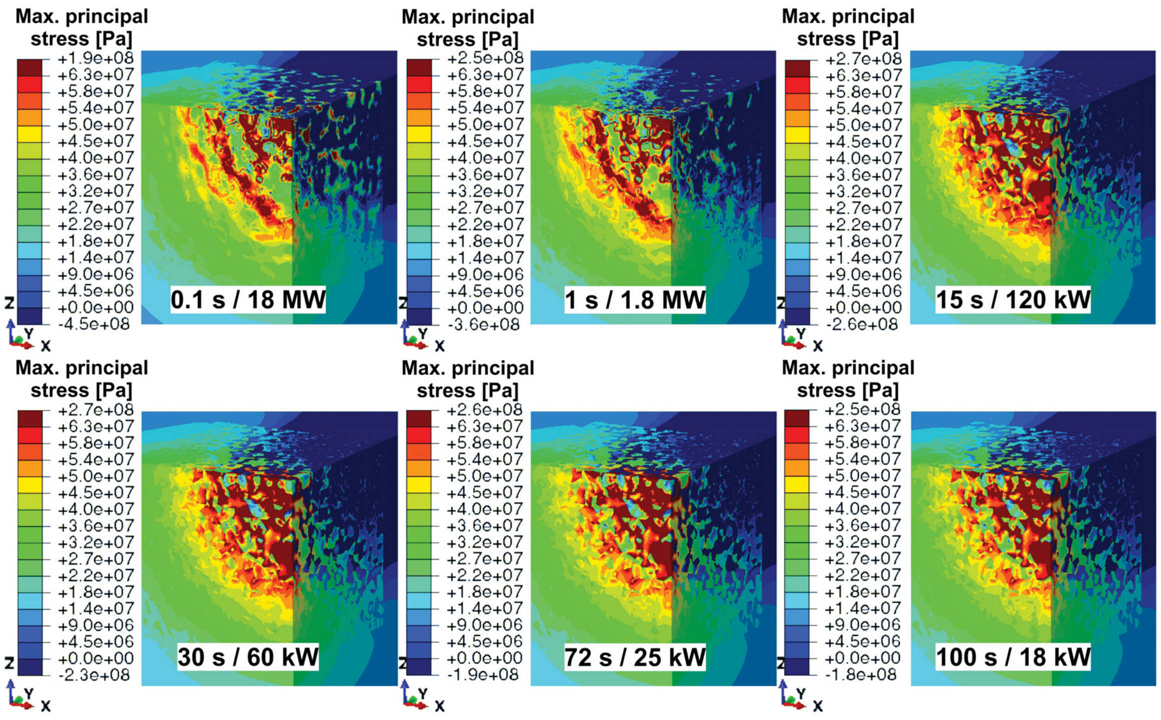


Fig. 5.58 Maximum principal stress field in Pa after microwave irradiation with the microwave power and irradiation time given in the figure for the same energy $\mathfrak{E}_{mw} = 1.8$ MJ.

Unlike the temperature field (figure 5.52), the largest maximum principal stresses are observed in the 15 s, 120 kW model. However, the remaining investigated models also reveal very high maximum principal stresses. A significant difference in the distribution of the stresses is observed. In the 0.1 s, 18 MW case the highest stresses are located at the phase boundaries of some plagioclase grains, whereas in the 100 s, 18 kW case significantly more volume contains high principal stresses. This tendency becomes more evident when visualizing the frequency density distribution of the six models (figure 5.59).

Figure 5.59 reveals that the mean of the maximum stress distribution rises with increasing irradiation times until 15 s is reached and then it drops (0.1 s, 18 MW: $\bar{x} = 2.62$ MPa; 15 s, 120 kW: $\bar{x} = 2.66$ MPa; 100 s, 18 kW: $\bar{x} = 2.55$ MPa). However, at 15 s, 120 kW the

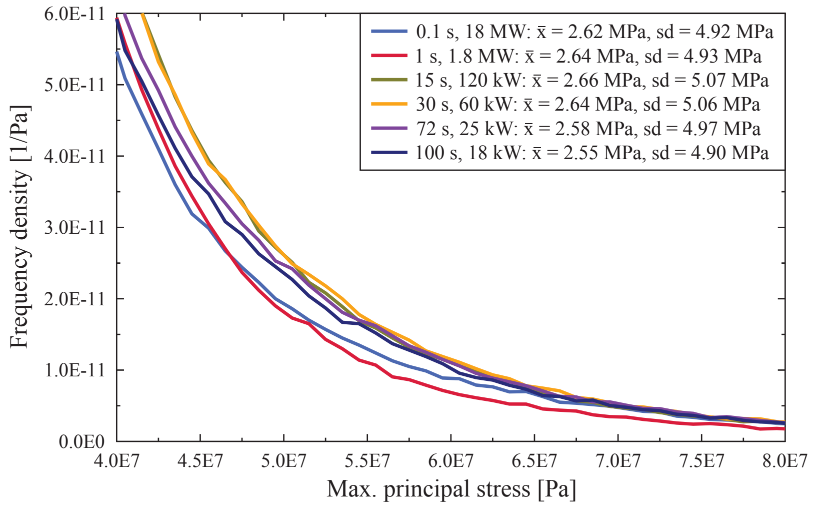


Fig. 5.59 Frequency density of the maximum principal stress of the total granite model scaled to the respective volumes corresponding to the stress values for the same energy $\mathfrak{E}_{mw} = 1.8$ MJ. The legend includes the arithmetic mean $\bar{x}$ as well as the standard deviation sd.

distribution is broader than in the 0.1 s or 100 s case (0.1 s, 18 MW: sd = 4.92 MPa; 15 s, 120 kW: sd = 5.07 MPa; 100 s, 18 kW: sd = 4.90 MPa). The graphs for 15 s, 120 kW and 30 s, 60 kW are very close to each other in the stress range depicted in figure 5.59. Moreover, it is obvious that in the 0.1 s, 18 MW and 1 s, 1.8 MW case significantly fewer material points are subjected to medium stress values, whereas especially the 0.1 s, 18 MW model reveals a large amount of highly stressed volume.

5.1.3.4 CDP calibration results

A purely elastic simulation is insufficient predicting any damage pattern for obvious reasons. Once the strength limit is reached or exceeded in an element this element will fail leading to a redistribution of the stresses which cannot be accounted for in an elastic model. Thus, it is inevitable to take damage mechanical aspects into account. A suitable damage model, in our case concrete damaged plasticity, requires a number of material parameters that need to be determined fairly accurately in order to be able to predict the damage behavior of the material reasonably well.

Various simulations with different morphologies are performed in order to calibrate the material parameters (mainly the thermal expansion coefficient α of the quartz phase) according to the thermal expansion measurements on granite samples. Finally, the uniaxial thermal expansion of the quartz phase above 800°C (blue vertical line in figure 5.60) is adapted in order to obtain the same thermo-mechanical response as the granite samples during the experiments.

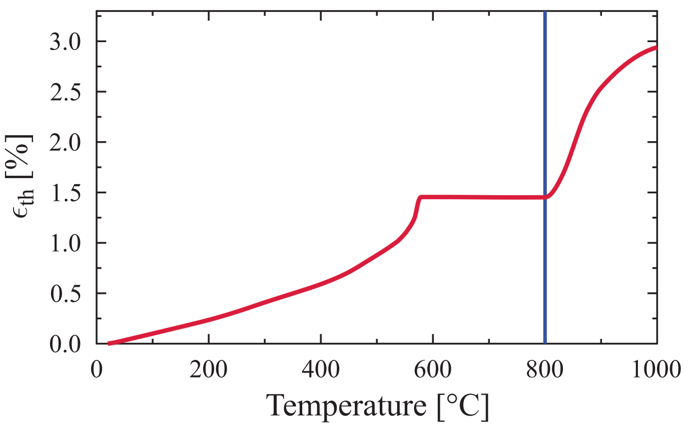


Fig. 5.60 Uniaxial thermal strains of quartz as a function of temperature °C (Carpenter et al., 1998). Values above 800°C are adapted according to the investigated granite.

A rapid increase in the thermal strains for temperatures higher than 800°C is observed. This sharp rise can be explained by the β quartz tridymite phase transition at 870°C which can lead to a jump in strains of up to 3.70% in one direction (Okrusch and Matthes, 2005). Moreover, it is assumed that the β quartz tridymite phase transformation only takes place during the first heating and not during reheating. Consequently, the uniaxial thermal strain in the second heating is constant for all temperature values higher than 600°C. In order to model this behavior, a user defined field variable is used in *Abaqus* and its value is adapted by a *USDFLD* subroutine (cf. appendix C) (Abaqus, 2014). This field variable is basically an indicator for the current phase of the quartz. With these thermal expansion values all presented results are obtained. Figure 5.61 shows the resulting strains of the various investigated morphologies.

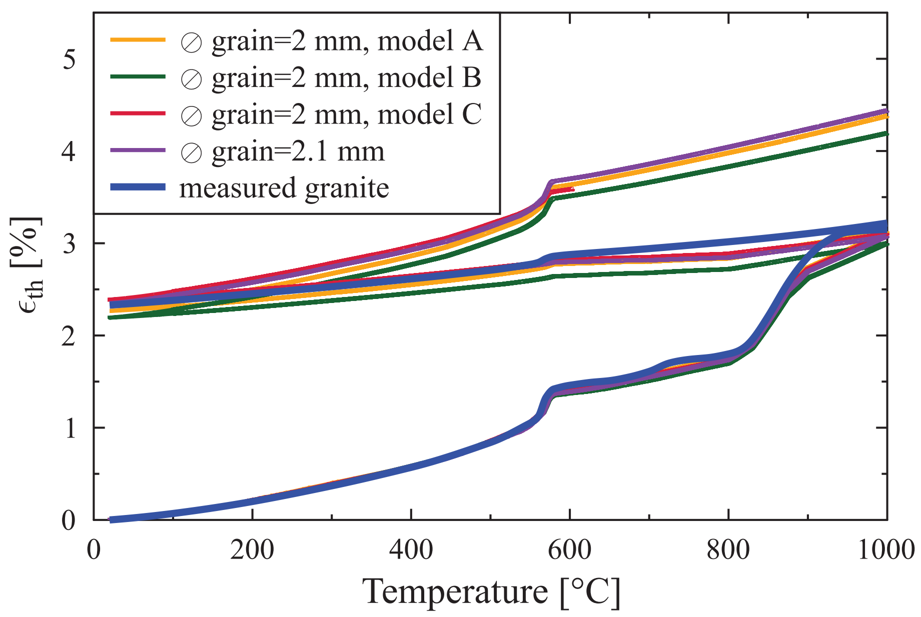


Fig. 5.61 Uniaxial thermal strains of the measured granite samples compared with four different numerical models, each one having its individual morphology.

The strain evolution calculated with the investigated CDP calibration models with different morphologies and slightly varying grain diameters agree well with the measured thermal strain values of the granite samples at least for the first heating and cooling. A maximum relative error of 4.6% at the end of the first heating (minimum: 0.5%) and 5.8% at the end of cooling (minimum: 0.9%) are observed. The relatively strong deviations between the different morphologies can also be seen in the thermal expansion measurements where they are even more pronounced (cf. figure 3 in Hartlieb et al. (2016)). However, the second heating cannot be captured by the CDP model. Evidently for a good representation of the second heating path a much more thorough analysis of the material model is required. However, this would go far beyond the scope of this thesis. Furthermore, for the problem at hand a second heating is not to be expected and thus will not have any influence on the granite CDP results. The strain components during first heating and cooling are analyzed in figure 5.62 exemplarily for model A with ⊘ grain = 2 mm.

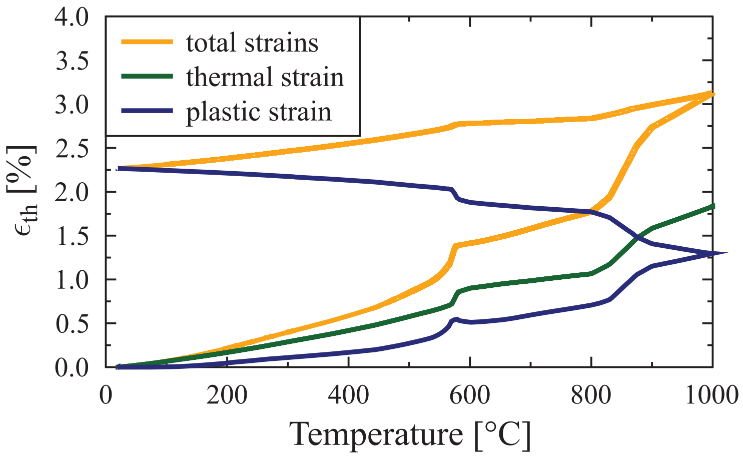


Fig. 5.62 Uniaxial thermal strain components of model A with ⊘ grain = 2 mm during first heating and cooling.

A maximum thermal strain of 1.83% is obtained in figure 5.62. Based on the strain definition the thermal strains are fully recoverable reaching zero strains at the end of the cooling step. Plastic deformation starts at temperatures of about 200°C and reaches 1.30% at the end of the heating step. Additionally, during cooling the plastic strains increase significantly and further smoothen the phase transitions. The spatial distribution of the scalar damage variable under tension d_t at different time points is depicted in figure 5.63 exemplary for model A with ⊘ grain = 2 mm.

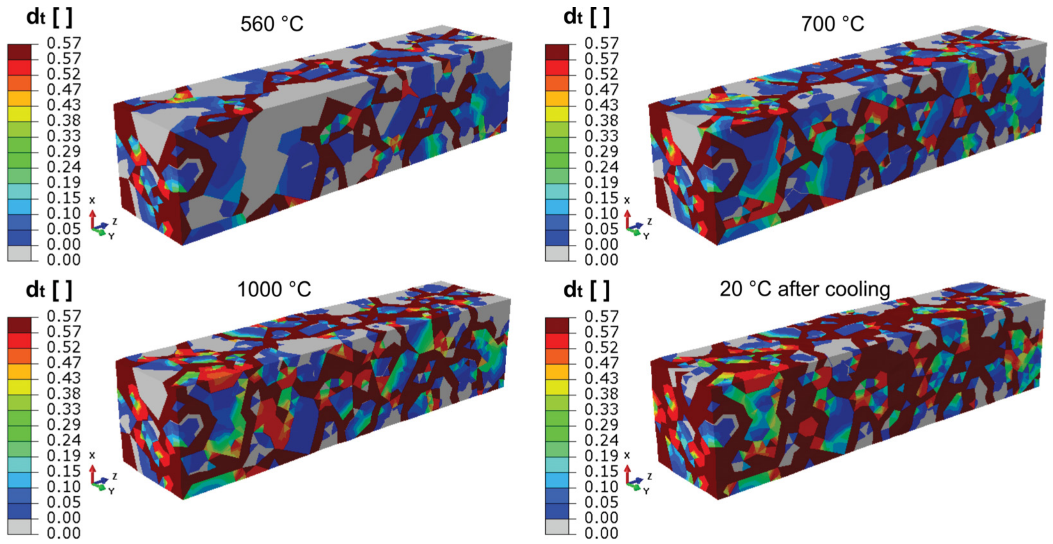


Fig. 5.63 Scalar stiffness degradation under tension d_t at different time points of model A with ⊘ grain = 2 mm.

In figure 5.63 severe damage (dark red in figure 5.63) is observed located around the phase boundaries. Moreover, a significant change is noted in the d_t distribution between the two phase transformations (cf. 560°C - 700°C and 700°C - 1000°C in figure 5.63). In most of the material points near phase boundaries plastic flow occurs. In order to determine the damage distribution on a microstructure level in more detail, a statistical analysis is performed by calculating the accumulated volume fraction of severely damaged material points under tension ($d_t \geq 0.57$) within a certain time step (figure 5.64).

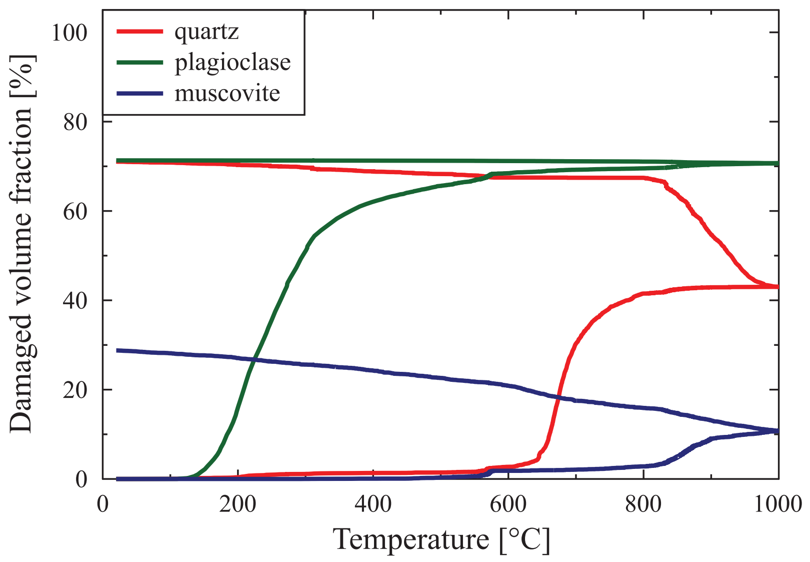


Fig. 5.64 Formation of volume fraction of damaged material points under tension ($d_t \geq 0.57$) for each constituent as a function of the temperature.

After 150°C first plagioclase grains are damaged in tension due to the strong mismatch of the thermal expansion coefficients (CTE) (cf. figure 4.16). Since plagioclase has the lower CTE, the grains are under strong tension and are therefore damaged first. Contrary, muscovite and quartz have almost equal CTEs up to a temperature of about 500°C and so hardly any damage is initiated. Only a small jump at the α to β phase transformation temperature is observed. However, at temperatures higher than 600°C quartz starts to damage rapidly. Muscovite shows a significant increase in damaged volume fraction during the β quartz tridymite phase transformation. At the end of the heating step 43% of the quartz, 71% of the plagioclase and 11% of the muscovite volume are damaged. At the beginning of the cooling step an increase in damaged quartz material points during the tridymite β quartz phase transformation is observed. Additionally, the muscovite grains also exhibit an increase in damage. Finally, at room temperature 71 vol.% of quartz, 71 vol.% of plagioclase and 29 vol.% of muscovite are damaged in tension.

5.1.3.5 Damage results

In the 3D three component simulations the calibrated concrete damaged plasticity material model is used (cf. section 5.1.3.4) to identify damage. The CDP model is only applied in the microstructure cube and not in the homogeneous part in order to reduce the computational cost. The distribution of the scalar stiffness degradation d_t within the microstructure at various irradiation times under a constant power of 18 MW is visualized in figure 5.65.

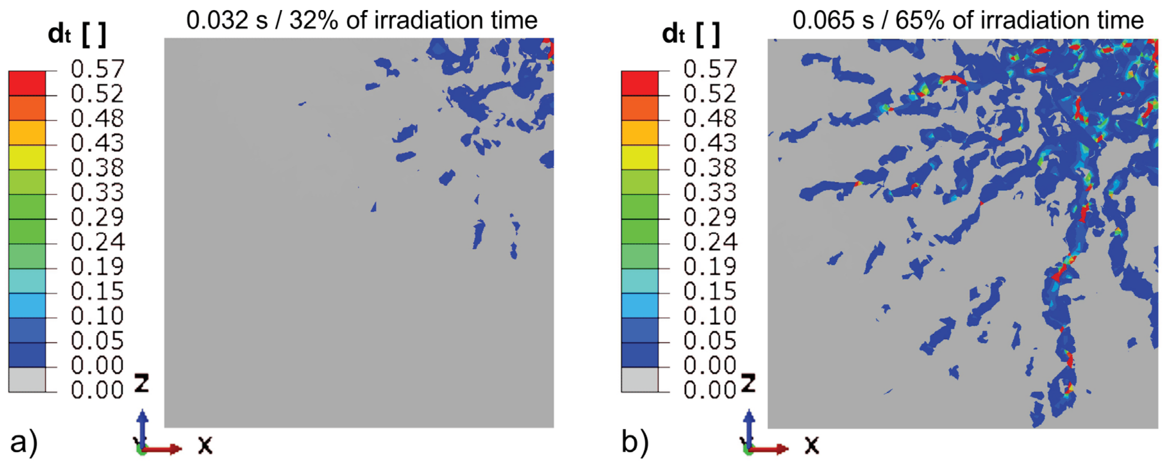


Fig. 5.65 Scalar stiffness degradation variable d_t in the three component model after a) 0.032 s (32% of irradiation time) and b) 0.065 s (65% of irradiation time) microwave irradiation with 18 MW.

Damage and subcritical cracks (blue areas figure 5.65) are initiated around quartz phase boundaries near the absorbing plagioclase grains originating close to the Gaussian axis. In contrast to the 3D two component model, where longer irradiation times are investigated, the highest damage is observed in quartz and not in the strongly microwave absorbing phase. These deviations between the two models arise due to the strong localized heating in the 0.1 s, 18 MW model which leads to differences between adjacent grains of about 450°C. Conversely, in the 15 s, 25 kW case (3D two component model) the temperature field is more homogeneous.

With longer irradiation times critical cracks (red in figure 5.65) propagate radially outside the irradiated volume. Analyzing the damage after 0.065 s of microwave irradiation yields radial patterns (figure 5.66).

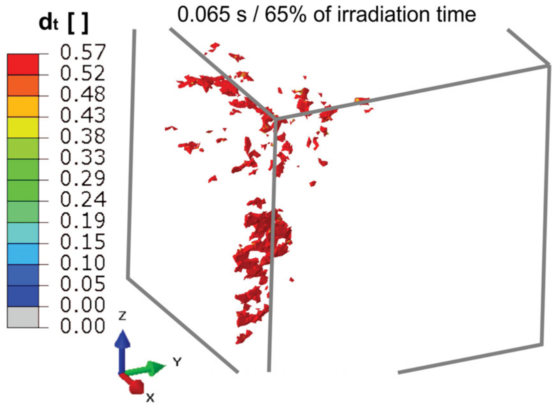


Fig. 5.66 Damaged elements ($d_t \geq 0.57$) in the three component model at 0.065 s (65% of irradiation time) of microwave irradiation with 18 MW. Microwaves propagate along positive y-axis.

Figure 5.66 shows damage located up to a depth of 3.3 cm. The observed radial damage pattern correlates to the 3D two component CDP results. Not only the material limit in tension but also in compression is reached after 0.065 s of irradiation (figure 5.67).

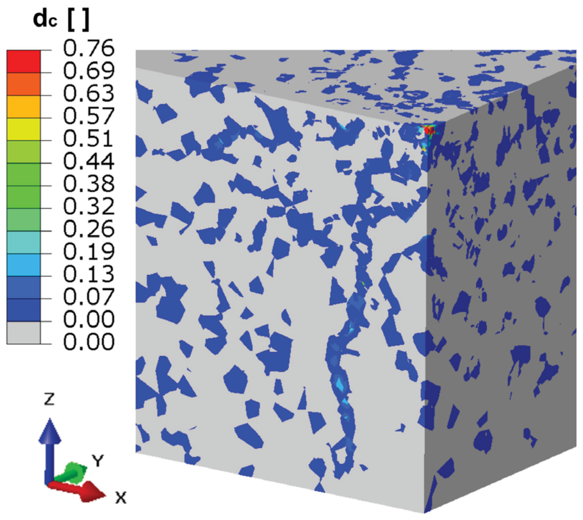


Fig. 5.67 Scalar stiffness degradation variable in compression d_c in the three component model after 0.065 s (65% of irradiation time) microwave irradiation with 18 MW.

Damage under compression is determined at the front face of the Gaussian axis. However, in some muscovite grains damage is also initiated due to high compressive stresses. The full 3D three component model taking into account the highly non-linear CDP material model is computationally extremely extensive leading to calculation times of around two months on a state-of-the-art 12 CPU cluster node equipped with 256 GB RAM. In addition to the

0.1 s, 18 MW case also the 1 s, 1.8 MW (figure 5.68) and 72 s, 25 kW (figure 5.69) cases have been investigated including the CDP material model.

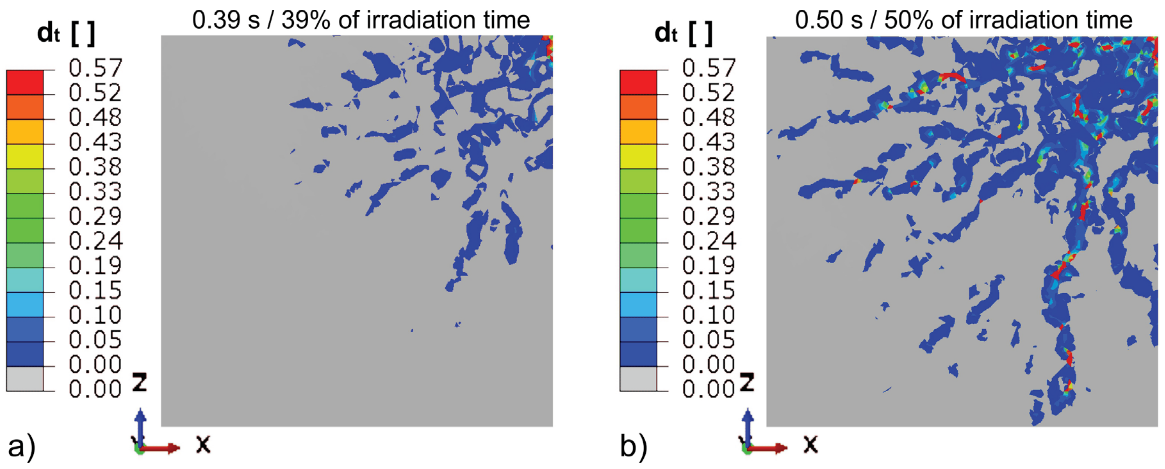


Fig. 5.68 Scalar stiffness degradation variable d_t in the three component model after a) 0.39 s (39% of irradiation time) and b) 0.50 s (50% of irradiation time) microwave irradiation with 1.8 MW.

Figure 5.68 reveals damage initiation close the Gaussian axis and damage propagation in radial direction. Damage can be located up to a depth of 2 cm. For the 72 s, 25 kW case only the results after 27.53 s are available (figure 5.69).

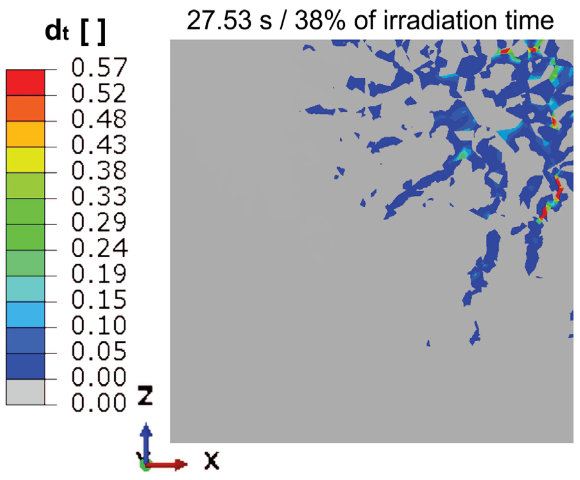
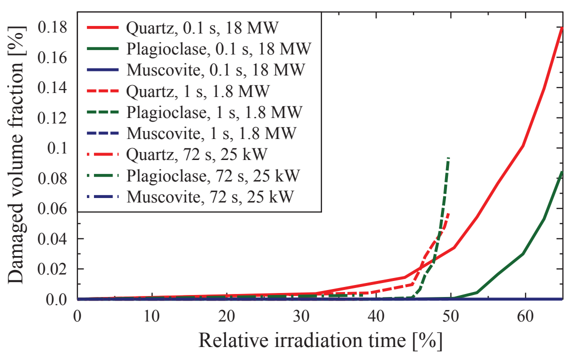


Fig. 5.69 Scalar stiffness degradation variable d_t in the three component model after 27.53 s (38% of irradiation time) of microwave irradiation with 25 kW.

After 27.53 s of microwave irradiation with 25 kW damage is initiated mainly in plagioclase grains which represent the microwave absorbing part (figure 5.69). The formation of damage of the investigated 3D three component CDP models is summarized in figures 5.70 and 5.71.



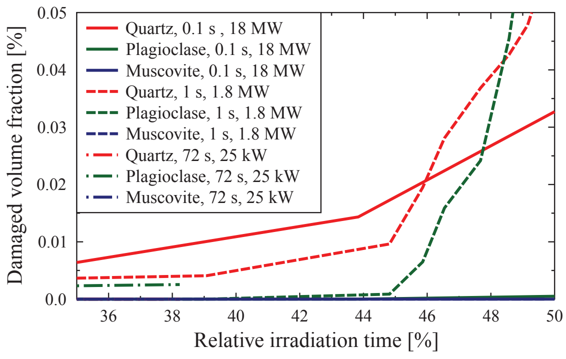


Fig. 5.70 Formation of volume fraction (within the microstructure) of damaged material points under tension $d_t \geq 0.57$ over the relative irradiation time [%].

Fig. 5.71 Formation of volume fraction (within the microstructure) of damaged material points under tension $d_t \geq 0.57$ between 35% and 50% of the irradiation time.

In the 0.1 s, 18 MW case figure 5.70 reveals higher damage under tension in the quartz than in the plagioclase phase. After 46% of the irradiation time of the 1 s, 1.8 MW model more volume is damaged in tension than in the 0.1 s, 18 MW case. Moreover, plagioclase is the most damaged phase after 48% of the irradiation time. Conversely, in the 72 s, 25 kW case only the plagioclase phase is damaged until an irradiation time of 38% (figure 5.71). In all three models no tension damage can be observed in the muscovite grains. Since the damage behavior in the subsequent time steps would follow the same tendency no additional output is generated. Obviously, the location of the initiation of damage depends on the microwave irradiation parameters (irradiation time and microwave power).

5.2 Homogeneous models

By considering a rather homogeneous hard rock such as basalt the influence of the strong coupling between the electromagnetic and thermal field is investigated. To this end, the measured temperature-dependent permittivity of basalt is used (cf. figure 4.29).

5.2.1 Thermal results

Based on the simulation methodology outlined in section 4.2 the thermal field is recalculated after a maximum temperature change of 50°C using an updated electromagnetic field. The thermal fields at the end of the heating time of the strongly (SCM) and weakly coupled models (WCM) are compared in figure 5.72.

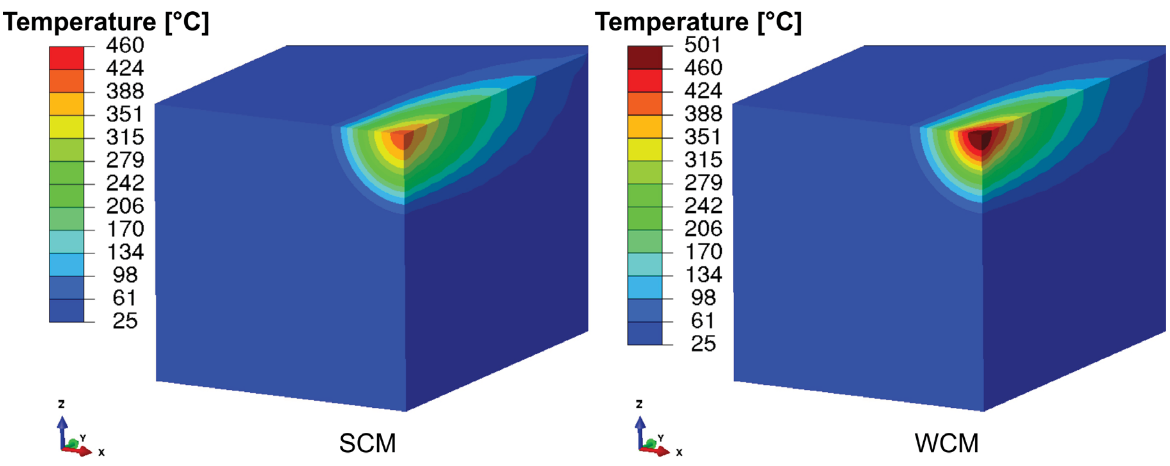


Fig. 5.72 Comparison between thermal fields of SCM and WCM for basalt after microwave irradiation with 1.8 MW for 1 s.

After a microwave irradiation of 1 s with 1.8 MW significant differences can be seen between the thermal fields of the SCM and WCM. The maximum temperature in the two models differs by about 100°C (maximum temperature 408°C in SCM and 501°C in WCM). Near the Gaussian axis a larger depth of microwave heated areas are observed in the SCM. These differences are caused by the decrease in the complex permittivity between 100°C and 400°C (cf. figure 4.29). Since the imaginary part of the permittivity decreases, the penetration depth increases (cf. equation 2.19). The influence of the strong FDTD-FEM coupling (SCM) on the resulting thermal field becomes evident when plotting the temperature over the time for varying depths along the Gaussian axis (figure 5.73).

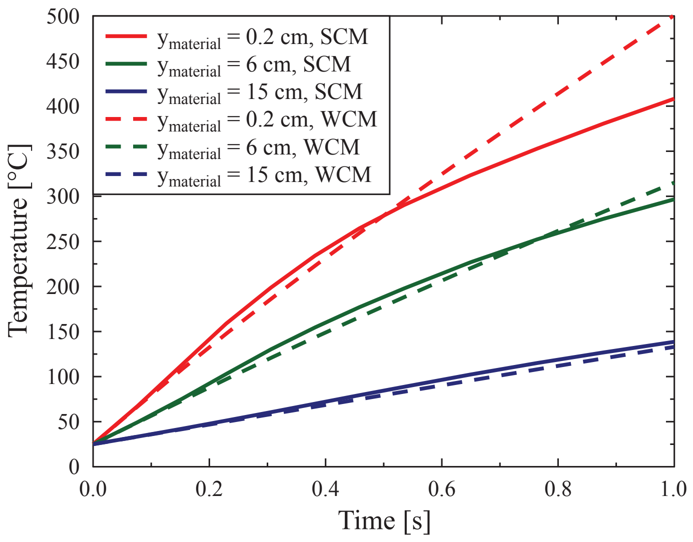


Fig. 5.73 Temperature as a function of time for points along the Gaussian axis with different y-positions within the material ($y_{material}$).

In a depth of 0.2 cm (where the highest temperature is observed) the strongest difference between the SCM and WCM is observed (figure 5.73). Until 0.5 s the SCM reveals higher temperatures due to the initial increase in the imaginary part of the permittivity (cf. figure 4.29). With longer irradiation significantly higher temperatures in the WCM are observed due to the rapid drop of the imaginary part of the permittivity in the SCM. In a depth of 15 cm a lower temperature in the WCM is found due to the reduced penetration depth. The influence of the temperature-dependent permittivity is further investigated by analyzing a constant microwave power of 25 kW and two different irradiation times (figure 5.74).

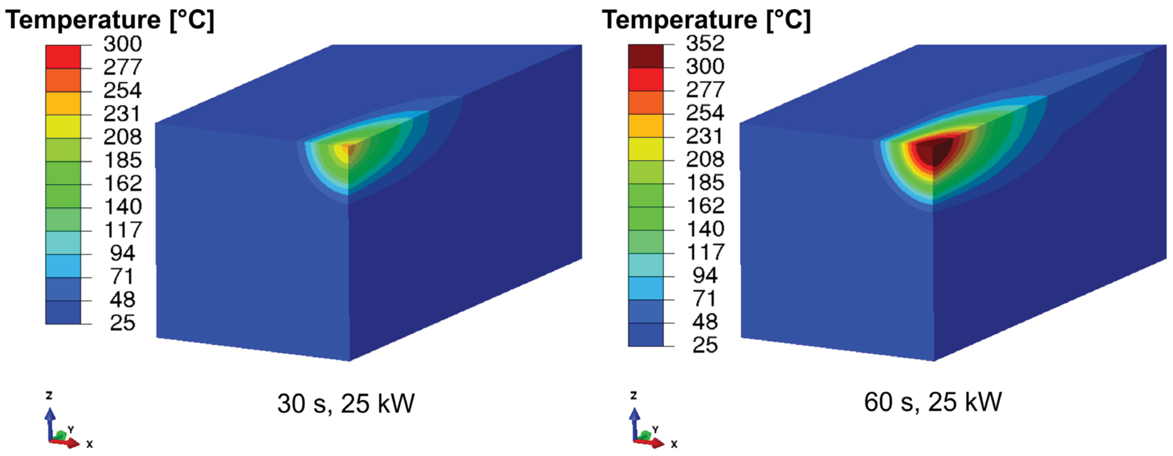


Fig. 5.74 Comparison between thermal fields of SCM for basalt after microwave irradiation with 25 kW for 30 s and 60 s.

A maximum temperature of 242°C is observed after microwave irradiation with 25 kW and 30 s whereas after 60 s a temperature of 352°C is reached. The 30 s, 25 kW case yields an almost ideal Gaussian beam which becomes distorted after 60 s of microwave irradiation with 25 kW. This effect is caused by the temperature-dependent complex permittivity of basalt (cf. figure 4.29). Between 20°C and 200°C the imaginary part of the permittivity is almost constant which results in an ideal Gaussian beam. Conversely, between 200°C and 400°C $\epsilon''_{r,b}$ drops rapidly resulting in a narrower distribution of the heated volume.

5.2.2 Linear elastic stress results

The thermal fields after each calculated increment are used as an input for the subsequent stress analysis. Differences in the maximum principal stress fields between the SCM and WCM are depicted in figure 5.75.

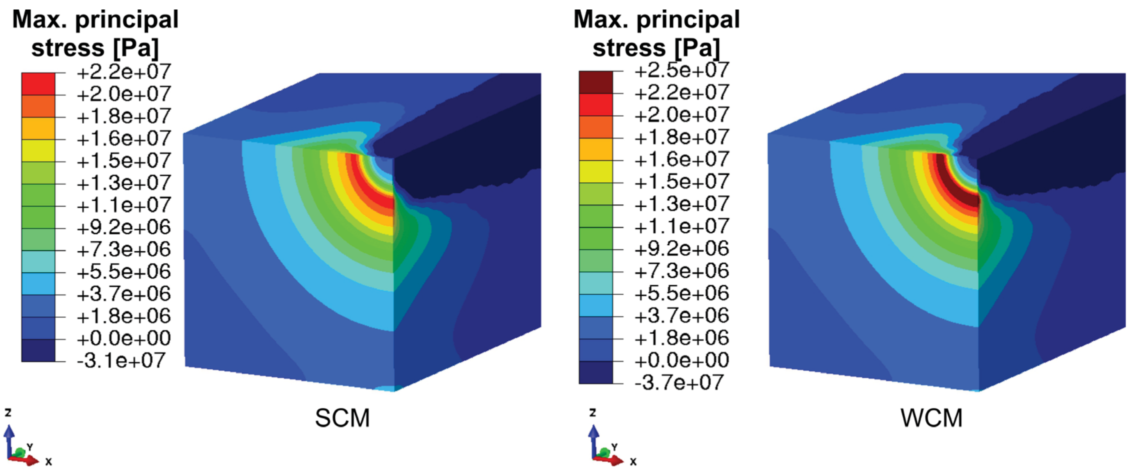


Fig. 5.75 Comparison between maximum principal stress fields of SCM and WCM for basalt after microwave irradiation with 1.8 MW for 1 s.

In the WCM higher maximum principal stresses are observed than in the SCM (figure 5.75). Also a larger volume fraction exposed to critical stress levels is found. The influence of varying irradiation time at constant power on the maximum principal stress field of the SCM is investigated in figure 5.76.

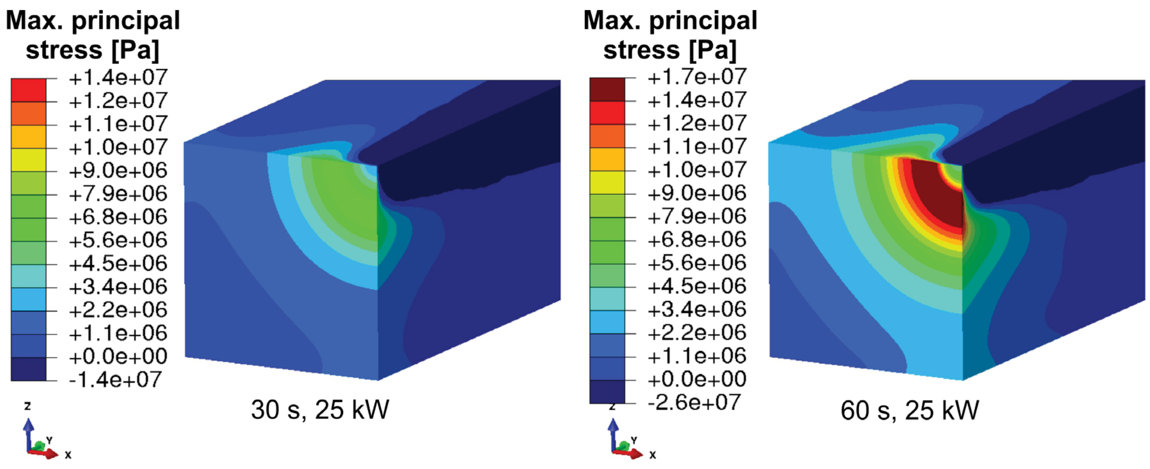


Fig. 5.76 Comparison between maximum principal stress fields of SCM for basalt after microwave irradiation with 25 kW for 30 s and 60 s.

The highest maximum principal stresses in figure 5.76 are observed around the main Gaussian beam localized at the front surface. Hence, damage in a circular area close to the surface is expected. In the 25 kW and 30 s microwave irradiation model the highest stresses are slightly below the tensile strength of the basalt material ($\sim$ 9 MPa). After 60 s of microwave irradiation with a microwave source of 25 kW the highest stress reaches 17 MPa.

Chapter 6

Discussion

In this chapter the results of the various numerical models are discussed. In order to compare the numerical results with experiments, high power microwave irradiation experiments were performed by Dr. Philipp Hartlieb from the Chair of Mining Engineering and Mineral Economics, Montanuniversitaet Leoben. Since mainly granite but also basalt are investigated in the numerical models, the experimental results of these hard rocks are presented in section 6.1.

6.1 Experimental work

The results of the microwave irradiation experiments have been published in Meisels et al. (2015) and Toifl et al. (2016b). These experiments on a hard rock blocks ($50 \times 50 \times 30$ cm^3, 30 cm in direction of microwave irradiation) are performed using an open-ended rectangular waveguide (4.3×8.6 cm^2) as the applicator. Due to the few centimeters distance between waveguide and rock the area of the irradiated spot on the rock is approximately circular (red circle in figure 6.1a). The microwave source emits radiation with 25 kW at a frequency of 2.45 GHz. The power transmitted into the granite block is estimated to be 30% less. In figure 6.1 a granite block after microwave irradiation for durations given in this figure is visualized.

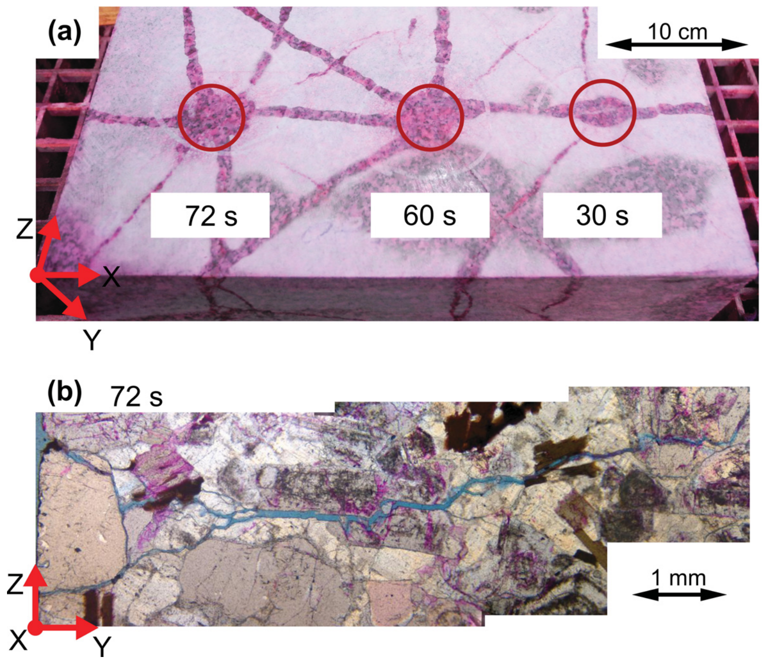


Fig. 6.1 Microwave irradiation with 25 kW (power of the microwave source) of granite for durations given in the figure. (a) Red circles indicate the area of highest intensity in the microwave beam. (b) Microstructure underneath hotspot of (a) after 72 s microwave irradiation (Toifl et al., 2016b).

As illustrated in figure 6.1, intense cracking caused by the thermo-mechanical stresses is observed in the granite block originating from a hot spot beneath the waveguide. Since the different irradiation spots are far apart from each other, only minor influences of the existing crack network on the crack initiation during the subsequent microwave irradiation experiment are assumed. By analyzing the in-depth crack paths into the material (cf. figure 6.1b) it is concluded that the microstructure has a major influence on the onset and subsequent accumulation of damage since the cracks (blue in figure 6.1b) mainly follow the grain boundaries. This implication agrees with the numerical results obtained in the inhomogeneous models where the stress formation is driven by microstructural details. In a rather homogeneous hard rock, such as basalt, a different damage behavior is observed (figure 6.2).

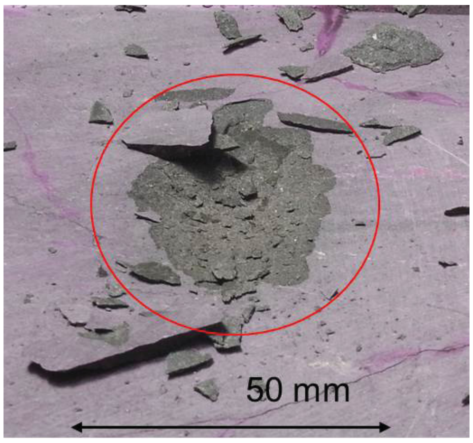


Fig. 6.2 Spot on basalt irradiated with microwaves with 25 kW for 15 s. Spallation is clearly visible. The irradiated area has approximately the size of the circle (Meisels et al., 2015).

The region of high intensity is marked in figure 6.2 by a circle with 5 cm diameter. Microwave irradiation of a basalt sample reveals spallation. Both experiments (figures 6.1 and 6.2) show severe damage due to microwave irradiation. However, the type of damage (cracking or spallation) depends on the type of rock, in particular on its dielectric, thermal and mechanical properties.

6.2 Discussion of 2D models

In the 2D FDTD electromagnetic results deviations of the local E_y^2 values between the heterogeneous and the homogeneous model are up to 10% due to the scattering by the discs (figure 5.2). Moreover, strong selective heating and therefore high thermal gradients between the constituents are observed. This tendency is exaggerated by the relatively low thermal conductivity of the plagioclase discs compared to the quartz-like matrix (cf. table 4.2). The temperature gradients due to selective heating and the significant deviation of thermal expansion coefficients between matrix and discs lead to high maximum principal stresses (figures 5.7 and 5.8). These cause interface damage of the discs in a wide range around the irradiation spot (figures 5.10 and 5.11). During the cool-down process cracks propagate far outside the irradiated spot at the boundaries between discs and matrix.

The accuracy of the presented 2D simulations is validated by preliminary experiments. A correlation between the intense cracking around the irradiation spot in the granite sample (figure 6.1) and the cracked discs at maximum temperature in the thermomechanical FE simulation (figure 5.10) can be found. The cracks at the granite experiments oriented in the depth of the sample mostly follow the grain boundaries, which concurs with the numerical simulations (cf. figure 5.10).

In the 2D simulations it is concluded that crack initiation and propagation are driven by the inhomogeneous microstructure of hard rocks. Moreover, the results show that the heterogeneous nature of rocks has to be considered for understanding microwave induced stresses and damage in inhomogeneous hard rocks.

6.3 Discussion of 3D two component models

A two component 3D hard rock model with resolved microstructure is used to determine the influence of the microstrucuture on microwave induced damage. Regarding the electromagnetic field of the microwave irradiated inhomogeneous rock model, $\overline{E^2}$ stripe-like patterns are determined (figure 5.12). Moreover, in the vicinity of the microstructure / bulk material interface no artificial reflections are observed, which indicates that Bruggeman's effective medium theory works well in the current application. Along the axis of the Gaussian beam (figure 5.14) a deviation of up to 12% between the homogeneous and inhomogeneous model is visible. After 15 s of microwave irradiation with 25 kW the resulting maximum principal stresses exceed the material strength in a wide range. The highest stresses are observed at the boundaries of the absorbing phase near the Gaussian axis of the beam (figure 5.20). Additionally, high compressive stresses exceeding the material limit are identified at the phase boundaries in phase T (figure 5.23). When elevated temperatures or high confining pressures are reached, a transition from brittle to ductile rock material behavior can occur. Under the assumption of ductile material behavior, plastic deformations arise near the phase boundaries of the inhomogeneous model due to high Tresca stresses (figure 5.24). Contrary, in the homogeneous model the highest stresses occur around the main heated area (figure 5.21). This varying stress formation implies a different damage mechanism for the two models.

By comparing the stress field of three different morphologies with the same filling factor ($f = 0.34$) the high influence of the microstructural details on the microwave induced stresses becomes evident. Although all three models feature equal averaged grain size ($\oslash$ grain = 3.4 mm) and filling factor, significant differences in the stress formation occur (figure 5.25). Even stronger differences are determined if these models are compared with two different filling factors ($f = 0.2$ and $f = 0.4$). The volume fraction in the microstructure that is exposed to maximum principal stresses higher than the tensile strength, increases with higher values of the filling factor (cf. figure 5.28). Additionally, the deviation between the different morphologies with the same filling factor is larger in the $f = 0.2$ case than in the other two cases. This effect can be explained by the higher possibility of accumulation of high absorbing grains near the Gaussian axis with increasing filling factor.

After 25 s of microwave irradiation phase transformation in the phase T (quartz) takes place which entails even higher stresses (figure 5.29). By performing a statistical analysis a strong shift of the stresses in phase T and phase A to higher maximum principal stresses is observed (figure 5.30). Furthermore, broader frequency density distributions of the maximum principal stresses in phase T and phase A are observed. A significantly higher portion of the irradiated volume experiences stress values exceeding 50 MPa after 25 s of microwave heating.

The stresses increase if the anisotropic nature of the quartz crystals is taken into account. By comparing the stress field after 25 s of irradiation in the isotropic with those of the anisotropic case (figure 5.33), a significantly larger volume fraction exposed to high maximum principal stresses is observed in the anisotropic model. Furthermore, a statistical analysis of the stress formation in the isotropic as well as the anisotropic case (figure 5.34) shows that the anisotropic model predicts fewer elements under compression and more elements under tension in phase T. Hence it is concluded that in the anisotropic model higher crack propagation dynamics is expected since the phase T grains do not have the potential to arrest a crack due to the lack of compressive stresses in the grains. However, if the anisotropic nature of the grains is considered in the models to predict microwave induced damage, several morphologies have to be investigated in order to obtain statistically reliable results.

The CDP material results reveal large areas of damage initiation and propagation (figure 5.35). Damage is initiated at the phase boundaries of phase A grains mainly near the Gaussian axis as well as around the main heated area. With increasing irradiation time the damage propagates far outside primarily following the phase boundaries. In the cooling step a higher degree of damage is determined near the Gaussian axis. In this region no damage can be observed in the homogeneous model (figure 5.36). Considering the damage in both phases, phase A contains significantly more damaged volume than phase T. With longer irradiation time combined with α to β phase transformation significantly more material points are damaged. A comparative analysis with extended model dimensions taking into account a CDP material definition throughout the entire model reveals longer damage propagation but similar damage initiation patterns. The damage pattern correlates qualitatively very well with the experiments performed on granite blocks, where damage propagation along the phase boundaries is observed as well.

The question of whether crack formation can be induced by microwave heating is essentially dominated by the macroscopic thermal gradients. However, the position of damage initiation sites and the rates as well as propagation path is influenced by the microstructure. The set of numerical results of the 3D two component models indicates that the consideration of microstructural details is crucial to determine reliable microwave induced stresses and damage in rocks with strongly absorbing phases. Moreover, different models with varying phase distributions but constant filling factors should be performed in order to reveal statistically reproducible stress fields. Obviously, for an accurate prediction of the stress formation the phase transformation of quartz as well as the anisotropic nature has to be taken into account.

6.4 Discussion of 3D three component models

In the three component 3D models real granite samples are investigated where the dielectric as well as the thermo-mechanical properties were taken from measurements. Additionally, the same granite used for the simulations was investigated in the microwave irradiation experiments (figure 6.1). In order to assess the influence of the microwave irradiation parameters on the induced damage, various analyses with different combinations of microwave power and irradiation time for varying energy (table 4.6, discussion of results in section 6.4.1) and constant energy levels (table 4.7, discussion of results in section 6.4.2) are performed.

First, the electric field inside the inhomogeneous granite model is obtained (figure 5.44) which serves as the major input for all following irradiation experiments (since the morphology is the same in all models). Due to the strong differences in the permittivities of the three minerals, strong variations in the electric field compared to a homogeneous case are observed. These differences amount up to 40% between the homogeneous and inhomogeneous model along the Gaussian beam axis (figure 5.46). The resulting absorbed power density (figure 5.47) is used as an input for the thermal FE calculations.

6.4.1 Constant microwave power

Strong differences between the various microwave irradiation durations and the two power levels are observed by considering models with varying microwave energy. With longer irradiation time the maximum temperature rises and the heated area increases (figure 5.48 - 5.50). The resulting temperatures for different microwave parameters and energies are summarized in figure 6.3.

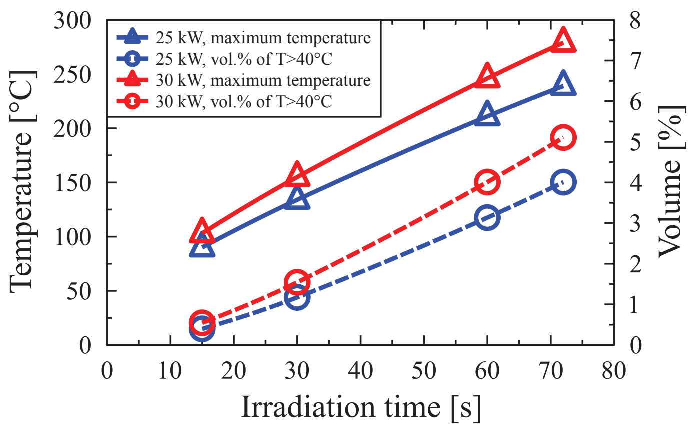


Fig. 6.3 Assessment of the influence of microwave irradiation parameters on the temperature field in a three component 3D granite model for constant microwave power (Toifl et al., 2016a).

In figure 6.3 an almost linear relation between the maximum temperature and the microwave irradiation time is observed. The slope of the straight line is steeper in the 30 kW than in the 25 kW case. Although the same microwave energy ($\mathfrak{E}_{mw} = 1.8$ MJ) is provided for the 72 s, 25 kW and the 60 s, 30 kW case, differences in the temperature fields are observed (figure 5.51). The maximum temperature after 60 s microwave irradiation with 30 kW is higher than after 72 s with 25 kW (246°C compared to 239°C). Only a marginally larger volume becomes exposed to temperature values higher than 40°C (4.01 vol.% compared to 4.00 vol.%). These variations are caused by the shorter time available for the heat conduction to compensate the thermal gradients in the 60 s case compared to 72 s. This effect is investigated in more detail with different models with the same microwave energy in section 6.4.2.

Strong variations in the maximum principal stress distribution are observed for the investigated models (figure 5.54 - 5.56). Much higher stresses are obtained with increasing irradiation times and power levels. The highest maximum principal stresses occur at the boundaries of plagioclase grains near the Gaussian axis. The influence of the microwave irradiation parameters on the formation of the stresses is summarized in figure 6.4.

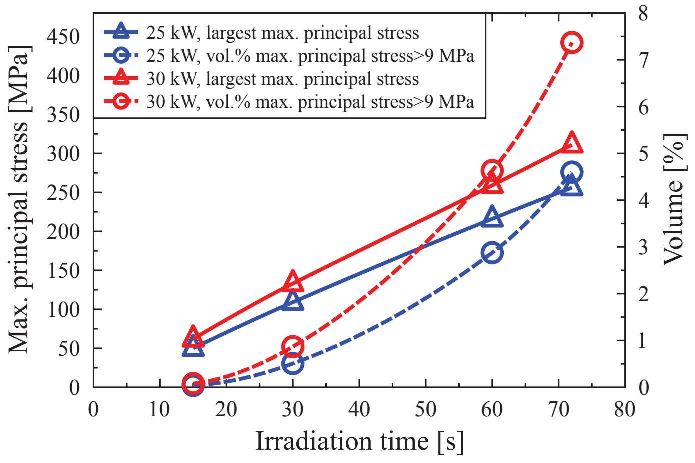


Fig. 6.4 Assessment of the influence of microwave irradiation parameters on the maximum principal stress field in a three component 3D granite model for constant microwave power (Toifl et al., 2016a).

A linear relation between the maximum principal stresses and the irradiation time is observed in figure 6.4. The stresses increase more rapidly in the 30 kW than in the 25 kW case. The curve representing the volume fraction being subjected to stresses greater than the material strength (9 MPa) as a function of the irradiation time is almost cubic. Contrary to the thermal field no significant differences in the stress fields are observed between the 60 s, 30 kW and the 72 s, 25 kW case. However, this circumstance is caused by the small variation of irradiation time and power and cannot be seen in section 6.4.2.

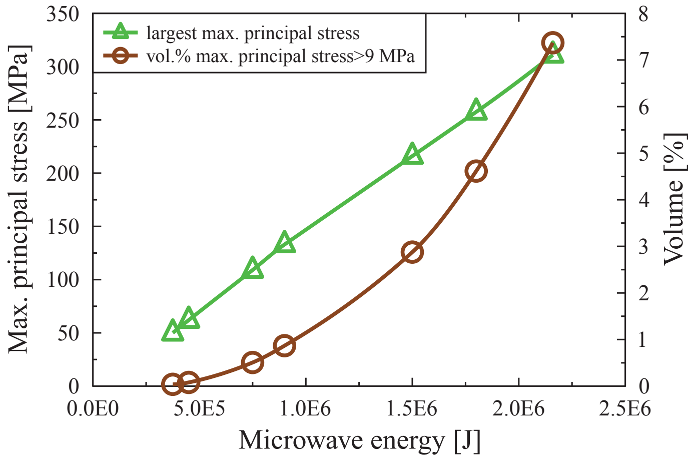


Fig. 6.5 Assessment of the influence of microwave energy $\mathfrak{E}_{mw}$ on the maximum principal stress field in a three component 3D granite model (Toifl et al., 2016a).

The microwave induced stresses in the models investigated with different irradiation parameters (cf. table 4.6) only depend on the supplied microwave energy, as demonstrated in figure 6.5 where the largest principal stress is plotted as a function of the energy. A linear dependence can be derived for the largest maximum principal stress and a cubic dependence for the critical volume fraction on the energy input.

6.4.2 Constant microwave energy

Since minor differences between 60 s, 30 kW and 72 s, 25 kW are observed (cf. section 6.4.1), various simulations with stronger differences in the microwave irradiation time and power but constant energy $\mathfrak{E}_{mw}$ are performed (cf. table 4.7). The models with microwave powers in the megawatt (MW) range correspond to pulsed magnetron applications. Significant differences in the temperature fields are observed. The maximum temperature in the 0.1 s, 18 MW model reaches 496°C whereas in the 100 s, 18 kW case only 227°C is observed. Moreover, very high selective heating for short durations can be seen where longer irradiation times lead to quite homogeneous fields. In figure 6.6 the temperature field is analyzed as a function of the microwave irradiation time for constant microwave energy.

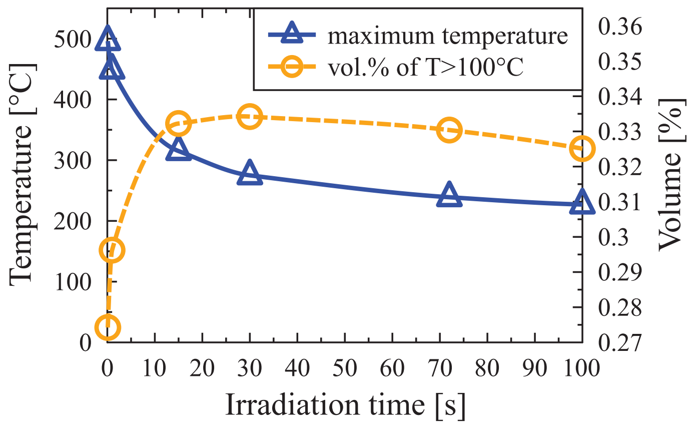


Fig. 6.6 Temperature field evaluation as a function of irradiation time with constant microwave energy.

The maximum temperature is a strongly nonlinear function of the microwave irradiation time. When the thermal conductance has enough time the maximum temperature rapidly decreases due to homogenization of the temperature field. Conversely, the volume fraction exposed to temperatures exceeding 100°C first rises and then drops with increasing microwave irradiation time.

Strong differences in the maximum principal stress fields are revealed between the models with constant microwave energy (figures 5.58 and 5.59). In the 0.1 s, 18 MW case the highest stresses are strongly localized at the phase boundaries whereas in the 100 s, 18 kW case a higher volume fraction becomes exposed to high stresses. The largest maximum stress is observed in the models for 15 s, 120 kW and 30 s, 60 kW (both 266.8 MPa, see figure 6.8). Figures 6.7 and 6.8 allow to investigate the stress formation for cases with constant provided energy as a function of irradiation time.

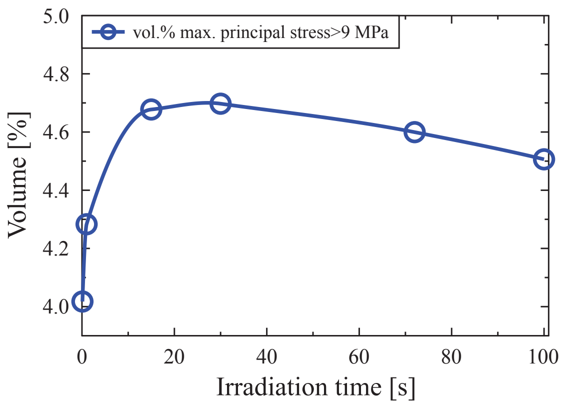


Fig. 6.7 Volume fraction exposed to maximum principal stresses higher than tensile strength (9 MPa) as a function of irradiation time for constant microwave energy.

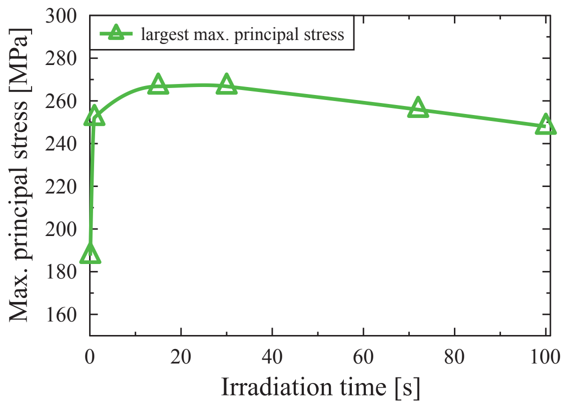


Fig. 6.8 Largest maximum principal stresses as a function of irradiation time for constant microwave energy in the 3D three component granite model.

In figure 6.7 the volume fraction subjected to maximum principal stresses higher than 9 MPa (which is the tensile strength) reveals a local maximum for the 30 s, 60 kW case. In lower irradiation cases the highest stresses become strongly localized and less volume experiences stresses exceeding the tensile strength (9 MPa) than with longer irradiation time (at least until 100 s). With irradiation lasting longer than 30 s the volume fraction of high stresses drops due to the homogenization of the thermal field.

With the measured granite a three component CDP material model is calibrated (cf. section 5.1.3.4). The thermo-mechanical behavior can be described well by the CDP model for the first heating and cooling (figure 5.61). Then, the calibrated model is used to assess the damage initiation and propagation in the 3D three component model under microwave irradiation. Different damage behaviors depending on the used microwave power and irradiation time are determined (cf. figures 5.70 and 5.71). Damage in tension is mainly observed close to the Gaussian axis and in radial direction near the main heated area. With short irradiation time and high power (0.1 s and 18 MW) the quartz phase is first damaged, whereas with longer irradiation time (72 s and 25 kW) the microwave absorbing plagioclase phase is mainly damaged in tension. Finally, it is concluded that not only the amount of damage but also the location highly depends on the microwave irradiation parameters.

6.5 Discussion of the 3D coupled homogeneous models

In the 3D homogeneous models strong coupling between FEM and FDTD is considered. The influence of this coupling is assessed using basalt samples with a measured temperature-dependent permittivity. After a microwave irradiation with 1.8 MW for 1 s the difference in the maximum temperatures between the strongly (SCM) and weakly coupled (WCM) models is around 100°C (figure 5.72). Moreover, varying penetration depths are revealed due to the change of the permittivity. The influence of the strong coupling on the resulting thermal field is most pronounced in a temperature range between 100°C and 400°C (cf. figure 5.74). In this range the imaginary part of the permittivity changes by a factor of 2.6 (cf. figure 4.29).

The highest maximum principal stresses in the homogeneous basalt models are observed near the main Gaussian beam (figure 5.76). Qualitatively, the location of the highest stresses correlates with the evidence of the microwave irradiation experiments on basalt samples. Longer irradiation time but constant power result in higher maximum principal stresses which penetrate deeper into the material (figure 5.76).

Chapter 7

Conclusion and Outlook

In the thesis at hand various numerical simulations have been performed on inhomogeneous as well as homogeneous hard rocks in order to quantify microwave induced stresses and damage.

7.1 Conclusion

For the inhomogeneous hard rocks preliminary 2D simulations on an artificial disc shaped microstructure have been carried out. As the main part of this work a novel comprehensive 3D simulation chain for determining the microwave induced stresses in realistic microstructures has been presented. The concept comprises the analysis of the electromagnetic, the thermal and the stress / damage fields of a microstructure in a block of a model rock irradiated with a microwave beam. The simulation methodology has been applied to an artificial two component rock (which is comparable to the 2D case) as well as to a three phase granite rock model. For the three phase granite model the phase distribution of the constituents have experimentally been determined and used for the calculations. Moreover, the permittivity and thermo-mechanical material parameters have been measured for a specific granite.

The conclusions of the assessment of the microwave irradiation behavior of inhomogeneous rocks are summarized below:

- **Inhomogeneous microstructure has a strong influence**: The numerical calculations of inhomogenenous rocks reveal strong differences between the electromagnetic, thermal, stress and damage fields between homogeneous (without microstructure) and inhomogeneous models (resolved microstructure). For a detailed assessment of microwave induced stresses in rather heterogeneous hard rocks such as granite the microstructure has to be resolved.

- **Quartz phase transformation induces significantly higher stresses**: When the α to β quartz transformation temperature of 573°C is reached a significant increase in stresses is determined. Therefore, microwave heating exceeding the α to β phase transformation temperature is preferable for rocks with a significant amount of quartz (i.e. granite, quartzolite, rhyolite, dacite).

- **Anisotropic quartz behavior leads to a different stress state**: The stress state changes significantly when the anisotropic material behavior of quartz is taken into account. Then the quartz grains near the Gaussian axis, which are subjected to strong compression in the isotropic case, change to tension in the anisotropic model. Therefore, a different crack pattern is expected in the anisotropic case. However, several morphologies have to be investigated in order to produce statistically reliable results.

- **CDP material model can capture microwave induced damage**: In the three component 3D calibration model it is proven that the CDP material model is capable of describing the damage behavior of the granite sample at least during first heating and cooling. The resulting damage patterns after microwave irradiation of the 2D as well as 3D models correlate with the performed microwave irradiation experiments.

- **Stress state varies with morphology**: With varying morphology but constant volumetric phase distribution differences in the maximum principal stress fields are observed. This behavior shows the strong influence of the microstructure on the stress formation but also the necessity to investigate various morphologies to obtain statistically reliable results.

- **Large differences between varying filling factors**: In the two component 3D model significant differences are observed in the maximum principal stresses between models with different filling factors (f, volume fraction of absorbing phase). With a higher amount of absorbing grains in the microstructure a higher volume fraction becomes exposed to stresses larger than the tensile strength. Moreover, the deviation between the different morphologies is smaller compared to models with fewer absorbing grains.

- **Longer irradiation times under constant power lead to higher stresses**: The parameter analysis of the 3D three component granite model reveals that with a constant provided microwave power (25 kW or 30 kW) the maximum temperature as well as the largest maximum stress increase linearly with the irradiation time. The volume fraction being subjected to stresses exceeding the tensile strength follows a cubic law as a function of the irradiation time.

- **With constant energy an optimum irradiation time can be found giving maximum stresses:** In the case of a constant provided microwave energy the maximum observed temperature in the 3D three component granite model first drops rapidly with an increase in the irradiation time. When a certain time is reached (in the investigated cases 30 s and 60 kW) the decrease in the maximum temperature over the time becomes significantly slower. Due to the very strong selective heating in the case of short irradiation times a smaller volume fraction becomes exposed to temperatures above 100°C. However, the volume fraction $>$100°C first rises rapidly until the 30 s, 60 kW case is reached and then decreases with increasing irradiation times due to the effect of the thermal conductance. A quite similar trend is observed for the volume fraction of material exposed to stresses exceeding the tensile strength (9 MPa). Here also the 30 s, 60 kW model reveals a local maximum. For a given phase distribution and provided microwave energy the derived numerical methodology allows to determine an optimum combination of irradiation time and power.

The presented 3D simulation chain for heterogeneous hard rocks allows assessing different grain diameters, morphologies, constituents and filling factors. Moreover, the methodology can easily be extended to more than three constituents as well as to an actual microstructure as measured by appropriate characterization techniques. Finally, the presented simulations as well as the experiments confirm that microwave treatment has the potential to induce high stresses which can eventually lead to damage formation.

7.2 Outlook

The presented simulation strategy serves as an excellent basis for detailed further investigations. In future research following issues should be considered to gain deeper insight into the microwave induced fragmentation:

- **Model validation**: An important issue for future research is the development of a comprehensive measurement setup in order to correlate the numerical models with the microwave experiments quantitatively. To this end, the damage network inside the rock should be visualized and correlated with the grain distribution of the rock.
- **Material characterization of the minerals**: In order to use realistic material parameters of the different phases, permittivity as well as thermo-mechanical measurements of the different minerals should be performed in future investigations. Especially, the permittivity of the minerals but also the thermal conductance, specific heat, thermal expansion, tension and compression (triaxial) behavior are needed to generate more realistic material models.
- **Real measured microstructure**: Generating models with grain and phase distributions as measured by computer tomography (CT) investigations would be preferable for validation purposes. Then the damage pattern obtained by such realistic models can directly be compared to the experiments.
- **CDP models in all thermo-mechanical simulations**: In all thermo-mechanical simulations the CDP model should be used in order to identify damage initiation and propagation. To this end, the computational power has to be increased significantly to achieve reasonable calculation times.
- **Coupled FDTD-FEM simulations also for the inhomogeneous case**: So far, the reported methodology in the inhomogeneous case is limited to temperature independent permittivities since no strong coupling between FDTD and FE simulations has been considered. Future work should also include this effect in a more sophisticated simulation procedure. This requires an improvement of the numerical code on the one hand, but also larger computational resources on the other hand.
- **Energetic assessment of microwave irradiation**: After the numerical models have quantitatively been validated by microwave irradiation experiments the energetic benefit of a foregoing microwave irradiation can be assessed. For this purpose a material

characterization test (e.g. tension, compression, Brazilian test) should be performed numerically as well as experimentally before and after microwave irradiation.

- **Fracture mechanical model**: After cracks have initiated their propagation should be investigated by appropriate fracture mechanics models. Some potential techniques which can be implemented in the framework of FE are XFEM, configurational force concept or a phase-field model.

The microwave induced rock fragmentation has proven its potential to assist fragmentation processes by introducing crack networks in hard rocks. In this vein, it may be possible in the future to significantly reduce the excavation costs in the mining and tunneling industry.

References

Abaqus (2014). Abaqus v6.14 documentation. http://www.3ds.com/products-services/simulia/products/abaqus/. Last accessed on 29.04.2016.

Ackermann, R. J. and Sorrell, C. A. (1974). Thermal expansion and the high–low transformation in quartz. I. High-temperature X-ray studies. *Journal of Applied Crystallography*, 7(5):461–467.

Ali, A. Y. and Bradshaw, S. M. (2009). Quantifying damage around grain boundaries in microwave treated ores. *Chemical Engineering and Processing: Process Intensification*, 48(11-12):1566–1573.

Ali, A. Y. and Bradshaw, S. M. (2010). Bonded-particle modelling of microwave-induced damage in ore particles. *Minerals Engineering*, 23(10):780–790.

Ali, A. Y. and Bradshaw, S. M. (2011). Confined particle bed breakage of microwave treated and untreated ores. *Minerals Engineering*, 24(14):1625–1630.

Amankwah, R. K., Khan, A. U., Pickles, C. A., and Yen, W. T. (2005). Improved grindability and gold liberation by microwave pretreatment of a free-milling gold ore. *Mineral Processing and Extractive Metallurgy*, 114(1):30–36.

Arzúa, J. and Alejano, L. R. (2013). Dilation in granite during servo-controlled triaxial strength tests. *International Journal of Rock Mechanics and Mining Sciences*, 61:43–56.

Atanasoff, J. V. and Hart, P. J. (1941). Dynamical determination of the elastic constants and their temperature coefficients for quartz. *Physical Review*, 59:85–96.

Atkinson, K. E. (1989). *An introduction to numerical analysis*. Wiley, New York, 2nd edition.

Balanis, C. A. (2012). *Advanced engineering electromagnetics*. John Wiley & Sons, Hoboken, 2nd edition.

Bathe, K.-J. (2007). *Finite element procedures*. Prentice-Hall, Upper Saddle River.

Beardsmore, G. R. and Cull, J. P. (2001). *Crustal heat flow: A guide to measurement and modelling*. Cambridge University Press, Cambridge and New York.

Benisek, A., Dachs, E., and Carpenter, M. A. (2013). Heat capacity and entropy of low structural state plagioclases. *Physics and Chemistry of Minerals*, 40(2):167–173.

Birch, A. F. and Clark, H. (1940). The thermal conductivity of rocks and its dependence upon temperature and composition. *American Journal of Science*, 238(8):529–558.

Bradshaw, S. M., Louw, W., van der Merwe, C., Reader, H., Kingman, S., Celuch, M., and Kijewska, W. (2007). Techno-economic considerations in the commercial microwave processing of mineral ores. *Journal of Microwave Power & Electromagnetic Energy*, 40:228–240.

Bruggeman, D. A. G. (1935). Berechnung verschiedener physikalischer Konstanten von heterogenen Substanzen. I. Dielektrizitätskonstanten und Leitfähigkeiten der Mischkörper aus isotropen Substanzen. *Annalen der Physik*, 416(7):636–664.

Busetti, S., Mish, K., Hennings, P., and Reches, Z. (2012a). Damage and plastic deformation of reservoir rocks: Part 2. Propagation of a hydraulic fracture. *AAPG Bulletin*, 96(9):1711–1732.

Busetti, S., Mish, K., and Reches, Z. (2012b). Damage and plastic deformation of reservoir rocks: Part 1. Damage fracturing. *AAPG Bulletin*, 96(9):1687–1709.

Carpenter, M. A., Salje, E. K. H., Graeme-Barber, A., Wruck, B., Dove, M. T., and Knight, K. S. (1998). Calibration of excess thermodynamic properties and elastic constant variations associated with the $\alpha \leftrightarrow \beta$ phase transition in quartz. *American Mineralogist*, 83:2–22.

Cassidy, N. J. (2009). Electrical and magnetic properties of rocks, soils and fluids. In Jol, H. M., editor, *Ground Penetrating Radar Theory and Applications*, pages 41–72. Elsevier, Amsterdam and Boston.

Cermak, V. and Rybach, L. (1982). Thermal properties. In Angenheister, G., editor, *SpringerMaterials - The Landolt-Börnstein Database: Physical Properties of Rocks*, pages 305–343. Springer, Berlin.

Chen, T. T., Dutrizac, J. E., Haque, K. E., Wyslouzil, W., and Kashyap, S. (1984). The relative transparency of minerals to microwave radiation. *Canadian Metallurgical Quarterly*, 23(3):349–351.

Chunpeng, L., Yousheng, X., and Yixin, H. (1990). Application of microwave radiation to extractive metallurgy. *Chinese Journal of Materials Science & Technology*, 6:121–124.

Church, R. H., Webb, W. E., and Salsman, J. B. (1988). *Dielectric properties of low-loss minerals: Report of Investigations 9194*. US Bureau of Mines.

Cyrus, M. and Beck, J. (1978). Generalized two- and three-dimensional clipping. *Computers & Graphics*, 3(1):23–28.

DOE (2007). US Department of Energy, Mining Industry Energy Bandwidth Study. http://www1.eere.energy.gov/manufacturing/resources/mining/pdfs/mining_bandwidth.pdf. Last accessed on 01.05.2016.

Ericson, C. (2005). *Real-time collision detection*. Morgan Kaufmann series in interactive 3D technology. Elsevier, Amsterdam and Boston.

Fitzgibbon, K. E. and Veasey, T. J. (1990). Thermally assisted liberation - a review. *Minerals Engineering*, 3(1-2):181–185.

Fließbach, T. (2012). *Elektrodynamik: Lehrbuch zur Theoretischen Physik II*. Springer, Berlin and Heidelberg, 6th edition.

Fuerstenau, D. W. and Abouzeid, A. Z. M. (2002). The energy efficiency of ball milling in comminution. *International Journal of Mineral Processing*, 67(1–4):161–185.

Gebrande, H., Kern, H., and Rummel, F. (1982). Elasticity and inelasticity. In Angenheister, G., editor, *SpringerMaterials - The Landolt-Börnstein Database: Physical Properties of Rocks*, pages 1–233. Springer, Berlin.

Gibert, B. and Mainprice, D. (2009). Effect of crystal preferred orientations on the thermal diffusivity of quartz polycrystalline aggregates at high temperature. *Tectonophysics*, 465(1-4):150–163.

Goranson, R. W. (1942). Heat capacity; heat of fusion. In *Handbook of Physical Constants*, volume 36 of *Geological Society of America Special Papers*, pages 223–242. Geological Society of America.

Gupta, M. and Wong, W. L. (2007). *Microwaves and metals*. John Wiley & Sons, Singapore and Hoboken.

Haque, K. E. (1999). Microwave energy for mineral treatment processes-a brief review. *International Journal of Mineral Processing*, 57:1–24.

Hartlieb, P., Leindl, M., Kuchar, F., Antretter, T., and Moser, P. (2012). Damage of basalt induced by microwave irradiation. *Minerals Engineering*, 31:82–89.

Hartlieb, P., Toifl, M., Kuchar, F., Meisels, R., and Antretter, T. (2016). Thermo-physical properties of selected hard rocks and their relation to microwave-assisted comminution. *Minerals Engineering*, 91:34–41.

Hassani, F., Nekoovaght, P. M., and Gharib, N. (2016). The influence of microwave irradiation on rocks for microwave-assisted underground excavation. *Journal of Rock Mechanics and Geotechnical Engineering*, 8(1):1–15.

Heaney, P. J. (1994). Structure and chemistry of the low-pressure silica polymorphs. In Heaney, P. J., Prewitt, C. T., and Gibbs, G. V., editors, *Silica*, volume 29 of *Reviews in mineralogy*, pages 1–40. Mineralogical Society of America, Washington D.C.

Hearmon, R. F. S. (1984). The elastic constants of crystals and other anisotropic materials. In Hellwege , K. H. and Hellwege, A. M., editors, *Landolt-Börnstein Tables, III/l8*, pages 1–154. Springer, Berlin.

Horai, K.-I. and Baldridge, S. (1972). Thermal conductivity of nineteen igneous rocks, I application of the needle probe method to the measurement of the thermal conductivity of rock. *Physics of the Earth and Planetary Interiors*, 5:151–156.

Hustrulid, W. A., McCarter, M. K., and Van Zyl, Dirk J. A (2001). *Slope Stability in Surface Mining*. Society for Mining, Metallurgy & Exploration, Incorporated, Littleton.

Ishii, T. K. (1995). *Handbook of microwave technology: Volume 2, Applications*. Academic Press, San Diego.

Jackson, J. D. (2011). *Klassische Elektrodynamik*. de Gruyter, Berlin and New York, 4th edition.

Jerby, E., Meir, Y., and Faran, M. (2013). Basalt melting by localized-microwave thermal-runaway instability. In *14 th International Conference on Microwave and High Frequency Heating, AMPERE-2013*, pages 255–258.

Jones, D. A., Kingman, S. W., Whittles, D. N., and Lowndes, I. (2005). Understanding microwave assisted breakage. *Minerals Engineering*, 18(7):659–669.

Jones, D. A., Kingman, S. W., Whittles, D. N., and Lowndes, I. S. (2007). The influence of microwave energy delivery method on strength reduction in ore samples. *Chemical Engineering and Processing: Process Intensification*, 46(4):291–299.

Jones, D. A., Lelyveld, T. P., Mavrofidis, S. D., Kingman, S. W., and Miles, N. J. (2002). Microwave heating applications in environmental engineering-a review. *Resources, Conservation and Recycling*, 34:75–90.

Kingman, S. W. (2006). Recent developments in microwave processing of minerals. *International Materials Reviews*, 51(1):1–12.

Kingman, S. W., Jackson, K., Bradshaw, S. M., Rowson, N. A., and Greenwood, R. (2004a). An investigation into the influence of microwave treatment on mineral ore comminution. *Powder Technology*, 146(3):176–184.

Kingman, S. W., Jackson, K., Cumbane, A., Bradshaw, S. M., Rowson, N. A., and Greenwood, R. (2004b). Recent developments in microwave-assisted comminution. *International Journal of Mineral Processing*, 74(1-4):71–83.

Kingman, S. W., Vorster, W., and Rowson, N. A. (2000). The influence of mineralogy on microwave assisted grinding. *Minerals Engineering*, 13(3):313–327.

Kogelnik, H. and Li, T. (1966). Laser beams and resonators. *Applied Optics*, 5(10):1550–1567.

Kržmanc, M. M., Valant, M., and Suvorov, D. (2003). The dielectric properties of plagioclase feldspars. *Materiali in tehnologije*, (37):13–17.

Lambert, J. and Edwards, A. (2016). http://www.ces.fau.edu/nasa/module-2/radiation-sun.php. Last accessed on 25.02.2016.

Le Chatelier, H. (1889). Sur la dilatation du quartz. *Comptes Rendus de l'Académie des Sciences de Paris*, 108:1046–1049.

Lee, J. and Fenves, G. L. (1998). Plastic-damage model for cyclic loading of concrete structures. *Journal of Engineering Mechanics*, 124:892–900.

Lee, J. and Kim, T. (2011). Application of microwave heating to recover metallic elements from industrial waste. In Grundas, S., editor, *Advances in Induction and Microwave Heating of Mineral and Organic Materials*, pages 303–312. InTech, Rijeka.

Lemaître, J. and Chaboche, J.-L. (1990). *Mechanics of solid materials*. Cambridge University Press, Cambridge.

Lindroth, D. P., Berglund, W. R., Morrell, R. J., and Blair, J. R. (1993). Microwave assisted drilling in hard rock. *Tunnels and Tunnelling International*, 25:24–27.

López-Almansa, F., Alfarah, B., and Oller, S. (2014). Numerical simulation of RC frame testing with damaged plasticity model. Comparison with simplified models. In *Second European conference on earthquake engineering and seismology*, pages 1–12.

Lubliner, J., Oliver, J., Oller, S., and Oñate, E. (1989). A plastic-damage model for concrete. *International Journal of Solids and Structures*, 25(3):299–326.

Maurer, W. (1968). *Novel drilling techniques*. Pergamon Press, London.

McGill, S. L. and Walkiewicz, J. W. (1987). Applications of microwave energy in extractive metallurgy. *Journal of Microwave Power and Electromagnetic Energy*, 22(3):175–177.

Meep (2016). Meep manual. http://ab-initio.mit.edu/wiki/index.php/Meep_manual. Last accessed on 29.04.2016.

Meisels, R. and Kuchar, F. (2007). Density-of-states and wave propagation in two-dimensional photonic crystals with positional disorder. *Journal of Optics A: Pure and Applied Optics*, 9(9):396–402.

Meisels, R., Toifl, M., Hartlieb, P., Kuchar, F., and Antretter, T. (2015). Microwave propagation and absorption and its thermo-mechanical consequences in heterogeneous rocks. *International Journal of Mineral Processing*, 135:40–51.

Meredith, R. J. (1998). *Engineers' handbook of industrial microwave heating*, volume 25 of *IEE power series*. Institution of Electrical Engineers, London.

Meschede, D. (2008). *Optik, Licht und Laser*. Vieweg+Teubner | GWV Fachverlage, Wiesbaden, 3rd edition.

Metaxas, A. C. and Meredith, R. J. (1993). *Industrial microwave heating*, volume 4 of *IEE power engineering series*. Peter Peregrinus, London.

Mikl-Resch, M. J., Antretter, T., Gimpel, M., Kargl, H., Pittino, G., Tichy, R., Ecker, W., and Galler, R. (2015). Numerical calibration of a yield limit function for rock materials by means of the Brazilian test and the uniaxial compression test. *International Journal of Rock Mechanics and Mining Sciences*, 74:24–29.

Moss, G. W. (1999). *Mathematical Models of the Alpha-Beta Phase Transition of Quartz*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg. http://scholar.lib.vt.edu/theses/available/etd-081099-142433/unrestricted/etd.pdf. Last accessed on 16.03.2015.

Napier-Munn, T. J. (1996). *Mineral comminution circuits: Their operation and optimisation*, volume 2 of *JKMRC monograph series in mining and mineral processing*. Julius Kruttschnitt Mineral Research Centre, Indooroopilly.

Nasseri, M. H. B., Mohanty, B., and Robin, P.-Y. F. (2005). Characterization of microstructures and fracture toughness in five granitic rocks. *International Journal of Rock Mechanics and Mining Sciences*, 42(3):450–460.

Nelson, S. O., Lindroth, D. P., and Blake, R. L. (1989). Dielectric properties of selected minerals at 1 to 22 GHz. *GEOPHYSICS*, 54(10):1344–1349.

Okrusch, M. and Matthes, S. (2005). *Mineralogie: Eine Einführung in die spezielle Mineralogie, Petrologie und Lagerstättenkunde*. Springer, Berlin, 7th edition.

Olubambi, P. A., Potgieter, J. H., Hwang, J. Y., and Ndlovu, S. (2007). Influence of microwave heating on the processing and dissolution behaviour of low-grade complex sulphide ores. *Hydrometallurgy*, 89(1-2):127–135.

Osepchuk, J. M. (1984). A history of microwave heating applications. *IEEE Transactions on Microwave Theory and Techniques*, 32:1200–1224.

Oskooi, A. and Johnson, S. G. (2011). Distinguishing correct from incorrect PML proposals and a corrected unsplit PML for anisotropic, dispersive media. *Journal of Computational Physics*, 230(7):2369–2377.

Oskooi, A. and Johnson, S. G. (2013). Electromagnetic wave source conditions. In Taflove, A., Oskooi, A., and Johnson, S. G., editors, *Advances in FDTD computational electrodynamics*, Artech House antennas and propagation library, pages 65–100.

Oskooi, A. F., Roundy, D., Ibanescu, M., Bermel, P., Joannopoulos, J., and Johnson, S. G. (2010). Meep: A flexible free-software package for electromagnetic simulations by the FDTD method. *Computer Physics Communications*, 181(3):687–702.

Peinsitt, T. (2009). Demonstration of the feasibility of weakening rocks with microwave techniques. Master's thesis, Graz University of Technology, Graz.

Peinsitt, T., Kuchar, F., Hartlieb, P., Moser, P., Kargl, H., Restner, U., and Sifferlinger, N. A. (2010). Microwave heating of dry and water saturated basalt, granite and sandstone. *International Journal of Mining and Mineral Engineering*, 2(1):18–29.

Peselnick, L. and Meister, R. (1965). Variational method of determining effective moduli of polycrystals: (a) hexagonal symmetry, (b) trigonal symmetry. *Journal of Applied Physics*, 36(9):2879.

Pickles, C. A. (2009). Microwaves in extractive metallurgy: Part 1 – Review of fundamentals. *Minerals Engineering*, 22(13):1102–1111.

Pitteri, M. (2015). On the elasticities of quartz across the α - β phase transformation. *Mathematics and Mechanics of Solids*, 20(4):433–460.

Prokopenko, A. (2011). Microwave heating for emolliating and fracture of rocks. In Grundas, S., editor, *Advances in Induction and Microwave Heating of Mineral and Organic Materials*, pages 313–338. InTech, Rijeka.

Quey, R., Dawson, P. R., and Barbe, F. (2011). Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing. *Computer Methods in Applied Mechanics and Engineering*, 200(17-20):1729–1745.

Roy, R. F., Beck, A. E., and Touloukian, Y. S. (1981). Thermophysical properties of rocks. In Touloukian, Y. S., Judd, W. R., and Roy, R. F., editors, *Physical properties of rocks and minerals*, volume II-2 of *McGraw-Hill/CINDAS data series on material properties : Group II, Properties of special materials*, pages 409–502. McGraw-Hill, New York.

RSoft (2014). Rsoft. http://optics.synopsys.com/. Last accessed on 29.04.2016.

Sahyoun, C., Rowson, N. A., Kingman, S. W., Groves, L., and Bradshaw, S. M. (2005). The influence of microwave pretreatment on copper flotation. *The Journal of The South African Institute of Mining and Metallurgy*, 105:7–14.

Salsman, J. B., Williamson, R. L., Tolley, W. K., and Rice, D. A. (1996). Short-pulse microwave treatment of disseminated sulfide ores. *Minerals Engineering*, 9(1):43–54.

Samouhos, M., Taxiarchou, M., Hutcheon, R., and Devlin, E. (2012). Microwave reduction of a nickeliferous laterite ore. *Minerals Engineering*, 34:19–29.

Santamarina, J. C. (1989). Rock excavation with microwaves: a literature review. In Kulhawy, F. H., editor, *Foundation Engineering: Current Principles and Practices*, volume 1, pages 459–473, Evanston. American Society of Civil Engineers.

Satish, H., Ouellet, J., Raghavan, V., and Radziszewski, P. (2006). Investigating microwave assisted rock breakage for possible space mining applications. *Mining Technology*, 115(1):34–40.

Saxena, S. K., Chatterjee, N., Fei, Y., and Shen, G. (1993). *Thermodynamic Data on Oxides and Silicates*. Springer, Berlin and Heidelberg.

Scheck, F. (2006). *Theoretische Physik*. Springer, Berlin, 2nd edition.

Schwab, A. J. (2013). *Begriffswelt der Feldtheorie*. Springer, Berlin and Heidelberg.

Scott, G., Bradshaw, S. M., and Eksteen, J. J. (2008). The effect of microwave pretreatment on the liberation of a copper carbonatite ore after milling. *International Journal of Mineral Processing*, 85(4):121–128.

Skinner, B. J. (1966). Section 6: Thermal expansion. In Clark, S. P. JR., editor, *Handbook of Physical Constants*, volume 97 of *Geological Society of America Memoirs*, pages 75–96. Geological Society of America.

Steland, A. (2013). *Basiswissen Statistik*. Springer Spektrum, Berlin and Heidelberg.

Taflove, A. (1988). Review of the formulation and applications of the finite-difference time-domain method for numerical modeling of electromagnetic wave interactions with arbitrary structures. *Wave Motion*, 10:547–582.

Taflove, A. (2005). *Computational electrodynamics: The finite-difference time-domain method, Third Edition*. Artech House, Boston and London.

Toifl, M., Hartlieb, P., Meisels, R., Antretter, T., and Kuchar, F. (2016a). Numerical study of the influence of irradiation parameters on the microwave-induced stresses in granite for industrial applications. In Wills, B. A., editor, *Comminution 16*. Minerals Engineering International, Cape Town.

Toifl, M., Meisels, R., Hartlieb, P., Kuchar, F., and Antretter, T. (2014). Microwave absorption and its thermo-mechanical consequences in heterogeneous rocks. In Soga, K., Kumar, K., Biscontin, G., and Kuo, M., editors, *Geomechanics from Micro to Macro*, pages 1545–1550. Crc Press, Leiden.

Toifl, M., Meisels, R., Hartlieb, P., Kuchar, F., and Antretter, T. (2015a). 3D numerical study on microwave induced stresses in inhomogeneous hard rocks. In Wills, B. A., editor, *Computational Modelling 15*. Minerals Engineering International, Falmouth.

Toifl, M., Meisels, R., Hartlieb, P., Kuchar, F., and Antretter, T. (2015b). Influence of the microstructure on rock damage during microwave irradiation: a 3D numerical analysis. In Schubert, W. and Kluckner, A., editors, *Future development of rock mechanics*, pages 731–736, Salzburg. Österreichische Gesellschaft für Geomechanik.

Toifl, M., Meisels, R., Hartlieb, P., Kuchar, F., and Antretter, T. (2016b). 3D numerical study on microwave induced stresses in inhomogeneous hard rocks. *Minerals Engineering*, 90:29–42.

Tromans, D. (2008). Mineral comminution: Energy efficiency considerations. *Minerals Engineering*, 21(8):613–620.

Umashankar, K. R. (1988). Numerical analysis of electromagnetic wave scattering and interaction based on frequency-domain integral equation and method of moments techniques. *Wave Motion*, 10(6):493–525.

Vaughan, M. T. and Guggenheim, S. (1986). Elasticity of muscovite and its relationship to crystal structure. *Journal of Geophysical Research*, (91):4657–4664.

Vorster, W., Rowson, N. A., and Kingman, S. W. (2001). The effect of microwave radiation upon the processing of Neves Corvo copper ore. *International Journal of Mineral Processing*, 63:29–44.

Walkiewicz, J. W., Clark, A. E., and McGill, S. L. (1991). Microwave-assisted grinding. *IEEE Transactions on Industry Applications*, 27:239–243.

Walkiewicz, J. W., Kazonich, G., and McGill, S. L. (1988). Microwave heating characteristics of selected minerals and compounds. *Mineral and Metallurgical Processing*, 5:39–42.

Wang, G., Radziszewski, P., and Ouellet, J. (2008). Particle modeling simulation of thermal effects on ore breakage. *Computational Materials Science*, 43(4):892–901.

Wang, Y. (2013). Numerical modelling of inhomogeneous rock breakage behaviour based on texture images. In Wills, B. A., editor, *Computational Modelling '13*. Minerals Engineering International, Falmouth.

Wang, Y. (2015). Numerical modelling of heterogeneous rock breakage behaviour based on texture images. *Minerals Engineering*, 74:130–141.

Wang, Y. and Djordjevic, N. (2014). Thermal stress FEM analysis of rock with microwave energy. *International Journal of Mineral Processing*, 130:74–81.

Webb, W. E. and Church, R. H. (1986). *Measurement of dielectric properties of minerals at microwave frequencies: Report of Investigations 9035*. US Bureau of Mines.

Weiland, T. (1977). Eine Methode zur Lösung der Maxwellschen Gleichungen für sechskomponentige Felder auf diskreter Basis. *AEÜ*, 31(3):116–120.

Whittles, D. N., Kingman, S. W., and Reddish, D. J. (2003). Application of numerical modelling for prediction of the influence of power density on microwave-assisted breakage. *International Journal of Mineral Processing*, 68:71–91.

Yee, K. S. (1966). Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Transactions on Industry Applications*, AP-14(3):302–307.

Zhao, X., Yan, L., and Huang, K. (2011). Review of numerical simulation of microwave heating process. In Grundas, S., editor, *Advances in Induction and Microwave Heating of Mineral and Organic Materials*, pages 27–48. InTech, Rijeka.

Zheng, Y., Wang, S., Feng, J., Ouyang, Z., and Li, X. (2005). Measurement of the complex permittivity of dry rocks and minerals: application of polythene dilution method and Lichtenecker's mixture formulae. *Geophysical Journal International*, 163(3):1195–1202.

Appendix A

Scripts and input files for 2D inhomogeneous models

Abaqus thermal model

Appendix1/2D_inhomogen/j154_01_16_25kW_refined22.inp

```
*Heading
** Thermal model for 2D inhomogeneous material
** Job name: j154_01_16_25kW_refined22
** Generated by: Abaqus/CAE 6.12-1
** Preprint , echo=NO, model=NO, history=NO, contact=NO
** Create the refined part
*Part , name=grid
*Node
1, 0.17 , 0. , 0.
1601, 0.33 , 0. , 0.
2401501, 0.17 , 0.15 , 0.
2403101, 0.33 , 0.15 , 0.
*NGEN, NSET=BOT
1, 1601, 1
*NGEN, NSET=TOP
2401501, 2403101, 1
*NFILL
BOT, TOP, 1500, 1601
*Element , type=DC2D4
1, 1, 2, 1603, 1602
*ELGEN, ELSET=ALL
1, 1600, 1, 1, 1500, 1601, 1600
** Include material distribution of the refined part created by inhom_ref22_01_04.py until inhom_ref22_04_04.py
*INCLUDE, Input=disk_subarea_ref22.inp
*INCLUDE, Input=matrix_subarea_ref22.inp
*Solid Section , elset=disk , material=disk
1. ,
*Solid Section , elset=matrix , material=matrix
1. ,
*End Part
** Create the coarse outer part
*Part , name=outer
*Node
1, 0., 0., 0.
341, 0.17, 0., 0.
661, 0.33, 0., 0.
1001, 0.5, 0., 0.
```

```
299300, 0., 0.1495, 0.
300300, 0.5, 0.1495, 0.
299640, 0.17, 0.1495, 0.
299960, 0.33, 0.1495, 0.
300301, 0., 0.15, 0.
301301, 0.5, 0.15, 0.
601602, 0., 0.3005, 0.
602602, 0.5, 0.3005, 0.
*NGEN, NSET=left_bot
1, 341, 1
*NGEN, NSET=left_top
299300, 299640, 1
*NGEN, NSET=right_bot
661, 1001, 1
*NGEN, NSET=right_top
299960, 300300, 1
*NGEN, NSET=middle
300301, 301301, 1
*NGEN, NSET=top
601602, 602602, 1
*NFILL, NSET=nleft
left_bot, left_top, 299, 1001
*NFILL, NSET=nright
right_bot, right_top, 299, 1001
*NFILL, NSET=ntop
middle, top, 301, 1001
*Element, type=DC2D4
1, 1, 2, 1003, 1002
*Element, type=DC2D4
300001, 300301, 300302, 301303, 301302
*Element, type=DC2D4
661, 661, 662, 1663, 1662
*ELGEN, ELSET=left
1, 340, 1, 1, 300, 1001, 1000
*ELGEN, ELSET=right
661, 340, 1, 1, 300, 1001, 1000
*ELGEN, ELSET=top
300001, 1000, 1, 1, 301, 1001, 1000
** Include material distribution of the refined part created by inhom_mat_creation_outer.py
*INCLUDE, Input=disk_outerarea.inp
*INCLUDE, Input=matrix_outerarea.inp
*Solid Section, elset=disk, material=disk
1.,
*Solid Section, elset=matrix, material=matrix
1.,
*End Part
*Assembly, name=Assembly
*Instance, name=grid-1, part=grid
          0.,               0.,               0.
*End Instance
*Instance, name=outer-1, part=outer
          0.,               0.,               0.
*End Instance
*Nset, nset=FIX, instance=outer-1
1
*ELSET, elset=grid_top, GENERATE,  instance=grid-1
2398401, 2400000, 1
*ELSET, elset=grid_left, GENERATE,  instance=grid-1
1, 2398401, 1600
*ELSET, elset=grid_right, GENERATE,  instance=grid-1
1600, 2400000, 1600
*NSET, Nset=ALL_grid, GENERATE, instance=grid-1
1, 2403101, 1
*SURFACE, name=S_grid_top, Type=element
grid_top, S3
*SURFACE, name=S_grid_left, Type=element
grid_left, S4
*SURFACE, name=S_grid_right, Type=element
grid_right, S2
*ELSET, elset=outer_middle, GENERATE,  instance=outer-1
300341, 300660, 1
*ELSET, elset=outer_left, GENERATE,  instance=outer-1
340, 299340, 1000
*ELSET, elset=outer_right, GENERATE,  instance=outer-1
661, 299661, 1000
```

```
*ELSET, elset=outer_l_bot, GENERATE, instance=outer-1
1, 340, 1
*ELSET, elset=grid_m_bot, GENERATE, instance=grid-1
1, 1600, 1
*ELSET, elset=outer_r_bot, GENERATE, instance=outer-1
661, 1000, 1
*ELSET, elset=outer_l, GENERATE, instance=outer-1
1, 600001, 1000
*ELSET, elset=outer_r, GENERATE, instance=outer-1
1000, 601000, 1000
*ELSET, elset=outer_top, GENERATE, instance=outer-1
600001, 601000, 1
*NSET, Nset=ALL_outer, instance=outer-1
nright
nleft
ntop
*SURFACE, name=S_outer_top, Type=element
outer_middle, S1
*SURFACE, name=S_outer_left, Type=element
outer_left, S2
*SURFACE, name=S_outer_right, Type=element
outer_right, S4
*SURFACE, name=S_bound, Type=element
outer_l_bot, S1
grid_m_bot, S1
outer_r_bot, S1
*SURFACE, name=TBC, Type=element
outer_l, S4
outer_top, S3
outer_r, S2
*TIE, name=tie_top
S_grid_top, S_outer_top
*TIE, name=tie_left
S_grid_left, S_outer_left
*TIE, name=tie_right
S_grid_right, S_outer_right
*End Assembly
** Include material files
*INCLUDE, input=quartz.inp
*INCLUDE, input=plagioclase.inp
*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23
*Boundary
FIX, 1, 1
FIX, 2, 2
FIX, 6, 6
*Initial Conditions, type=TEMPERATURE
ALL_grid, 25.
ALL_outer, 25.
*Step, name=Heating
*Heat Transfer, end=PERIOD, deltmx=50.
2., 15., 0.0006, 15.
*Dflux
*INCLUDE, Input=dflux_sub_ref22.inp
*INCLUDE, Input=dflux_outer_ref22.inp
*Sfilm
S_bound, F , 25., 20.
*Sradiate
TBC, R, 25., 0.8
S_bound, R, 25., 0.8
*Restart, write, frequency=0
*Output, field
*Element output
TEMP, IVOL, HFL
*Node output
NT
*Output, history, variable=PRESELECT
*End Step
*Step, name=Cooling
*Heat Transfer, end=PERIOD, deltmx=200.
10., 3600., 0.036, 3600.
*Dflux, op=NEW
*Sfilm
S_bound, F , 25., 20.
*Sradiate
TBC, R, 25., 0.8
```

```
S_bound, R, 25., 0.8
*Restart, write, frequency=0
*Output, field
*Element output
TEMP, IVOL, HFL
*Node output
NT
*Output, history, variable=PRESELECT
*End Step
```

Python script to create material distribution in the refined part (1 of 4)

Appendix1/2D_inhomogen/inhom_ref22_01_04.py

```
######################################################################
# Script to generate the material distribution in the refined part
# Script 1 from 4 to perform parallel computations
######################################################################
#import
import math
#
######################################################################
#initialization
a=[]  #field where the lines of the file containing geometric information are saved
b=[]  #field with geometric information of discs
d=[]  #distance of centre of FE to the origin of the coordinate system
y=0
z=0
c=0
e=0
g=0
j=0
l=1
zz=0
cb=0
output=[] #Temp file for output
quadin=[] #Elements which are within discs
inlist=False  #Check if element is already saved
floats=[] #Temp file to split field a
######################################################################
#input
inp=raw_input('Insert name of file that contains the geometric information: ')
# In this thesis: 'gen8'
matrix=raw_input('Insert name for the matrix element set: ')
# In this thesis: 'matrix'
disk=raw_input('Insert name for the discs element set: ')
# In this thesis: 'disk'
######################################################################
#main
dat=open(inp+'.txt','r')    #open file containing geometric information
for line in dat:         #loop over all entries
  a.append(line.rstrip())   #save each line in field a
dat.close()            #close file
del a[len(a)-1]          #delete empty line
while y<len(a):          #loop over all entries in field a
  if a[y][0]==" ":       #if a double space entry exists replace it by only one
    a[y]=a[y].replace(' ', '',1)
  #Split line
  floats=[float(x) for x in ((a[y].replace("  "," ")).replace(";","")).split(" ")]
  b.append(floats)       #save splinted line in b field
  #first entry of b is the x coordinate of centre of disc in cm
  #second entry of b is the y coordinate of centre of disc in cm
  #third entry of b is the radius of the disc in cm
  y=y+1
f=open('disk_01_04.inp','w')  #open output file for discs elements
f.write("*ELSET, Elset="+disk+", instance=grid-1\n")
k=open('matrix_01_04.inp','w')  #open output file for matrix elements
k.write("*ELSET, Elset="+matrix+", instance=grid-1\n")
#Check which finite elements are inside disc and which outside
```

```
while z<len(b):          #loop over all discs
  Mx=24.975+b[z][0]      #middle point of current disc in cm, attention: change in coordinate system
  # check if disc is in the refined part
  if (16.75<Mx<33.25) and (b[z][1]>0 and b[z][1]<4.0):
    #Calculate finite element in which middle point lies
    nx=int((Mx-16.975)/0.01)+1
    ny=int(b[z][1]/0.01)
    #Calculate amount of elements within radius of disc
    cv=int(b[z][2]/0.01)+2
    #Check if boundaries of the refined part are reached or exceeded
    if (nx-cv)>1:
      cx=nx-cv       #element at the left quadrant of the disc
    else:
      cx=1         #element at the left quadrant of the disc
    cb=cx
    #Change this block for script 02 until 04
    if (ny-cv)>1:
      cy=ny-cv       #element at the bottom quadrant of the disc
    else:
      cy=1         #element at the bottom quadrant of the disc
    #
    if (nx+cv)<1600:
      borderx=nx+cv    #element at the right quadrant of the disc
    else:
      borderx=1600     #element at the right quadrant of the disc
    #Change this block for script 02 until 04
    if (ny+cv)<375:
      bordery=ny+cv    #element at the top quadrant of the disc
    else:
      bordery=375      #element at the top quadrant of the disc
    #
    while cy<=bordery:     #loop over all elements between left and right quadrant of the disc
      while cx<=borderx:  #loop over all elements between bottom and top quadrant of the disc
        #calculate distance of the middle point of the current FE regarding the origin of the coordinate system
        d=[[(cx-1)*0.01,(cy-1)*0.01],[cx*0.01,(cy-1)*0.01],[cx*0.01,(cy)*0.01]]
        #Check if middle point of the current FE is within disc
        if (math.pow((((d[1][0]+d[0][0])/2)-(Mx-16.975)),2)+math.pow((((d[2][1]+d[1][1])/2)-b[z][1]),2))<=(math.pow((b[z
    ][2]),2)):
          quadin.append([cx,cy])  #if yes add the amount of elements in y and x direction of the current FE
        cx=cx+1
      cx=cb
      cy=cy+1
  z=z+1
#all finite elements within discs are written to the output file
while e<len(quadin):
  output.append(quadin[e][0]+(quadin[e][1]-1)*1600)
  e=e+1
#write all finite elements within discs in outputfile
while j<len(output):
  if ((output[j]/1600)>=0 and (output[j]/1600)<=374):
    f.write(str(output[j])+"\n")
  j=j+1
#All remaining elements which are in the outer part are written to the matrix outputfile
while l<=600000:
  if (l/1600>=0 and l/1600<=374):
    m=0
    while m<len(output):
      if l==output[m]:
        inlist=True
        m=len(output)+1
      m=m+1
    if inlist==False:
      k.write(str(l))
      k.write("\n")
  l=l+1
  inlist=False
f.close()
k.close()
```

Python script to create material distribution in the coarse part

Appendix1/2D_inhomogen/inhom_mat_creation_outer.py

```
########################################################################
# Script to generate the material distribution in the outer part
# Node distance in outer part: 0.5 mm
########################################################################
#load packages
import math
#
########################################################################
#initialization
a=[]  #field where the lines of the file containing geometric information are saved
b=[]  #field with geometric information of discs
d=[]  #distance of centre of FE to the origin of the coordinate system
y=0
z=0
c=0
e=0
g=0
j=0
l=1
zz=0
cb=0
output=[] #Temp file for output
quadin=[] #Elements which are within discs
inlist=False  #Check if element is already saved
floats=[] #Temp file to split field a
########################################################################
#input
inp=raw_input('Insert name of file that contains the geometric information: ')
# In this thesis: 'gen8'
matrix=raw_input('Insert name for the matrix element set: ')
# In this thesis: 'matrix'
disk=raw_input('Insert name for the discs element set: ')
# In this thesis: 'disk'
########################################################################
#main
dat=open(inp+'.txt','r')  #open file containing geometric information
for line in dat:      #loop over all entries
  a.append(line.rstrip()) #save each line in field a
dat.close()          #close file
del a[len(a)-1]       #delete empty line
while y<len(a):       #loop over all entries in field a
  if a[y][0]==" ":    #if a double space entry exists replace it by only one
    a[y]=a[y].replace(' ', '',1)
  #Split line
  floats=[float(x) for x in ((a[y].replace("  "," ")).replace(";","")).split(" ")]
  b.append(floats)    #save splinted line in b field
  #first entry of b is the x coordinate of centre of disc in cm
  #second entry of b is the y coordinate of centre of disc in cm
  #third entry of b is the radius of the disc in cm
  y=y+1
f=open('disk_outerarea.inp','w')  #open output file for discs elements
f.write("*ELSET, Elset="+disk+", instance=grid-1\n")
k=open('matrix_outerarea.inp','w')  #open output file for matrix elements
k.write("*ELSET, Elset="+matrix+", instance=grid-1\n")
#Check which finite elements are inside disc and which outside
while z<len(b):   #loop over all discs
  Mx=24.975+b[z][0] #middle point of current disc in cm, attention: change in coordinate system
  # check if disc is in the outer part
  if (b[z][1]<6.77 and Mx<=21.205 or Mx>=28.745) or (b[z][1]>=6.77):
    #Calculate finite element in which middle point lies
    nx=int(Mx/0.05)+1
    ny=int(b[z][1]/0.05)
    #Calculate amount of elements within radius of disc
    cv=int(b[z][2]/0.05)+2
    #Check if boundaries of the outer part are reached or exceeded
    if (nx-cv)>0:
      cx=nx-cv    #element at the left quadrant of the disc
    else:
      cx=1      #element at the left quadrant of the disc
    cb=cx
    if (ny−cv)>0:
      cy=ny−cv     #element at the bottom quadrant of the disc
    else:
      cy=0       #element at the bottom quadrant of the disc
    if (nx+cv)<1000:
      borderx=nx+cv #element at the right quadrant of the disc
    else:
      borderx=1000  #element at the right quadrant of the disc
    if (ny+cv)<600:
      bordery=ny+cv #element at the top quadrant of the disc
    else:
      bordery=600   #element at the top quadrant of the disc
    while cy<=bordery:  #loop over all elements between left and right quadrant of the disc
      while cx<=borderx: #loop over all elements between bottom and top quadrant of the disc
        #calculate distance of the middle point of the current FE regarding the origin of the coordinate system
        d=[[(cx−1)*0.05,(cy)*0.05],[cx*0.05,(cy)*0.05],[cx*0.05,(cy+1)*0.05]]
        #Check if middle point of the current FE is within disc
        if (math.pow((((d[1][0]+d[0][0])/2)−Mx),2)+math.pow((((d[2][1]+d[1][1])/2)−b[z][1]),2))<=(math.pow((b[z][2]),2)):
          quadin.append([cx,cy])  #if yes add the amount of elements in y and x direction of the current FE
        cx=cx+1
      cx=cb
      cy=cy+1
  z=z+1
#all finite elements within discs are written to the output file
while e<len(quadin):  #loop over all elements inside discs
  output.append(quadin[e][0]+quadin[e][1]*1000)
  e=e+1
#write all finite elements within discs in output file
while j<len(output):
  if ((l/1000)<140 and (l%1000<=420 or l%1000>=581)) or (l/1000>=140):
    f.write(str(output[j])+"\n")
  j=j+1
#All remaining elements which are in the outer part are written to the matrix output file
while l<=601000:
  if ((l/1000)<140 and (l%1000<=420 or l%1000>=581)) or (l/1000>=140):
    m=0
    while m<len(output):
      if l==output[m]:
        inlist=True
        m=len(output)+1
      m=m+1
    if inlist==False:
      k.write(str(l))
      k.write("\n")
  l=l+1
  inlist=False
f.close()
k.close()
```

Python script to create body heat flux in each FE in the refined part

Appendix1/2D_inhomogen/script_subarea_refined22.py

```
########################################################################
# Script to generate the body heat flux in each finite element of the refined part
########################################################################
#load packages
import ast
import json
########################################################################
#Variables that have to be defined
#
oline=4   #Amount of first lines that should be omitted
flux=2E8  #default value of the global heat flux
instance="grid−1" #Name of the instance
Nset="BF" #Element set on which DFLUX is applied
#
#########################################################################
```

```
#Initialising
a=[0]        #save the absorbed power density file
b=[]         #2D field of the absorbed power density of each FDTD grid point
i=0
n=0
y=0
floats=[0]
c=1
d=1
r=0
e=1
inp=''
z=1
zz=1
########################################################################
#ask for data
inp=raw_input('Insert the lines which should be omitted: ')
oline=int(inp)  #save the amount of lines which should be omitted on int variable
# In this thesis: 4
inp=raw_input('Insert body flux: ')
# In this thesis: 847000000
flux=float(inp) #save the constant value C which scales all Pabs values on float variable
instance=raw_input('Insert name of instance: ')
# In this thesis: grid-1
Nset=raw_input('Insert name of element set for body flux: ')
# In this thesis: BF
name=raw_input('Insert the name for the new input file: ')
# In this thesis: dflux_sub_ref22.inp
########################################################################
dat=open('input.fld','r') #open file where the absorbed power density of each FDTD point is saved
for line in dat:       #loop over file
  a.append(line.rstrip()) #save each line in field a
dat.close()          #close file
while n<=(oline):      #loop over line which should be omitted
  del a[0]          #delete line
  n=n+1
while y<len(a):        #loop over all lines of field a
  floats=[float(x) for x in a[y].split()] #split line in single entries
  b.append(floats)     #add each absorbed power density value in 2D field
  y=y+1
f=open(name,'w')       #open output file
f.write("**"+str(flux)+"\n")#write the constant value C which scales the Pabs values
while c<=len(b[c]):      #loop over all lines of absorbed power density values
  if c<=300:          #only considers FDTD grids inside refined area
    while d<=len(b):  #loop over all entries of each line
      if 341<=d<=660: #only considers FDTD grids inside refined area
        while z<=5: #consider a refinement of the FE mesh to the FDTD grid in the refined part by a factor 5
          while zz<=5:
            #write corresponding DFLUX value of each FE to the output file
            f.write(instance)
            f.write(".")
            f.write(str(((c-1)*5+z-1)*1600+(d-341)*5+zz))
            f.write(", ")
            f.write(Nset)
            f.write(", ")
            r=b[d-1][c-1]*flux
            f.write(str(r))
            f.write("\n")
            zz=zz+1
          zz=1
          z=z+1
      z=1
      d=d+1
      e=e+1
  d=1
  c=c+1
f.close
```

Python script to create body heat flux in each FE in the coarse part

Appendix1/2D_inhomogen/script_outerarea_refined22.py

```
#########################################################################
# Script to generate the body heat flux in each finite element of the coarse part
########################################################################
#load packages
import ast
import json
########################################################################
#Variables that have to be defined
#
oline=4   #Amount of first lines that should be omitted
flux=2E8  #default value of the global heat flux
instance="grid-1" #Name of the instance
Nset="BF" #Element set on which DFLUX is applied
#
#########################################################################
#initialization
a=[0]        #save the absorbed power density file
b=[]         #2D field of the absorbed power density of each FDTD grid point
i=0
n=0
y=0
floats=[0]
c=1
d=1
r=0
e=1
inp=''
z=1
zz=1
#########################################################################
#ask for data
inp=raw_input('Insert the lines which should be omitted: ')
oline=int(inp)  #save the amount of lines which should be omitted on int variable
# In this thesis: 4
inp=raw_input('Insert body flux: ')
# In this thesis: 847000000
flux=float(inp) #save the constant value C which scales all Pabs values on float variable
instance=raw_input('Insert name of instance: ')
# In this thesis: outer-1
Nset=raw_input('Insert name of element set for body flux: ')
# In this thesis: BF
name=raw_input('Insert the name for the new input file: ')
# In this thesis: dflux_outer_ref22.inp
#########################################################################
dat=open('input.fld','r')   #open file where the absorbed power density of each FDTD point is saved
for line in dat:        #loop over file
  a.append(line.rstrip())   #save each line in field a
dat.close()           #close file
while n<=(oline):       #loop over line which should be omitted
  del a[0]          #delete line
  n=n+1
while y<len(a):         #loop over all lines of field a
  floats=[float(x) for x in a[y].split()] #split line in single entries
  b.append(floats)      #add each absorbed power density value in 2D field
  y=y+1
f=open(name,'w')        #open output file
f.write("**"+str(flux)+"\n")  #write the constant value C which scales the  Pabs values
while c<=len(b[c]):       #loop over all lines of absorbed power density values
  if c<=300:          #only considers FDTD grids inside coarse area
    while d<=len(b):    #loop over all lines of absorbed power density values
      if d<=340 or d>660: #only considers FDTD grids inside coarse area
        #write corresponding DFLUX value of each FE to the output file
        f.write(instance)
        f.write(".")
        f.write(str((c-1)*1000+d))
        f.write(", ")
        f.write(Nset)
        f.write(", ")
        r=b[d-1][c-1]*flux
        f.write(str(r))
        f.write("\n")
      d=d+1
    d=1
  else:
    while d<=len(b):     #only considers FDTD grids inside coarse area
        #write corresponding DFLUX value of each FE to the output file
        f.write(instance)
        f.write(".")
        f.write(str((c-1)*1000+d))
        f.write(", ")
        f.write(Nset)
        f.write(", ")
        r=b[d-1][c-1]*flux
        f.write(str(r))
        f.write("\n")
        d=d+1
    d=1
  c=c+1
f.close
```

Python script to calculate the thermal energy of the entire model

Appendix1/2D_inhomogen/script_calc_heat_global_154_01_16.py

```
######################################################################
# Script to calculate the thermal energy in the entire model
######################################################################
#load packages
import numpy
#from abaqus import *
from odbAccess import *
######################################################################
#initialization
adg=[]
bdg=[]
cdg=[]
gdg=[]
hdg=[]
ddg=[]
i_dg=[]
kdg=[]
tolddg=[]
amg=[]
bmg=[]
cmg=[]
gmg=[]
hmg=[]
dmg=[]
i_mg=[]
kmg=[]
toldmg=[]
ado=[]
bdo=[]
cdo=[]
gdo=[]
hdo=[]
ddo=[]
i_do=[]
kdo=[]
tolddo=[]
amo=[]
bmo=[]
cmo=[]
gmo=[]
hmo=[]
dmo=[]
i_mo=[]
kmo=[]
```

```
toldmo=[]
kk=0
ll=[]
minus=0
tin=25.             #initial temperature
time=15.            #total time of heating
eges=0.
density_matrix=2703.     #density of matrix material
density_disk=2649.       #density of disc material
frame=1
timeold=0.
e35=0.
#
####################################################################
#method to interpolate the cp value
def getcp (temp, cptable):
  z=1
  if temp<=cptable[0][1]:
    cp=cptable[0][0]
  else:
    if temp>=cptable[len(cptable)-1][1]:
      cp=cptable[len(cptable)-1][0]
    else:
      while z<len(cptable):
        if cptable[z-1][1]<=temp<=cptable[z][1]:
          cp=cptable[z-1][0]+(temp-cptable[z-1][1])*((cptable[z][0]-cptable[z-1][0])/(cptable[z][1]-cptable[z-1][1]))
          z=len(cptable)
        z=z+1
  return cp
#calculate thermal energy of each increment
def calcE (x, cp, d, tin, den):
  if len(tin)==0:
    en=((d[1][x]-25)*cp*d[2][x]*den)
  else:
    en=((d[1][x]-tin[x])*cp*d[2][x]*den)
  return en
#Method to calculate the average HFL on the boundary elements
def avgHFL (t, time, index):
  u=[[],[]]
  zz=0
  while zz<len(t[0]):
    if 1<=index<=2:
      if 1<=t[0][zz]<=800:
        x=(-((t[1][zz][1,]+t[1][zz+1][1,])/2))
        u[0].append(t[0][zz])
        u[1].append(x*0.0001*time)
      else:
        if t[0][zz]>800:
          zz=len(t[0])
    else:
      if 2<index<=4:
        if (1<=t[0][zz]<=420) or (581<=t[0][zz]<=1000):
          x=(-((t[1][zz][1,]+t[1][zz+1][1,])/2))
          u[0].append(t[0][zz])
          u[1].append(x*0.0005*time)
        else:
          if t[0][zz]>1000:
            zz=len(t[0])
    zz=zz+4
  return u
#main
odb = openOdb(path='j154_01_16_25kW_refined22.odb')        #open odb file
disk_grid=odb.rootAssembly.instances['GRID-1'].elementSets['DISK']     #load material sets
matrix_grid=odb.rootAssembly.instances['GRID-1'].elementSets['MATRIX']
disk_outer=odb.rootAssembly.instances['OUTER-1'].elementSets['DISK']
matrix_outer=odb.rootAssembly.instances['OUTER-1'].elementSets['MATRIX']
material_disk_cp=odb.materials['DISK'].specificHeat        #load specific heat data
material_matrix_cp=odb.materials['MATRIX'].specificHeat
cptable_disk=material_disk_cp.table
cptable_matrix=material_matrix_cp.table
lastFrame=odb.steps['Heating'].frames[-1]
f=open("energy_j154_01_16_25kW_refined22_con_08_09_2014.txt",'w')
f.write("Increment  Heating [J]  Flux over boundaries [J] Energy [J]  \n")
#loop over all frames in the first step
while frame<=lastFrame.frameId:
```

```
yzmg=0
yxmg=0
yhmg=0
yzdg=0
yxdg=0
yhdg=0
yzmo=0
yxmo=0
yhmo=0
yzdo=0
yxdo=0
yhdo=0
einc=0
minusinc=0
curFrame = odb.steps['Heating'].frames[frame]
temp=curFrame.fieldOutputs['TEMP']
vol=curFrame.fieldOutputs['IVOL']
heat=curFrame.fieldOutputs['HFL']
timec=curFrame.frameValue
time=timec-timeold
field_matrix_grid=temp.getSubset (region=matrix_grid, position=INTEGRATION_POINT, elementType='DC2D4')
fieldvol_matrix_grid=vol.getSubset (region=matrix_grid, position=INTEGRATION_POINT, elementType='DC2D4')
fieldHFL_matrix_grid=heat.getSubset (region=matrix_grid, position=INTEGRATION_POINT, elementType='DC2D4')
fieldValues_matrix_grid=field_matrix_grid.values
fieldvolVal_matrix_grid=fieldvol_matrix_grid.values
fieldHFLVal_matrix_grid=fieldHFL_matrix_grid.values
field_disk_grid=temp.getSubset (region=disk_grid, position=INTEGRATION_POINT, elementType='DC2D4')
fieldvol_disk_grid=vol.getSubset (region=disk_grid, position=INTEGRATION_POINT, elementType='DC2D4')
fieldHFL_disk_grid=heat.getSubset (region=disk_grid, position=INTEGRATION_POINT, elementType='DC2D4')
fieldValues_disk_grid=field_disk_grid.values
fieldvolVal_disk_grid=fieldvol_disk_grid.values
fieldHFLVal_disk_grid=fieldHFL_disk_grid.values
field_matrix_outer=temp.getSubset (region=matrix_outer, position=INTEGRATION_POINT, elementType='DC2D4')
fieldvol_matrix_outer=vol.getSubset (region=matrix_outer, position=INTEGRATION_POINT, elementType='DC2D4')
fieldHFL_matrix_outer=heat.getSubset (region=matrix_outer, position=INTEGRATION_POINT, elementType='DC2D4')
fieldValues_matrix_outer=field_matrix_outer.values
fieldvolVal_matrix_outer=fieldvol_matrix_outer.values
fieldHFLVal_matrix_outer=fieldHFL_matrix_outer.values
field_disk_outer=temp.getSubset (region=disk_outer, position=INTEGRATION_POINT, elementType='DC2D4')
fieldvol_disk_outer=vol.getSubset (region=disk_outer, position=INTEGRATION_POINT, elementType='DC2D4')
fieldHFL_disk_outer=heat.getSubset (region=disk_outer, position=INTEGRATION_POINT, elementType='DC2D4')
fieldValues_disk_outer=field_disk_outer.values
fieldvolVal_disk_outer=fieldvol_disk_outer.values
fieldHFLVal_disk_outer=fieldHFL_disk_outer.values
#loops over disk grid elements
for vdg in fieldValues_disk_grid:
  adg.append(vdg.elementLabel)
  bdg.append(vdg.data)
for wdg in fieldvolVal_disk_grid:
  cdg.append(wdg.data)
for ydg in fieldHFLVal_disk_grid:
  gdg.append(ydg.elementLabel)
  hdg.append(ydg.data)
ddg.append(adg)      #element label
ddg.append(bdg)      #temperature value
ddg.append(cdg)      #IVOL values
i_dg.append(gdg)     #element label
i_dg.append(hdg)     #HFL value
i_dg.append(cdg)     #IVOL value
#loops over matrix grid elements
for vmg in fieldValues_matrix_grid:
  amg.append(vmg.elementLabel)
  bmg.append(vmg.data)
for wmg in fieldvolVal_matrix_grid:
  cmg.append(wmg.data)
for ymg in fieldHFLVal_matrix_grid:
  gmg.append(ymg.elementLabel)
  hmg.append(ymg.data)
dmg.append(amg)
dmg.append(bmg)
dmg.append(cmg)
i_mg.append(gmg)
i_mg.append(hmg)
i_mg.append(cmg)
#loops over disc outer elements
```

```
for vdo in fieldValues_disk_outer:
  ado.append(vdo.elementLabel)
  bdo.append(vdo.data)
for wdo in fieldvolVal_disk_outer:
  cdo.append(wdo.data)
for ydo in fieldHFLVal_disk_outer:
  gdo.append(ydo.elementLabel)
  hdo.append(ydo.data)
ddo.append(ado)
ddo.append(bdo)
ddo.append(cdo)
i_do.append(gdo)
i_do.append(hdo)
i_do.append(cdo)
#loops over matrix outer elements
for vmo in fieldValues_matrix_outer:
  amo.append(vmo.elementLabel)
  bmo.append(vmo.data)
for wmo in fieldvolVal_matrix_outer:
  cmo.append(wmo.data)
for ymo in fieldHFLVal_matrix_outer:
  gmo.append(ymo.elementLabel)
  hmo.append(ymo.data)
dmo.append(amo)
dmo.append(bmo)
dmo.append(cmo)
i_mo.append(gmo)
i_mo.append(hmo)
i_mo.append(cmo)
#loop over all elements of matix grid
print "CP table quartz:"
countcp=0
while countcp<len(cptable_matrix):
        print str(cptable_matrix[countcp][0])+" "+str(cptable_matrix[countcp][1])
  countcp=countcp+1
while yzmg<len(dmg[0]):
  if len(toldmg)==0:
    cpmg=getcp((dmg[1][yzmg]+tin)/2,cptable_matrix)       #calculate current cp value
    print "TEMP avg "+str((dmg[1][yzmg]+tin)/2)
  else:
    cpmg=getcp((dmg[1][yzmg]+toldmg[yzmg])/2,cptable_matrix)  #calculate current cp value
    print "TEMP avg "+str((dmg[1][yzmg]+toldmg[yzmg])/2)
  einc=einc+calcE(yzmg, cpmg, dmg, toldmg, density_matrix)    #calculate energy value of the increment
  yzmg=yzmg+1
#loop over all elements of disc grid
while yzdg<len(ddg[0]):
  if len(tolddg)==0:
    cpdg=getcp((ddg[1][yzdg]+tin)/2,cptable_disk)
  else:
    cpdg=getcp((ddg[1][yzdg]+tolddg[yzdg])/2,cptable_disk)
  einc=einc+calcE(yzdg, cpdg, ddg, tolddg, density_disk)
  yzdg=yzdg+1
#loop over all elements of matrix outer
while yzmo<len(dmo[0]):
  if len(toldmo)==0:
    cpmo=getcp((dmo[1][yzmo]+tin)/2,cptable_matrix)
  else:
    cpmo=getcp((dmo[1][yzmo]+toldmo[yzmo])/2,cptable_matrix)
  einc=einc+calcE(yzmo, cpmo, dmo, toldmo, density_matrix)
  yzmo=yzmo+1
#loop over all elements of disc outer
while yzdo<len(ddo[0]):
  if len(tolddo)==0:
    cpdo=getcp((ddo[1][yzdo]+tin)/2,cptable_disk)
  else:
    cpdo=getcp((ddo[1][yzdo]+tolddo[yzdo])/2,cptable_disk)
  einc=einc+calcE(yzdo, cpdo, ddo, tolddo, density_disk)
  yzdo=yzdo+1
print einc
eges=eges+einc
#delete of old temperature values
del toldmg[:]
del tolddg[:]
del toldmo[:]
del tolddo[:]
```

```
  #save last temperature values of the current increment in told field
  #loop over elements of disc grid
  while yhdg<len(ddg[0]):
    tolddg.append(ddg[1][yhdg])
    yhdg=yhdg+1
  #loop over elements of matrix grid
  while yhmg<len(dmg[0]):
    toldmg.append(dmg[1][yhmg])
    yhmg=yhmg+1
  #loop over elements of disc outer
  while yhdo<len(ddo[0]):
    tolddo.append(ddo[1][yhdo])
    yhdo=yhdo+1
  #loop over elements of matrix outer
  while yhmo<len(dmo[0]):
    toldmo.append(dmo[1][yhmo])
    yhmo=yhmo+1
  kdg=avgHFL(i_dg, time, 1) #calculate averaged HFL of disc grid elements
  kmg=avgHFL(i_mg, time, 2) #calculate averaged HFL of matrix grid elements
  kdo=avgHFL(i_do, time, 3) #calculate averaged HFL of disc outer elements
  kmo=avgHFL(i_mo, time, 4) #calculate averaged HFL of matrix outer elements
  #loop over all elements of matrix grid
  while yxmg<len(kmg[0]):
    if kmg[0][yxmg]==500:
      print kmg[1][yxmg]
    minusinc=minusinc+kmg[1][yxmg]  #sum up HFL to subtract from total energy
    yxmg=yxmg+1
  #loop over all elements of disc grid
  while yxdg<len(kdg[0]):
    if kdg[0][yxdg]==500:
      print kdg[1][yxdg]
    minusinc=minusinc+kdg[1][yxdg]  #sum up HFL to subtract from total energy
    yxdg=yxdg+1
  minus=minus+minusinc
  del adg[:]
  del bdg[:]
  del cdg[:]
  del ddg[:]
  del gdg[:]
  del hdg[:]
  del i_dg[:]
  del amg[:]
  del bmg[:]
  del cmg[:]
  del dmg[:]
  del gmg[:]
  del hmg[:]
  del i_mg[:]
  #loop over all elements of matrix outer
  while yxmo<len(kmo[0]):
    minusinc=minusinc+kmo[1][yxmo]
    yxmo=yxmo+1
  #loop over all elements of disc outer
  while yxdo<len(kdo[0]):
    minusinc=minusinc+kdo[1][yxdo]
    yxdo=yxdo+1
  minus=minus+minusinc
  del ado[:]
  del bdo[:]
  del cdo[:]
  del ddo[:]
  del gdo[:]
  del hdo[:]
  del i_do[:]
  del amo[:]
  del bmo[:]
  del cmo[:]
  del dmo[:]
  del gmo[:]
  del hmo[:]
  del i_mo[:]
  f.write(str(frame)+"    "+str(einc)+"    "+str(minusinc)+"    "+str(einc+minusinc)+"\n")
  frame=frame+1
  timeold=timec
print eges
print minus
print (eges-minus)
f.write("Heating: "+str(eges)+"\n")
f.write("Flux over boundaries: "+str(minus)+"\n")
f.write("Total energy input: "+str(eges-minus))
f.close()
```

Appendix B

Scripts and input files for 3D inhomogeneous two component models

Calculations of electromagnetic, thermal and linear elastic stress fields

Neper script which creates the microstructure cube

Appendix2/poly_28_08.sh

```
#Bash script to start Neper
# poly_74
neper -T -n 30000 -domain cube 0.08, 0.08, 0.08 -o poly_74.tess
#
neper -FM poly_74.tess -gmsh /usr/bin/gmsh -maxff 20 -mloop 3 -rcl 0.9 -o poly_74_mesh.msh -format inp,tess -outdim 3 -
      order 1 -mesh2dalgo fron,mead,dela -mesh3dalgo netg,netg/netg,netg/gmsh,netg/gmne
#
```

Global *Python* script which starts all other programs

Appendix2/3D_poly_model_v2_2.py

```
########################################################################
# This Python script automatically calculates the temperature and stress fields
# 3D two component model
########################################################################
#
#load packages
import sys                              # better than:     import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
from datetime import datetime
os.system('python -V')
start_time = datetime.now()
print('Starting time = '+str(start_time))
```

```
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()            # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic temperaturefield calculation.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',     type=str, required=True, help='Name of the model.')
parser.add_argument('--initialBF',  type=float, required=True, help='Initial constant body heat multiplier.')
parser.add_argument('--abaquspath', type=str, required=True, help='Name of the abaqus executable.')
parser.add_argument('--cpus',       type=int, required=True, help='Number of cpus which shall be used for the Abaqus
     calculation.')
parser.add_argument('--provenergy', type=float, required=True, help='Provided microwave energy.')
parser.add_argument('--tess',    type=str, required=True, help='Name of the tess file containing the grains.')
parser.add_argument('--phfraction',    type=float, required=True, help='Phase fraction of the plagioclase phase.')
parser.add_argument('--amountpoly',    type=int, required=True, help='Amount of polyhedra in the model.')
parser.add_argument('--inpintegrationpoints', type=str, required=True, help='Name of the file containing the coordinates
     of the intergrationpoints.')
parser.add_argument('--inpelementset', type=str, required=True, help='Name of the file containing the elementsets.')
args = parser.parse_args(namespace=parameter)
####################################################################
#initialization
fine=0
count=0
####################################################################
#main
#
print('Start of automatic 3D polyhedron calculation\n')
path_working_dir = os.path.dirname(os.path.abspath(__file__))       # The directory of the current __file__ is cut down
     to the directory without the filename.
print('The current working direcotry is: '+str(path_working_dir))
#create poly distribution
if 0 != os.system("g++ create_poly_distribution_aut.cpp -o create_poly_distribution_aut.exe"):
        sys.exit('Error during create_poly_distribution_aut.cpp compilation')
  fine=1
if fine==0:
  if 0 != os.system("./create_poly_distribution_aut.exe "+str(parameter.phfraction)+" "+str(parameter.model)+" "+str(
     parameter.amountpoly)):
          sys.exit('Error during create_poly_distribution_aut.exe calculation')
    fine=1
  else:
    print ("Phase distribution finished")
#start Meep FDTD calculation
if fine==0:
  if 0 != os.system("source /export/opt/gcc-4.6.4/bin/setvars.sh; source /export/opt/2014-05-12_openmpi-1.8.1-gcc-4.6.4/
     bin/setvars.sh; mpirun ./polyhedron_aut.exe "+str(parameter.tess)+" "+str(parameter.model)+" | tee meep_"+str(
     parameter.model)+".log"):
          sys.exit('Error during meep electric field calculation')
    fine=1
  else:
    print('Meep electric field calculated')
#start hdf5 E2 calculation
if fine==0:
  if 0 != os.system("source /export/opt/gcc-4.6.4/bin/setvars.sh; "+str(path_working_dir)+"/hdf5_aut.exe "+str(parameter.
     model)+" | tee hdf5calc_"+str(parameter.model)+".log"):
          sys.exit('Error during hdf5_aut.exe calculation!')
    fine=1
  else:
    print ('Hdf5 files successfully calculated!')
    os.system("rm ex-*; rm ey-*; rm ez-*")
#calculate absorbed power density
if fine==0:
  if 0 != os.system("source /export/opt/gcc-4.6.4/bin/setvars.sh; "+str(path_working_dir)+"/hdf5_int_v4_aut.exe "+str(
     parameter.model)+" | tee hdf5_int_v4_aut_"+str(parameter.model)+".log"):
          sys.exit('Error during hdf5_int_v4_aut.exe calculation!')
    fine=1
  else:
    print ('Hdf5_int_v4_aut files successfully calculated!')
#assign phases
if fine==0:
  if 0 != os.system(str(parameter.abaquspath)+' python -u phase_assignment_aut.py --model '+str(parameter.model)+' | tee
     phase_assignment_'+str(parameter.model)+'.log'):
          sys.exit('Error during phase_assignment_aut.py calculation!')
    fine=1
  else:
    print ('Phase_assignment_aut.py successfully done!')
#create Abaqus input file
if fine==0:
  if 0 != os.system(str(parameter.abaquspath)+' python -u create_heat_input.py --model '+str(parameter.model)+' | tee
      create_heat_input_'+str(parameter.model)+'.log'):
          sys.exit('Error during create_heat_input.py.')
    fine=1
  else:
    print ('Heat abaqus input file written!')
#create DFLUX subroutine
if fine==0:
  print('Start the creation of the DFLUX usersubroutine with a BF of '+str(parameter.initialBF)+'\n')
  if 0 != os.system(str(parameter.abaquspath)+' python -u create_BF_subroutine.py --model '+str(parameter.model)+' --
      initialBF '+str(parameter.initialBF)+' --nameBF BF_intpoint_server_'+str(parameter.model)+'_'+str(count)+'.f --work
       '+path_working_dir):
          sys.exit('Error during create_BF_subroutine.py.')
    fine=1
  else:
    print('Heat input file generated')
#start first Abaqus heating job
if fine==0:
  print('Start first NT11 heat calculation \n')
  if 0 != os.system(str(parameter.abaquspath)+" job=j101_01_"+str(parameter.model)+"_03_02_1000_poly_intpoint_heat inp=
      j101_01_"+str(parameter.model)+"_03_02_1000_poly_intpoint_heat.inp"+" user=BF_intpoint_server_"+str(parameter.model
      )+'_'+str(count)+".f cpus=1 interactive | tee j101_01_"+str(parameter.model)+"03_02_1000_poly_intpoint_1.log"):
    sys.exit("Error during first NT11 calculation")
    fine=1
    print("NT11 Abaqus job finished and start of heat calculation. \n")
if fine==0:
  if 0 != os.system(str(parameter.abaquspath)+' python -u script_calc_heat_aut.py --model '+str(parameter.model)+' --
      heatout energy_j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat_'+str(count)):
          sys.exit('Error during heat calculation.')
    fine=1
#read in total energy after first loop
if fine==0:
  print('Compare energies')
  f=open('energy_j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat_'+str(count)+'.txt', 'r')
  for line in f:
    line1=line.rstrip()
    if 'Total' in line1:
      temp=line1.split(': ')
      curenergy=float(temp[1])
  f.close()
#check if error in energy is less than 3 percent
while abs((curenergy-parameter.provenergy)/parameter.provenergy)>0.03 and fine==0:
  print('Energydifference= '+str(((curenergy-parameter.provenergy)/parameter.provenergy)*100.)+'% after '+str(count+1)+'.
       thermal calculation is bigger than 3% --> recalculation')
  #calculate new BF
  newBF=parameter.initialBF*(parameter.provenergy/curenergy)
  print('New BF= '+str(newBF))
  count=count+1
  #rewrite DFLUX
  if 0 != os.system(parameter.abaquspath+' python -u create_BF_subroutine.py --model '+str(parameter.model)+' --initialBF
       '+str(newBF)+' --nameBF BF_intpoint_server_'+str(parameter.model)+'_'+str(count)+'.f --work '+path_working_dir):
          sys.exit('Error during create_BF_subroutine.py.')
    fine=1
  if 0 != os.system(parameter.abaquspath+" job=j101_01_"+str(parameter.model)+"_03_02_1000_poly_intpoint inp=j101_01_"+
      str(parameter.model)+"_03_02_1000_poly_intpoint.inp"+" user=BF_intpoint_server_"+str(parameter.model)+'_'+str(count
      )+".f cpus=1 interactive | tee j101_01_"+str(parameter.model)+"_03_02_1000_poly_intpoint_"+str(count)+".log"):
    sys.exit("Error during NT11 calculation")
    fine=1
  if 0 != os.system(parameter.abaquspath+' python -u script_calc_heat_aut.py --model '+str(parameter.model)+' --heatout
      energy_j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat_'+str(count)):
          sys.exit('Error during heat calculation.')
    fine=1
  f=open('energy_j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat_'+str(count)+'.txt', 'r')
  for line in f:
    line1=line.rstrip()
    if 'Total' in line1:
      temp=line1.split(': ')
      curenergy=float(temp[1])
  f.close()
print('Temperaturefield calculation of heating step finished with an error of '+str((curenergy-parameter.provenergy)/
      parameter.provenergy))
#start Abaqus cooling job
```

```
if fine==0:
  os.system('cp j101_01_cooling.inp j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling.inp')
  if 0 != os.system(str(parameter.abaquspath)+' job=j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling inp
      =j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling.inp user=BF_cooling.f cpus='+str(parameter.cpus)
      +' interactive | tee j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling_'+str(parameter.inpNT11cool)
      +'.log'):
    sys.exit("Error during cooling NT11 calculation")
    fine=1
  else:
    print("Temperaturefield calculation of cooling step finished. \n")
#create stress Abaqus input file
if fine==0:
  if 0 != os.system(parameter.abaquspath+' python -u create_stress_input.py --model '+str(parameter.model)):
          sys.exit('Error during create_stress_input.py.')
    fine=1
  else:
    print("Stress input file successfully created.")
#start stress calculation
if fine==0:
  print('Start of stress calculation')
  if 0 != os.system(str(parameter.abaquspath)+' job=j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint inp=
      j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint.inp cpus='+str(parameter.cpus)+' interactive | tee
      j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint.log'):
    sys.exit("Error during stress calculation")
    fine=1
  else:
    print("Stress calculation finished.")
exit()
```

C++ script which performs phase assignment according to the filling factor

Appendix2/create_poly_distribution_aut.cpp

```
//
// Script to create random phase distribution
// automatic version
// Version 1
#include <fstream>
#include <vector>
#include <iostream>
#include <string>
#include <stdio.h>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#define _USE_MATH_DEFINES
#include <math.h>
using namespace std;
using std::ofstream;
//
//variable definition
//
string phasefilename;   //name of the phase file
//
//method which creates a vector containing the polyhedra and the corresponding phase
void getphases (int amountofpoly, double phfraction){
  std::vector<int>phases;
  ofstream myfile;
  myfile.open(phasefilename.c_str());
  myfile<<"polyhedra: phase:\n";
  int rannumber;
  int intran;
  cout<<"philfactor= "<<phfraction<<endl;
  srand(time(NULL));
  for (int i=0; i<amountofpoly; i++){
    rannumber = int(rand() % 100 + 1);  //create a random number between 1 and 100
    if (rannumber<=(phfraction*100.)) //fill factor
      phases.push_back(1);        //1 corresponds to plagioclase
    else
      phases.push_back(0);        //0 corresponds to quartz
    myfile<<i;
    myfile<<" ";
    myfile<<phases[i];
    myfile<<"\n";
  }
}
//
//main
int main(int argc, char **argv) {
  double phfraction;              //phase fraction of plagioclase
  int amountpoly;               //amount of polyhedra in the model
  std::stringstream str_phfraction;   //string of phase fraction of plagioclase
  std::ostringstream str_model;     //string of model name
  std::stringstream str_amountpoly;   //string of amount of polyhedra in the model
  str_phfraction<<argv[1];
  str_model<<argv[2];
  str_amountpoly<<argv[3];
  str_phfraction>>phfraction;
  str_amountpoly>>amountpoly;
  phasefilename="phases_"+str_model.str()+".txt";
  getphases (amountpoly, phfraction);
  return 0;
}
```

C++ script for the *Meep* FDTD calculation

Appendix2/polyhedron_aut.cpp

```
//
//MEEP FILE
//Version 1.0
//inhomogeneous material
//
#include <meep.hpp>
#include <fstream>
#include <vector>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#define _USE_MATH_DEFINES
#include <math.h>
#include <mpi.h>
using namespace meep;
//
//variable definition
//
double frequencyHz=2.45e9;        //frequency of microwaves
double c_m=299792458;           //speed of light
double sfactor=0.4;             //factor which the neper mesh is scaled
//plagioclase
double pepsreal=7.690;            //real part of eps
double pepsima=2.787;           //imaginary part of eps
//quartz
double qepsreal=7.1;            //real part of eps
//
// averaged material
double avgepsreal=7.4;            //real part of eps
double avgepsima=0.88;            //imaginary part of eps
//
```

```
std::vector<int> phases;          //variable to save the phases to the corresponding polyhedra
std::vector<std::vector<std::vector<double> > >dat; //variable to save the geometric data of the microstructure
std::vector<std::vector<double> >boundCon;     //variable to save the bounding container of the different polyhedron
int lastpoly=-1;             //variable to save the polyhedron of the last point
double timestepair=0.0;
double w0=0.043;              // waist radius of Gaussian beam
double e0=1;                //  standard value of E
int mynode, totalnodes;
MPI_Status status;
string tessfile;              //name of the tess file
string phasefile;             //name of the phase file
//
//definition of the different methods
//
// Method which read data from the Neper file
std::vector<std::vector<std::vector<double> > >getpolyhedron () {
  int omittedlines=6;           // insert the line which should be omitted
  string s;
  std::vector<std::string> v;
  ifstream t;
  int amountvertex;           //amount of vertex
  int amountedge;             //amount of edges
  int amountface;             //amount of faces
  int amountpolyhedron;         //amount of polyhedron
  std::vector<std::vector<double> > vertex; // index of vertex is the column; row is in the form: Amount of edges, edge
     number 1, edge number 2, ..., x- , y-, z- value of vertex
  std::vector<std::vector<double> > edges; // index of edge is the column; row is in the form: vertex 1, vertex 2, number
      of faces, face 1, face 2, ...
  std::vector<std::vector<double> > faces; // index of face is the column; row is in the form: polyhedron 1, polyhedron
     2, number of vertices, ver_1, ver_2,..., edge_1, edge_2, ...
  std::vector<std::vector<double> > polyhedron; // poly_centre_x, poly_centre_y, poly_centre_z, number of faces, face 1,
     face 2, ...
  std::vector<std::vector<std::vector<double> > >ret; //vector which returns all the results
  t.open(tessfile.c_str(), ios::in);
  if (t.good()==false){         //check if tess file exist
    t.close();              //if not close file
    cout<<"tess file is missing!!!"<<endl;  //write error message
    exit(1);              //end program
  }
  while (!t.eof())
  { getline (t,s);
    v.push_back(s);
  }
  t.close();
  for (int j=1; j<=omittedlines; j++){
    v.erase(v.begin());
  }
// save vertex
  int hv=0;
  amountvertex=atoi(v[hv].c_str());
  v.erase(v.begin());
  for (int j=0; j<3*amountvertex; j=j+3){
    std::vector<std::string> temp1;      // amount of edges, edge number 1, edge number 2, ...
    std::vector<std::string> temp2;
    istringstream iss1(v[j+1]);
    copy(istream_iterator<string>(iss1),  // split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp1));
    std::vector<double> row;
    istringstream iss2(v[j+2]);
    copy(istream_iterator<string>(iss2),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp2));
    for (int n=0; n<=2; n++){  // insert coordinates of vertex
      row.push_back(sfactor*atof(temp2[n].c_str()));  //save component and scale it with factor
    }
    vertex.push_back(row);  //insert row in 2 dimensional vector
  }
  cout<<"z koordinate von vertex 1:"<<vertex[0][2]<<endl;
  ret.push_back(vertex);
  int he=hv+amountvertex*3+1;
  amountedge=atoi(v[he].c_str());
  for (int j=he+1; j<he+1+4*amountedge; j=j+4){
    std::vector<std::string> temp3;
    std::vector<std::string> temp4;
    istringstream iss3(v[j+1]);
    copy(istream_iterator<string>(iss3), //split of the single lines
        istream_iterator<string>(),
      back_inserter<vector<string> >(temp3));
    std::vector<double> row2;
    for (int n=0; n<2;n++){ //insert vertex 1, vertex 2
      row2.push_back(atof(temp3[n].c_str()));
    }
    istringstream iss4(v[j+2]);
    copy(istream_iterator<string>(iss4),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp4));
    row2.push_back(atof(temp4[0].c_str())); // insert amount of faces
    for (int n=1; n<=atoi(temp4[0].c_str()); n++){
      row2.push_back(atof(temp4[n].c_str()));
    }
    edges.push_back(row2); //insert row2 in 2 dimensional vector
  }
  ret.push_back(edges);
// save faces
  int hf=he+amountedge*4+2;
  amountface=atoi(v[hf].c_str());
  for (int j=hf+1; j<hf+1+7*amountface; j=j+7){
    std::vector<std::string> temp5;
    std::vector<std::string> temp6;
    std::vector<std::string> temp7;
    istringstream iss5(v[j+2]);
    copy(istream_iterator<string>(iss5), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp5));
    std::vector<double> row3;
    for (int n=0; n<4;n++){ //insert face eq_a, face eq_b, face eq_c, face eq_d
      row3.push_back(atof(temp5[n].c_str()));
    }
    istringstream iss6(v[j+3]);
    copy(istream_iterator<string>(iss6),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp6));
    row3.push_back(atof(temp6[0].c_str())); // insert amount of vertex
    for (int n=1; n<=atoi(temp6[0].c_str()); n++){  //insert ver_1, ver_2, ...
      row3.push_back(atof(temp6[n].c_str()));
    }
    faces.push_back(row3); //insert row3 in 2 dimensional vector
  }
  ret.push_back(faces);
// save polyhedron
  int hp=hf+amountface*7+2;
  amountpolyhedron=atoi(v[hp].c_str());
  for (int j=hp+1; j<hp+1+3*amountpolyhedron; j=j+3){
    std::vector<std::string> temp8;
    std::vector<std::string> temp9;
    istringstream iss8(v[j]);
    copy(istream_iterator<string>(iss8), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp8));
    std::vector<double> row4;
    for (int n=1; n<4;n++){ //insert coordinates of the middle point of the polyhedron
      row4.push_back(sfactor*atof(temp8[n].c_str()));   //save component and scale it with factor
    }
    istringstream iss9(v[j+2]);
    copy(istream_iterator<string>(iss9),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp9));
    row4.push_back(atof(temp9[0].c_str())); // insert amount of faces
    for (int n=1; n<=atoi(temp9[0].c_str()); n++){  //insert face_1, face_2, ...
      row4.push_back(atof(temp9[n].c_str()));
    }
    polyhedron.push_back(row4); //insert row 4 in 2 dimensional vector
  }
  ret.push_back(polyhedron);
 return ret;
}
//Method to get a point on the face
std::vector<double>getver (int face){
  std::vector<double> coordinates;
  if (face<0)
    face=face*(-1);
  int curvertex=dat[2][face-1][5];        //a vertex which lies on the face
  for (int i=0; i<3; i++){            //loop over x, y, z
    coordinates.push_back(dat[0][curvertex-1][i]);  //coordinate of the vertex
  }
  return coordinates;
}
//Method to calculated the inproduct
double inproduct (std::vector<double> normal, double point[3], std::vector<double> ver){
  double inproduct;
  double vecverpoint[3];          //vector from vertex to point
  for (int i=0; i<3; i++){
    vecverpoint[i]=point[i]-ver[i];
  }
  inproduct=normal[0]*vecverpoint[0]+normal[1]*vecverpoint[1]+normal[2]*vecverpoint[2]; //inproduct of vecverpoint and
      normal
  return inproduct;
}
//Method to calculate the Bounding Sphere of the polyhedra
std::vector<std::vector<double> > getBS (){
  std::vector<std::vector<double> >BS;
  int face;
  double radius;
  double radiusold;
  for (unsigned int i=0; i<dat[3].size(); i++){
    radiusold=0.;
    std::vector<double> row;
    std::vector<int>ver;
    for (int j=0; j<3; j++)
      row.push_back(dat[3][i][j]);
    for (int z=0; z<dat[3][i][3]; z++){          //loop over faces
      face=dat[3][i][z+4];                //face ID
      if (face<0)                   //if face ID is negative make it positive
        face=face*(-1);
      for (int v=0; v<dat[2][face-1][4]; v++){     //loop over vertexes of the face
        ver.push_back(dat[2][face-1][v+5]);      //save ID of vertex
      }
    }
    for (unsigned int vv=0; vv<ver.size(); vv++){    //loop over vertexes
      radius=sqrt(pow((dat[0][ver[vv]-1][0]-row[0]),2)+pow((dat[0][ver[vv]-1][1]-row[1]),2)+pow((dat[0][ver[vv]-1][2]-row
      [2]),2));   //calculate radius
      if (radiusold<radius)              //if new radius is bigger than update radius
        radiusold=radius;
    }
    row.push_back(radiusold);
    BS.push_back(row);
  }
  return BS;
}
//Method which assign the intersecting faces to the polyhedron
void assignface (){
  int face;                      //faceID
  int count;                      //count amount of boundary faces
  bool alreadyin;                   //variable which is true if the current faces is allready saved
  std::vector<int> assignedfaces;             //variable where all allready assigned faces are saved
  for (unsigned int i=0; i<dat[3].size(); i++){      //loop over all polyhedron
    count=0;
    dat[3][i].push_back(count);
    for (int j=0; j<dat[3][i][3];j++){           //loop over all faces
      alreadyin=false;
      face=dat[3][i][j+4];
      if (face<0)                   //ensure that faceID is positive
        face=face*(-1);
      for (unsigned int z=0; z<assignedfaces.size(); z++){     //loop over all already saved faces
        if (face==assignedfaces[z]){         //check if face is already saved by an other polyhedra
          alreadyin=true;              //if yes set alreadyin on true
          z=assignedfaces.size()+1;        //and end loop
        }
      }
      if (alreadyin==false){               //check if the face is already saved
        dat[3][i].push_back(face);           // if not save face ID
        assignedfaces.push_back(face);         //saved the faceID in assignedfaces
        count++;                     //increase count
      }
    }
    dat[3][i][dat[3][i][3]+4]=count;             //save amount of boundary faces
  }
}
//Method which return a polyhedron in which the specific point lies
double dielectric (double point[3]){
  bool inside=false;
  int isinpoly;
  int faceIDsearch;
  int amountface;
  double veMP;                         //length of vector from the middle point to the sought point
  std::vector<int>pospoly;                  //vector of possible polyhedra
  double D;                          //inproduct
  if (lastpoly>=0){
    veMP=sqrt(pow(point[0]-boundCon[lastpoly][0],2)+pow(point[1]-boundCon[lastpoly][1],2)+pow(point[2]-boundCon[lastpoly
      ][2],2));
    if (veMP<=boundCon[lastpoly][3]){          //check if point is inside the bounding box
      amountface=dat[3][lastpoly][3];          //amount of faces of the polyhedron
      for (int face=0; face<amountface; face++){   //loop over faces of the polyhedra
        std::vector<double> normal;
        if (dat[3][lastpoly][face+4]>0){
          for (int i=1; i<=3; i++){
            normal.push_back((dat[2][dat[3][lastpoly][face+4]-1][i])*(-1));
          }
        }
        else{
          for (int i=1; i<=3; i++){
                                        normal.push_back(dat[2][(dat[3][lastpoly][face+4]*(-1))-1][i]);
                                }
        }
        std::vector<double> ver=getver(dat[3][lastpoly][face+4]);    //get coordinates of a vertex on the phase
        D=inproduct(normal, point, ver);                //project vector (point-vert) on the normal vector
        if (D<0){
          inside=false;
          face=amountface+1;
        }
        else{
          if (D==0){
            if (dat[3][lastpoly][face+4]<0)                        //check if faceID is positiv
                                          faceIDsearch=dat[3][lastpoly][face+4]*(-1);          //if not make it positiv
            else
                                          faceIDsearch=dat[3][lastpoly][face+4];
            for (int j=0; j<dat[3][lastpoly][amountface+4]; j++){    //loop over all boundary faces
              if (faceIDsearch==dat[3][lastpoly][amountface+4+j]){   //search for faceID
                inside=true;             //if yes set inside true and end loop over boundary faces
                j=dat[3][lastpoly][amountface+4]+1;
              }
              else{
                inside=false;           //if not set inside false
              }
            }
            face=amountface+1;               //end loop over faces of polyhedron
          }
          else
            inside=true;
        }
      }
    }
  }
  if (inside==true){
    isinpoly=lastpoly+1;                     //if yes save polyID
  }
  else{
    for (unsigned int i=0; i<boundCon.size(); i++){
      veMP=sqrt(pow(point[0]-boundCon[i][0],2)+pow(point[1]-boundCon[i][1],2)+pow(point[2]-boundCon[i][2],2));
      if (veMP<=boundCon[i][3])               //check if point is inside the bounding box
        pospoly.push_back(i);                //if yes save the possible polyhedron
    }
    for (unsigned int poly=0; poly<pospoly.size(); poly++){ //loop over the possible polyhedra
      amountface=dat[3][pospoly[poly]][3];          //amount of faces of the polyhedron
      for (int face=0; face<amountface; face++){        //loop over faces of the polyhedron
        std::vector<double> normal;
        if (dat[3][pospoly[poly]][face+4]>0){
          for (int i=1; i<=3; i++){
            normal.push_back((dat[2][dat[3][pospoly[poly]][face+4]-1][i])*(-1));
          }
        }
        else{
          for (int i=1; i<=3; i++){
                                    normal.push_back(dat[2][(dat[3][pospoly[poly]][face+4]*(-1))-1][i]);
                           }
        }
        std::vector<double> ver=getver(dat[3][pospoly[poly]][face+4]);  //get coordinates of a vertex on the phase
        D=inproduct(normal, point, ver);                //project vector (point-vert) on the normal vector
        if (D<0){
          inside=false;
          face=amountface+1;
        }
        else{
          if (D==0){
            if (dat[3][pospoly[poly]][face+4]<0)                     //check if faceID is positive
                                             faceIDsearch=dat[3][pospoly[poly]][face+4]*(-1);        //if not make it
      positive
            else
                                             faceIDsearch=dat[3][pospoly[poly]][face+4];
            for (int j=0; j<dat[3][pospoly[poly]][amountface+4]; j++){     //loop over all boundary faces
              if (faceIDsearch==dat[3][pospoly[poly]][amountface+4+j]){ //search for faceID
                inside=true;          //if yes set inside true and end loop over boundary faces
                j=dat[3][pospoly[poly]][amountface+4]+1;
              }
              else{
                inside=false;         //if not set inside false
              }
            }
            face=amountface+1;            //end loop over faces of polyhedron
          }
          else
            inside=true;
        }
      }
      if (inside==true){            //check if point is in current polyhedron
        isinpoly=pospoly[poly]+1;     //if yes save polyID
        lastpoly=pospoly[poly];       //save the last poly
        poly=pospoly.size()+1;        //and end loop over possible polyhedra
      }
    }
  }
  return isinpoly;
}
//Method which creates a vector containing the polyhedra and the corresponding phase
std::vector<int> getphases (){
  std::vector<int>phases;
  ifstream p;
  string s;
  std::vector<std::string>v;
  p.open(phasefile.c_str(), ios::in);
  if (p.good()==false){         //check if phase file exist
                p.close();          //if not close file
                cout<<"phase file is missing!!!"<<endl;   //print error message
                exit(1);          //exit program
        }

  while (!p.eof()){
    getline(p,s);
    v.push_back(s);
  }
  p.close();
  v.erase(v.begin());
  for (unsigned int i=0; i<dat[3].size();i++){
    std::vector<std::string> temp1;
    istringstream iss1(v[i]);
    copy(istream_iterator<string>(iss1), //  split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp1));
    for (int n=0; n<2; n++){
      if (n==1){
        phases.push_back(atoi(temp1[n].c_str()));
      }
    }
  }
  return phases;
}
//Method to create real part of eps
double eps (const vec &p) {
  int polyhedra;
  double realeps;     //real part of eps
  vec r= p-vec(0.17,0.2,0.17);
  if ((p.x()<0.1)||(p.x()>0.4)||(p.y()<0.2)||(p.y()>0.5)||(p.z()<0.1)||(p.z()>0.4)){
    realeps=1.;
  }
  else {
    if ((p.x()<0.17)||(p.x()>0.33)||(p.y()<0.2)||(p.y()>0.28)||(p.z()<0.17)||(p.z()>0.33)){
      realeps=avgepsreal;
    }
    else{
      double point[3];
      if ((r.x()<=0.08)&&(r.z()<=0.08)){
        point[0]=r.x();
        point[1]=r.y();
        point[2]=r.z();
      }
      else{
        if (r.x()<=0.08){
          point[0]=r.x();
          point[1]=r.y();
          point[2]=r.z()-2*(r.z()-0.08);
        }
        else{
          if (r.z()<=0.08){
            point[0]=r.x()-2*(r.x()-0.08);
            point[1]=r.y();
            point[2]=r.z();
          }
          else{
            point[0]=r.x()-2*(r.x()-0.08);
            point[1]=r.y();
            point[2]=r.z()-2*(r.z()-0.08);
          }
        }
      }
      polyhedra=dielectric(point);
      if (phases[polyhedra-1]==0)       //quartz
        realeps=qepsreal;
      if (phases[polyhedra-1]==1)
        realeps=pepsreal;         //plagioclase
    }
  }
  r.~vec();
  return realeps;
}
//Method to define real part of eps for air
double epsair (const vec &p) {
  double epsair=1.;
  return epsair;
}
//Class and method to define imaginary part of eps
class my_material:public material_function {
  bool has_conducitivity(component c) {
      if(c == Dz)
            return true;
          else
            return false;
  }
  double conductivity (component c, const vec &l){
    int polyhedrac;
    double con;
    vec rc= l-vec(0.17, 0.2, 0.17);
    if ((l.x()<0.1)||(l.x()>0.4)||(l.y()<0.2)||(l.y()>0.5)||(l.z()<0.1)||(l.z()>0.4)){
      con=0;
    }
    else {
      if ((l.x()<0.17)||(l.x()>0.33)||(l.y()<0.2)||(l.y()>0.28)||(l.z()<0.17)||(l.z()>0.33)){
        con=((2*M_PI*(frequencyHz/c_m)*avgepsima)/avgepsreal);      //averaged material
      }
      else{
        double pointc[3];
        if ((rc.x()<=0.08)&&(rc.z()<=0.08)){
          pointc[0]=rc.x();
          pointc[1]=rc.y();
          pointc[2]=rc.z();
        }
        else{
          if (rc.x()<=0.08){
            pointc[0]=rc.x();
            pointc[1]=rc.y();
            pointc[2]=rc.z()-2*(rc.z()-0.08);
          }
          else{
            if (rc.z()<=0.08){
              pointc[0]=rc.x()-2*(rc.x()-0.08);
              pointc[1]=rc.y();
              pointc[2]=rc.z();
            }
            else{
              pointc[0]=rc.x()-2*(rc.x()-0.08);
              pointc[1]=rc.y();
              pointc[2]=rc.z()-2*(rc.z()-0.08);;
            }
          }
        }
        polyhedrac=dielectric(pointc);
        if (phases[polyhedrac-1]==0)                    //quartz
          con=0;
        if (phases[polyhedrac-1]==1)
          con=((2*M_PI*(frequencyHz/c_m)*pepsima)/pepsreal);      //plagioclase
      }
    }
    rc.~vec();
    return con;
  }
};
//Method to define the gauss profile
complex<double> gauss(const vec &p){
  complex<double> amplitude;
  double radius=sqrt(pow(p.x(),2)+pow(p.z(),2));
  amplitude=(e0*exp(-pow((radius/w0),2)));
  p.~vec();
  return amplitude;
}
//Method which define the polycrystal structure of the model
void polycrystal(double a, component c) {
  double pml_thickness=0.1;
  double start, end, totalstart, totalend;    //variable to calculate time
  totalstart=MPI_Wtime();
  my_material ui;
  int timestep=0;
  grid_volume v = vol3d(0.5,0.6,0.5,a);
  start=MPI_Wtime();
  symmetry S=mirror(X,v)-mirror(Z,v);
  structure s1(v, eps, pml(pml_thickness), S, 0, 0.5, false, DEFAULT_SUBPIXEL_TOL, DEFAULT_SUBPIXEL_MAXEVAL);
  end=MPI_Wtime();
  cout<<"Eps defined by processor "<<mynode<<" of "<<totalnodes<<"nodes, after "<<end-start<<"s"<<endl;
  start=MPI_Wtime();
  s1.set_conductivity(Dz,ui);
  end=MPI_Wtime();
  cout<<"Con defined by processor "<<mynode<<" of "<<totalnodes<<"nodes, after  "<<end-start<<"s"<<endl;
  fields f1(&s1);
  f1.use_real_fields();
  f1.output_hdf5(Dielectric, v.surroundings());
  double freq = frequencyHz/c_m;
  double amplitude=-1.0;
  cout<<"variables defined by processor "<<mynode<<" of "<<totalnodes<<endl;
  start=MPI_Wtime();
  continuous_src_time src(freq);
  volume src_plane(vec(0.0,0.19,0.0),vec(0.5,0.19,0.5));
  f1.add_volume_source(Hx,src,src_plane,gauss,amplitude);
  f1.add_volume_source(Ez,src,src_plane,gauss,amplitude);
  end=MPI_Wtime();
  cout<<"source f1 defined by processor "<<mynode<<" of "<<totalnodes<<"nodes after "<<end-start<<"s"<<endl;
  master_printf("volume sources added...\n");
  start=MPI_Wtime();
//Version where the whole period with all time points is calculated
  while (f1.time()<=(25./freq)){
    f1.step();
    if (f1.time()>=(24./freq)){      //calculate time average E field for every node of ever element
      timestep++;
      cout<<"total timestep "<<timestep<<endl;
      f1.output_hdf5(Ex,v.surroundings());
      f1.output_hdf5(Ey,v.surroundings());
      f1.output_hdf5(Ez,v.surroundings());
    }
  }
  end=MPI_Wtime();
  cout<<"total amount of timesteps "<<timestep<<" after "<<end-start<<"s"<<endl;
  totalend=MPI_Wtime();
  cout<<"the total calculation finished after "<<totalend-totalstart<<"s "<<endl;
}
//Main program
int main(int argc, char **argv) {
  initialize mpi(argc, argv);
  std::ostringstream str_model;   //string of model name
  std::ostringstream str_tess;    //string of tessfile
  str_tess<<argv[1];
  str_model<<argv[2];
  phasefile="phases_"+str_model.str()+".txt";
  tessfile=str_tess.str();
  MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
  MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
  dat=getpolyhedron();         //vector which returns all the results
    phases=getphases();          //get the phase to the corresponding polyhedra; 0=quartz, 1=plagioclase
  assignface();            //get the polyhedron which correspond to the bounding faces
  boundCon=getBS();           //calculate boundary spheres
  double resolution=1000;        //1000  pixels per distance
  polycrystal(resolution, Ez);
  master_printf("finished.\n");
  return 0;
}
```

C++ script to calculate $\overline{E^2}/\overline{E^2_{0air}}$

Appendix2/hdf5_aut.cpp

```
//
//C++ script to calculate the time averaged squared electric field
//Automatic version
//Version 1.0
//
#include <fstream>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#define _USE_MATH_DEFINES
#include <math.h>
#include <cmath>
#include <vector>
#include "H5Cpp.h"
#ifndef H5_NO_NAMESPACE
#ifndef H5_NO_STD
using std::cout;
using std::endl;
using std::string;
using std::ifstream;
using std::ofstream;
using std::istringstream;
using std::ios;
using std::istream_iterator;
using std::vector;
using std::back_inserter;
#endif // H5_NO_STD
#endif
#ifndef H5_NO_NAMESPACE
using namespace H5;
#endif
//
//variable definition
//
const double lx=0.5;          //dimension in x-direction of full calculation space
const double ly=0.6;          //dimension in y-direction of full calculation space
const double lz=0.5;          //dimension in z-direction of full calculation space
const double starttime=5874.;    //starttime of E field calculation (start of one period)
const double endtime=6119.;      //endtime of E-field calculation (End of one period)
const double e0=59650.8;       //E2 at the source position with just air
const double e0x=8.8151e-25;     //E2x at the source position with just air
const double e0y=2.98858e-24;    //E2y at the source position with just air
const double e0z=59650.8;      //E2z at the source position with just air
const int pixel=1000;         //Pixel/meter
const int amountnodes=703553;    //amount of nodes
const int amountelements=4060685; //amount of elements
const int amountpoly=30000;      //amount of polyhedra
const double x=0.1;           //x position of coordinate system of bulk material
const double y=0.2;           //y position of coordinate system of bulk material
const double z=0.1;           //z position of coordinate system of bulk material
const double frequency=2.45e9;     //frequency of electromagnetic waves
//
//GLOBAL VARIABLES
std::vector<std::vector<std::vector<double> > > e;     //field in which the E2 data is saved
std::vector<std::vector<std::vector<double> > > ex;    //field in which the E2x data is saved
std::vector<std::vector<std::vector<double> > > ey;    //field in which the E2y data is saved
std::vector<std::vector<std::vector<double> > > ez;    //field in which the E2z data is saved
std::vector<std::vector<std::vector<double> > >datoutput; //variable to save the node data of the output elements
std::vector<std::vector<double> > integrationpointsglobal;  //variable to save coordinates of the integration points
std::vector<int> phases;       //variable to save the phases to the corresponding polyhedra
int nx;                //amount of points in x-direction; NOTE: nx=ny=nz!!!!
int ny;                //amount of points in y-direction
int nz;                //amount of points in z-direction
double h;              //grid constant lx/nx
double ttimestep;          //total amount of time steps within one period
//
//MAIN PROGRAMM
//
int main (int argc, char **argv){
  //VARIABLE DEFINITION
  std::ostringstream argvstring;
  argvstring<<argv[1];
  H5std_string out_e("output3D_"+argvstring.str()+"_E2.h5");  //output file for E2 hdf5 output
  H5std_string out_ex("output3D_"+argvstring.str()+"_Ex.h5"); //output file for E2x hdf5 output
  H5std_string out_ey("output3D_"+argvstring.str()+"_Ey.h5"); //output file for E2y hdf5 output
  H5std_string out_ez("output3D_"+argvstring.str()+"_Ez.h5"); //output file for E2z hdf5 output
  ttimestep=(endtime-starttime)+1;
  string component;
  string filename;
  nx=lx*double(pixel);       //amount of grid points in x direction
  ny=ly*double(pixel);       //amount of grid points in y direction
  nz=lz*double(pixel);       //amount of grid points in z direction
  h=1./double(pixel);        //grid constant
  int rank;
  hsize_t dims[3];
  std::ostringstream strs;
  //BODY
  //open first ez file in order to define global array
  component="ez";
  strs<<starttime;
  if (starttime<10000){
    filename=component+"-00000"+strs.str()+".h5";
  }
  else{
    filename=component+"-0000"+strs.str()+".h5";
  }
  cout<<"filename: "<<filename<<endl;
  H5std_string FILE_NAME(filename);
```

```
strs.str("");
strs.clear();
      H5std_string DATASET_NAME(component);
H5File file( FILE_NAME, H5F_ACC_RDONLY );        //open file
DataSet dataset = file.openDataSet( DATASET_NAME ); //open dataset
DataSpace filespace = dataset.getSpace();        //filespace for rank and dimension
rank = filespace.getSimpleExtentNdims();         //get number of dimensions in the file dataspace
rank = filespace.getSimpleExtentDims( dims );    //number of dimensions in the file dataspace
      DataSpace mspace1(rank, dims);
DSetCreatPropList cparms = dataset.getCreatePlist();//get properties list
  H5File filewrite_e2(out_e, H5F_ACC_TRUNC );       //create file
H5File filewrite_ex2(out_ex, H5F_ACC_TRUNC );     //create file in which the e2 data is stored
  H5File filewrite_ey2(out_ey, H5F_ACC_TRUNC );     //create file in which the e2 data is stored
  H5File filewrite_ez2(out_ez, H5F_ACC_TRUNC );     //create file in which the e2 data is stored
DataSpace filespacewrite=filespace;            //space to write data
H5std_string DATASET_NAME_WRITE("E2");          //define dataset name
H5std_string DATASET_NAME_Ex2("Ex2");         //define dataset name for E2x output
H5std_string DATASET_NAME_Ey2("Ey2");         //define dataset name for E2y output
H5std_string DATASET_NAME_Ez2("Ez2");         //define dataset name for E2z output
DataSet datasetwrite_e2=filewrite_e2.createDataSet(DATASET_NAME_WRITE, PredType::NATIVE_DOUBLE, mspace1,cparms);  //
    create dataset
DataSet datasetwrite_ex2=filewrite_ex2.createDataSet(DATASET_NAME_Ex2, PredType::NATIVE_DOUBLE, mspace1,cparms);  //
    create dataset for E2x output
DataSet datasetwrite_ey2=filewrite_ey2.createDataSet(DATASET_NAME_Ey2, PredType::NATIVE_DOUBLE, mspace1,cparms);  //
    create dataset for E2y output
DataSet datasetwrite_ez2=filewrite_ez2.createDataSet(DATASET_NAME_Ez2, PredType::NATIVE_DOUBLE, mspace1,cparms);  //
    create dataset for E2z output
filespace.close();
dataset.close();
      file.close();
cout << "dataset rank = " << rank << ", dimensions "<< dims[0] << " x "<< dims[1] <<" x "<<dims[2]<<endl;
//allocate e 3D array
e.resize(nx);
ex.resize(nx);
ey.resize(nx);
ez.resize(nx);
for (int i=0;i<nx; i++){
  e[i].resize(ny);
  ex[i].resize(ny);
  ey[i].resize(ny);
  ez[i].resize(ny);
  for (int j=0; j<ny; j++){
    e[i][j].resize(nz);
    ex[i][j].resize(nz);
    ey[i][j].resize(nz);
    ez[i][j].resize(nz);
  }
}
//create coldata vector in which E field data is saved
std::vector<std::vector<std::vector<std::vector<double> > > >coldata; //field in which all Efield data is saved:
    coldata[0]=Ex, coldata[1]=Ey, coldata[2]=Ez
coldata.resize(6);
for (int k=0; k<6; k++){
  coldata[k].resize(nx);
        for (int i=0;i<nx; i++){
                coldata[k][i].resize(ny);
                for (int j=0; j<ny; j++){
                      coldata[k][i][j].resize(nz);
      for (int h=0; h<nz; h++){
        coldata[k][i][j][h]=0.;
      }
                }
        }
}
// sum up E2 data
for (int i=0; i<3; i++){            //loop over all E field components
  if (i==0)
    component="ex";
  if (i==1)
    component="ey";
  if (i==2)
    component="ez";
  for (double time=starttime; time<=endtime; time++){ //loop over all times
    strs<<time;
    if (time<10000){
        filename=component+"-00000"+strs.str()+".h5";
      }
      else{
        filename=component+"-0000"+strs.str()+".h5";
      }
      strs.str("");
              strs.clear();
      cout<<"filename "<<filename<<endl;
      H5std_string FILE_NAME(filename);
              H5std_string DATASET_NAME(component);
              H5File file( FILE_NAME, H5F_ACC_RDONLY );                 //open file
              DataSet dataset = file.openDataSet( DATASET_NAME );      //open dataset
        DataSpace filespace = dataset.getSpace();                   //filespace for rank and dimension
      double *data_out=new double[nx*ny*nz];              //one dim field in which the outputdata is buffered
      dataset.read( data_out, PredType::NATIVE_DOUBLE, mspace1, filespace );   //read data from hdf5 file
      //loop over all points
      for (int x=0; x<nx; x++){
        for (int y=0; y<ny; y++){
          for (int z=0; z<nz; z++){
            if (time==starttime){
              coldata[i][x][y][z]=pow(((data_out[z+y*nz+nz*ny*x]/2)*376.73),2);        //half of boundary point, 376.73 is
 sqrt(mu_0/eps_0) which accounts for the Meep units
            }
            else{
              if (time==endtime)
                coldata[i][x][y][z]=coldata[i][x][y][z]+pow(((data_out[z+y*nz+nz*ny*x]/2)*376.73),2); //half of boundary
    point, 376.73 is sqrt(mu_0/eps_0)
              else
                coldata[i][x][y][z]=coldata[i][x][y][z]+pow((data_out[z+y*nz+nz*ny*x]*376.73),2); //sum up of E2 values (
    components), 376.73 is sqrt(mu_0/eps_0)
            }
          }
        }
      }
      filespace.close();
              dataset.close();
              file.close();
      delete[] data_out;
    }
  }
  //loop over all points
  for (int x=0; x<nx; x++){
          for (int y=0; y<ny; y++){
                  for (int z=0; z<nz; z++){
                          e[x][y][z]=sqrt(pow((coldata[0][x][y][z]/(ttimestep-1)),2)+pow((coldata[1][x][y][z]/(ttimestep
    -1)),2)+pow((coldata[2][x][y][z]/(ttimestep-1)),2))/e0; //calculate E2 values and scale to the source value
        ex[x][y][z]=(coldata[0][x][y][z]/(ttimestep-1))/e0x;     //calculate Ex2
        ey[x][y][z]=(coldata[1][x][y][z]/(ttimestep-1))/e0y;     //calculate Ey2
        ez[x][y][z]=(coldata[2][x][y][z]/(ttimestep-1))/e0z;     //calculate Ez2
      }
                  }
        }
  double *data_write_e2=new double[nx*ny*nz];
  double *data_write_ex2=new double[nx*ny*nz];
  double *data_write_ey2=new double[nx*ny*nz];
  double *data_write_ez2=new double[nx*ny*nz];
  for (int x=0; x<nx; x++){
                for (int y=0; y<ny; y++){
                        for (int z=0; z<nz; z++){
        data_write_e2[z+y*nz+nz*ny*x]=e[x][y][z];
        data_write_ex2[z+y*nz+nz*ny*x]=ex[x][y][z];
        data_write_ey2[z+y*nz+nz*ny*x]=ey[x][y][z];
        data_write_ez2[z+y*nz+nz*ny*x]=ez[x][y][z];
      }
    }
  }
  datasetwrite_e2.write(data_write_e2, PredType::NATIVE_DOUBLE, mspace1, filespacewrite);
  datasetwrite_ex2.write(data_write_ex2, PredType::NATIVE_DOUBLE, mspace1, filespacewrite);
  datasetwrite_ey2.write(data_write_ey2, PredType::NATIVE_DOUBLE, mspace1, filespacewrite);
  datasetwrite_ez2.write(data_write_ez2, PredType::NATIVE_DOUBLE, mspace1, filespacewrite);
  delete[]data_write_e2;
  delete[]data_write_ex2;
  delete[]data_write_ey2;
  delete[]data_write_ez2;
  filespacewrite.close();
  mspace1.close();
  datasetwrite_e2.close();
  datasetwrite_ex2.close();
  datasetwrite_ey2.close();
  datasetwrite_ez2.close();
  filewrite_e2.close();
  filewrite_ex2.close();
  filewrite_ey2.close();
  filewrite_ez2.close();
  return 0;
}
```

C++ script to calculate P_{abs} at the integration points

Appendix2/hdf5_int_v4_aut.cpp

```
//
//C++ script to calculate absorbed power density at the integration points
//automatic version
//Version 1.0
//
#include <fstream>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#define _USE_MATH_DEFINES
#include <math.h>
#include <cmath>
#include <vector>
#include "H5Cpp.h"
#ifndef H5_NO_NAMESPACE
#ifndef H5_NO_STD
using std::cout;
using std::endl;
using std::string;
using std::ifstream;
using std::ofstream;
using std::istringstream;
using std::ios;
using std::istream_iterator;
using std::vector;
using std::back_inserter;
#endif // H5_NO_STD
#endif
#ifndef H5_NO_NAMESPACE
using namespace H5;
#endif
//VARIABLES which have to be defined by the user
const double lx=0.5;            //dimension in x-direction of full calculation space
const double ly=0.6;            //dimension in y-direction of full calculation space
const double lz=0.5;            //dimension in z-direction of full calculation space
const double pepsima=2.787;         //imaginary part of eps
const double avg_granite_epsima=0.88;    //imaginary part of eps of avg_granite
const int pixel=1000;           //Pixel/meter
const double x=0.1;             //x position of coordinate system of bulk material
const double y=0.2;             //y position of coordinate system of bulk material
const double z=0.1;             //z position of coordinate system of bulk material
const double xp=0.07;           //x difference between coordinate system of bulk material and poly part
const double yp=0.0;            //y difference between coordinate system of bulk material and poly part
const double zp=0.07;           //z difference between coordinate system of bulk material and poly part
const double frequency=2.45e9;       //frequency of electromagnetic waves
const int polyelements=281988;       //Amount of outer elements
//
//GLOBAL VARIABLES
std::vector<std::vector<std::vector<double> > > e;     //field in which the E2 data is saved
```

```
std::vector<std::vector<double> > integrationpoints;  //2dim field in which the coordinates of the integration points are
       saved
std::vector<int> phases;            //variable to save the phases to the corresponding polyhedra
int nx;                    //amount of points in x-direction
int ny;                    //amount of points in y-direction
int nz;                    //amount of points in z-direction
double h;                  //grid constant lx/nx
std::vector<std::vector<double> >elementsets;      //variable to save the element sets
std::ostringstream model;          //name of the model
std::ostringstream inpintegrationpoints;  //name of the input file containing the coordinates of the integration points
std::ostringstream inpelementset;      //name of the file containing the phase information
string outputfile;              //string to the outputfile
string phasefile;              //string of the phasefile
//
//INIT MEHTHODS
//
std::vector<std::vector<double> > getintpoint ();
std::vector<std::vector<double> > getelementsets ();
double interpolation (std::vector<double> P);
std::vector<int> getphases (int amountpoly);
//
//MAIN PROGRAMM
//
int main (int argc, char **argv){
  //VARIABLE DEFINITION
  model<<argv[1];
  inpintegrationpoints<<argv[2];
  inpelementset<<argv[3];
  outputfile="absorption_101_"+model.str()+"_03_02_1000_intpoints.txt";
  phasefile="phases_"+model.str()+".txt";
  nx=lx*double(pixel);           //amount of grid points in x direction
  ny=ly*double(pixel);           //amount of grid points in y direction
  nz=lz*double(pixel);           //amount of grid points in z direction
  h=1./double(pixel);            //grid constant
  int rank;
  hsize_t dims[3];
  std::ostringstream strs;
  double absorption;
  ofstream abs;
  int curphase;
  double con=0;
  //BODY
  //open first ez file in order to define global array
        H5std_string DATASET_NAME("E2");
  H5File e2_data("output3D_"+model.str()+"_E2.h5", H5F_ACC_RDONLY );     //open file
  DataSet dataset = e2_data.openDataSet( DATASET_NAME );  //open dataset
  DataSpace filespace = dataset.getSpace();   //filespace for rank and dimension
  rank = filespace.getSimpleExtentNdims();    //get number of dimensions in the file dataspace
  rank = filespace.getSimpleExtentDims( dims ); //number of dimensions in the file dataspace
        DataSpace mspace1(rank, dims);
  DSetCreatPropList cparms = dataset.getCreatePlist();  //get properties list
  cout << "dataset rank = " << rank << ", dimensions "<< dims[0] << " x "<< dims[1] <<" x "<<dims[2]<<endl;
  //allocate e 3D array
  e.resize(nx);
  for (int i=0;i<nx; i++){
    e[i].resize(ny);
    for (int j=0; j<ny; j++){
      e[i][j].resize(nz);
    }
  }
  // save E2 homogeneous material data
  double *data_out=new double[nx*ny*nz];    //one dim field in which the E2 data is buffered
  dataset.read(data_out, PredType::NATIVE_DOUBLE, mspace1, filespace);    //read data from E2 hdf5 file
  //loop over all points
  for (int x=0; x<nx; x++){
    for (int y=0; y<ny; y++){
      for (int z=0; z<nz; z++){
        e[x][y][z]=data_out[z+y*nz+nz*ny*x];  //save e2 value
      }
    }
  }
  filespace.close();
  dataset.close();
  e2_data.close();
  delete[] data_out;
  cout<<"dimensions e: "<<e.size()<<" x "<<e[0].size()<<" x "<<e[0][0].size()<<endl;
  cout<<"End h5 saving"<<endl;
  integrationpoints=getintpoint();        //read in integration point coordinates: [x][y]: y=0:element ID, y=1: x-coord, y
      =2: z-coord., y=3: z- coordinate
  elementsets=getelementsets();          //read in element set data
  cout<<"amount of polysets= "<<elementsets.size()<<endl;
  abs.open(outputfile.c_str());
  phases=getphases(elementsets.size());    // get the phase to the corresponding polyhedra; 0=quartz, 1=plagioclase
  int oldelement=0;
  int oldpoly=0;
  int curelement=0;
  cout<<"begin absorption calculation"<<endl;
  for (unsigned int iintp=0; iintp<integrationpoints.size(); iintp++){        //loop over all integration points
    bool endloop=false;
    if (integrationpoints[iintp][0]<=polyelements){       //check if element is in output element set
      absorption=interpolation(integrationpoints[iintp])*2*M_PI*frequency*avg_granite_epsima*8.85418781762e-12;
      abs<<absorption<<endl;
    }
    else{
      curelement=integrationpoints[iintp][0]-polyelements;          //calculate current poly element ID
      if (oldelement==curelement){             //check if Element ID equals Element ID of former integration point
        if (curphase==0){
          absorption=0;
        }
        if (curphase==1){
          con=pepsima*8.85418781762e-12;
          absorption=interpolation(integrationpoints[iintp])*2*M_PI*frequency*con;
        }
        abs<<absorption<<endl;
      }
      else{                       //search for element set
        for (unsigned int ipoly=oldpoly; ipoly<elementsets.size(); ipoly++){      //loop over all elementsets
          for (unsigned int element=0; element<elementsets[ipoly].size(); element++){ //loop over all elements
            if (elementsets[ipoly][element]==curelement){
              curphase=phases[ipoly];
              element=elementsets[ipoly].size()+1;
              endloop=true;
              if (curphase==0){           //quartz
                con=0;
                absorption=con;
              }
              if (curphase==1){
                con=pepsima*8.85418781762e-12;           //plagioclase
                absorption=interpolation(integrationpoints[iintp])*2*M_PI*frequency*con;
              }
              abs<<absorption<<endl;
              oldelement=integrationpoints[iintp][0]-polyelements;
              oldpoly=ipoly;
            }
          }
          if (endloop)
            ipoly=elementsets.size()+1;
        }
        if (endloop==false){
          cout<<"ERROR: element "<<curelement<<" not found in element sets"<<endl;
        }
      }
    }
  }
  abs.close();
  return 0;
}
//Method to read in and save the element ID as well as the coordinates of the integration points
std::vector<std::vector<double> > getintpoint (){
  int deletbegin=28;            //lines which should be omitted at the begin of the file
  int deletend=10;            //lines which should be omitted at the end of the file
  ifstream f;
  string s;
  std::vector<std::string> v;
  std::vector<std::vector<double> > coordinates;  //vector in which the coordinates of the integration points are saved0
  f.open(inpintegrationpoints.str().c_str(), ios::in);
  if (f.good()==false){        //check if input file exist
              f.close();          //if not close file
              cout<<"inp file is missing!!!"<<endl; //print error message and
              exit(1);          //exit program
        }
  while (!f.eof())
  { getline (f,s);
    v.push_back(s);
  }
  f.close();
  for (int j=1; j<=deletbegin; j++){
    v.erase(v.begin());
  }
  for (int k=1; k<=deletend; k++){
    v.erase(v.end());
  }
  cout<<"groesse v="<<v.size()<<endl;
  cout<<"letzte zeile"<<v[v.size()-1]<<endl;
  for (unsigned int i=0; i<v.size(); i++){   //save all integration points
    std::vector<std::string> temp1;
    istringstream iss1(v[i]);
    copy(istream_iterator<string>(iss1),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp1));
    std::vector<double> row;
    for (int j=0; j<4; j++){   //save the element ID as well as the integration point node coordinates
      row.push_back(atof(temp1[j].c_str()));
    }
    coordinates.push_back(row);
  }
  cout.precision(8);
  return coordinates;
}
//Method to find FDTD grid point
double interpolation (std::vector<double> P){
  //VARIABLE declaration
  double valueP;
  double deltax;
  double deltay;
  double deltaz;
  int nx;          //position of the FDTD grid point in x direction
  int ny;
  int nz;
  //
  //MAIN Body
  nx=((P[1]+x+xp)/h);      //calculate position in grid, (p[x]+x0-h/2) since different coordinate system is used
  ny=((P[2]+y+yp)/h);
  nz=((P[3]+z+zp)/h);
  return e[nx][ny][nz];
}
//Method which creates a vector containing the polyhedra and the corresponding phase
std::vector<int> getphases (int  amountpoly){
  std::vector<int>phases;
  ifstream f;
  string s;
  std::vector<std::string>v;
  f.open(phasefile.c_str(),ios::in);
  if (f.good()==false){          //check if phase file exist
              f.close();           //if not close file
              cout<<"phase file is missing!!!"<<endl;   //print error message
              exit(1);           //exit program
        }

  while (!f.eof()){
    getline(f,s);
    v.push_back(s);
  }
  f.close();
  v.erase(v.begin());
  for (int i=0; i<amountpoly;i++){
    std::vector<std::string> temp1;
    istringstream iss1(v[i]);
    copy(istream_iterator<string>(iss1), // split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp1));
    for (int n=0; n<2; n++){
      if (n==1){
        phases.push_back(atoi(temp1[n].c_str()));
      }
    }
  }
  return phases;
}
//Method to read in and save the elements and the corresponding nodes
std::vector<std::vector<double> > getelementsets (){
  std::vector<std::vector<double> > elementdata;
  ifstream f;
  string s;
  std::vector<std::string> v;
  f.open(inpelementset.str().c_str(), ios::in);
  if (f.good()==false){        //check if input file exist
                  f.close();          //if not close file
                  cout<<"inp file is missing!!!"<<endl; //print error message and
                  exit(1);          //exit program
        }
  while (!f.eof())
  { getline (f,s);
    v.push_back(s);
  }
  f.close();
  bool newpoly=true;
  unsigned int loc=6;
  bool em;
  bool firstem=false;
  int skip=0;
  std::vector<double> row3;
  // save the polyhedra and the corresponding element number
  for (unsigned int k=0; k<v.size(); k++){       //loop over whole file
    std::vector<std::string> temp3;
    istringstream iss3(v[k]);
    copy(istream_iterator<string>(iss3),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp3));
    if (firstem==true){
      skip++;
      em=temp3.empty();               //em=true if the line is empty
      if ((em==false)&&(newpoly==true)){       //check if line is not empty and a new poly is detetected
        loc=temp3[1].find("poly");
      }
      if (newpoly==true){               //check if a new poly is requiered
        if (loc != 6)               //if "poly" is not on the 7th digit end loop
          k=v.size()+2;
        else
          newpoly=false;
      }
      else{
        if (em==true){                 //if the line is empty erase the certain line
          elementdata.push_back(row3);
          row3.erase(row3.begin(), row3.end());
          newpoly=true;
        }
        else{
          for (unsigned int j=0; j<temp3.size(); j++){
          string str3=temp3[j];          // save each element
            if ((str3.length()-1)==',')      // if element contain , delete ,
              str3.erase(str3.length()-1);
            if (str3.empty()==false)
                row3.push_back(atof(str3.c_str()));
          }
        }
      }
    }
    if (skip==0){
      firstem=temp3.empty();
    }
  }
  return elementdata;
}
```

Python script to assign material properties to the finite elements

Appendix2/phase_assignment_aut.py

```
########################################################################
# Script to assign finite elements to the corresponding materials
########################################################################
#load packages
import math
import argparse                    # to parse arguments
#
########################################################################
#initialization
a=[]
b=[]
y=0
z=0
floats=[]
########################################################################
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()           # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic phase assignment.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',    type=str, required=True, help='Name of the model.')
args = parser.parse_args(namespace=parameter)
########################################################################
#main
dat=open('phases_'+str(parameter.model)+'.txt','r')
for line in dat:
  a.append(line.rstrip())
dat.close()
del a[0]
while y<len(a):
  if a[y][0]==" ":
    a[y]=a[y].replace(' ', '',1)
  floats=[int(x) for x in a[y].split(" ")]
  b.append(floats)
  y=y+1
f=open('quartz_elements_'+str(parameter.model)+'.inp','w')
f.write("*ELSET, Elset=quartz\n")
k=open('plagioclase_elements_'+str(parameter.model)+'.inp','w')
k.write("*ELSET, Elset=plagioclase\n")
#check which phase belongs to each polyhedron
while z<len(b):
  if (b[z][1]==0):               #check if phase==0
    f.write("poly"+str(b[z][0]+1)+"\n")   #write element set in quartz_elements.inp
  if (b[z][1]==1):               #check if phase==1
    k.write("poly"+str(b[z][0]+1)+"\n")   #write element set in plagioclase_elements.inp
  z=z+1
f.close()
k.close()
```

Python script to create the *FORTRAN* subroutine

Appendix2/create_BF_subroutine.py

```
##########################################################################
# Script which automatically creates the DFLUX and UEXTERNALDB subroutine
##########################################################################
#load packages
import sys                              # better than:     import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()                # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created Dflux file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('—model',     type=str, required=True, help='Name of the model.')
parser.add_argument('—initialBF',  type=float, required=True, help='Initial constant body heat multiplier.')
parser.add_argument('—nameBF',        type=str, required=True, help='Name of the DFLUX subroutine.')
parser.add_argument('—work',        type=str, required=True, help='Path of the working directory.')
args = parser.parse_args(namespace=parameter)
##########################################################################
#main
#open and write in subroutine file
df=open(str(parameter.nameBF), 'w')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('CCCCC\n')
df.write('CCCCC        Modul to use data in various subroutines\n')
df.write('CCCCC\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n\n')
df.write('      MODULE Information\n\n')
df.write('      DOUBLE PRECISION, DIMENSION(:), allocatable ::\n')
df.write('     *  absorption\n\n')
df.write('C    information:\n')
df.write('C ipoint      Position in absorbed power denisty file (starts with 0)\n')
df.write('C absorption   Absorbed power density at integration point\n\n')
df.write('      save\n')
df.write('      END MODULE Information\n')
df.write('C\n')
df.write('C\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C Subroutine UEXTERNALDB to read in external files\n')
df.write('C\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('      SUBROUTINE UEXTERNALDB(LOP,LRESTART,TIME,DTIME,KSTEP,KINC)\n')
df.write('C\n')
df.write('      USE Information\n')
df.write('C\n')
df.write('      INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('      DIMENSION TIME(2)\n')
df.write('C\n')
df.write('      integer :: inpoint, stat\n')
df.write('C\n')
df.write('      inpoint=18498644\n')
df.write('      stat=1\n')
df.write('C\n')
df.write('      IF (LOP==0) THEN\n')
df.write('        ALLOCATE(absorption(inpoint))\n')
df.write('        WRITE(*,*) \'======= EINLESEN  =========\'\n')
path=str(parameter.work)+'/absorption_101_'+str(parameter.model)+'_03_02_1000_intpoints'
firstline=path[:42]
secondline=path[42:]+'.txt\', iostat=stat, status=\'old\')'
df.write('        OPEN (unit=140, file=\''+firstline+'\n')
if len(secondline)>66:
  secondlinenew=secondline[:66]
  thirdline=secondline[66:]
  df.write('     *'+secondlinenew+'\n')
  df.write('     *'+thirdline+'\n')
```

```
else:
  df.write('      *'+secondline+'\n')
df.write('        if (stat==0) then\n')
df.write('          write(*,*) \'Das Oeffnen der Datei hat geklappt\'\n')
df.write('        END IF\n')
df.write('        READ(140,*)(absorption(I), I=1,inpoint)\n')
df.write('        CLOSE(unit=140)\n')
df.write('      END IF\n')
df.write('      RETURN\n')
df.write('      END\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C Subroutine DFLUX to calculate body heat flux\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('      SUBROUTINE DFLUX(FLUX,SOL,KSTEP,KINC,TIME,NOEL,NPT,COORDS,\n')
df.write('     1 JLTYP,TEMP,PRESS,SNAME)\n')
df.write('C\n')
df.write('      USE Information\n')
df.write('C\n')
df.write('      INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('      DIMENSION FLUX(2), TIME(2), COORDS(3)\n')
df.write('      CHARACTER*80 SNAME\n')
df.write('      integer :: I\n')
df.write('      double precision BF\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C Definition of variable BF (body flux scale factor)\n')
df.write('      BF='+str(parameter.initialBF)+'\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C SOL estimated results variable\n')
df.write('C FLUX body heat flux which has to be defined\n')
df.write('C TIME step \n')
df.write('C COORDS Coordinate of integration point\n')
df.write('C\n')
df.write('      IF (NOEL.lt.281989) THEN\n')
df.write('        FLUX(1)=BF*absorption((NOEL-1)*8+NPT)\n')
df.write('      ELSE\n')
df.write('        FLUX(1)=BF*absorption(281988*8+(NOEL-281988-1)*4+NPT)\n')
df.write('      END IF\n')
df.write('      RETURN\n')
df.write('      END\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C         Subroutine UVARM to set user defined variables\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('      SUBROUTINE UVARM(UVAR,DIRECT,T,TIME,DTIME,CMNAME,ORNAME,\n')
df.write('     1 NUVARM,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,NDI,NSHR,COORD,\n')
df.write('     2 JMAC,JMATYP,MATLAYO,LACCFLA)\n')
df.write('C\n')
df.write('      USE Information\n')
df.write('C\n')
df.write('      INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('      CHARACTER*80 CMNAME,ORNAME\n')
df.write('      CHARACTER*3 FLGRAY(15)\n')
df.write('      DIMENSION UVAR(NUVARM),DIRECT(3,3),T(3,3),TIME(2)\n')
df.write('      DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)\n')
df.write('      double precision BF\n')
df.write('C     The dimensions of the variables FLGRAY, ARRAY and JARRAY\n')
df.write('C     must be set equal to or greater than 15.\n')
df.write('      BF='+str(parameter.initialBF)+'\n')
df.write('      IF (TIME(1).gt.1.e-2) THEN\n')
df.write('        IF (NOEL.lt.281989) THEN\n')
df.write('          UVAR(1)=absorption((NOEL-1)*8+NPT)\n')
df.write('        ELSE\n')
df.write('          UVAR(1)=absorption(281988*8+(NOEL-281988-1)*4+NPT)\n')
df.write('        END IF\n')
df.write('      END IF\n')
df.write('      RETURN\n')
df.write('      END\n')
df.close()
```

Python script to create the *Abaqus* heat input file

Appendix2/create_heat_input.py

```
########################################################################
# Script to automatically create the Abaqus heat input file
########################################################################
#load packages
import sys                              # better than:     import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()                # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created heat file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',     type=str, required=True, help='Name of the model.')
args = parser.parse_args(namespace=parameter)
########################################################################
#main
#open and write in Abaqus input file
inp=open('j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat.inp', 'w')
inp.write('**Automatic generated heat input file for j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat.inp\n
      ')
inp.write('*Part, name=poly\n')
inp.write('*INCLUDE, Input=poly_74_mesh.inp\n')
inp.write('*INCLUDE, Input=quartz_elements_'+str(parameter.model)+'.inp\n')
inp.write('*INCLUDE, Input=plagioclase_elements_'+str(parameter.model)+'.inp\n')
inp.write('*Solid Section, elset=quartz, material=quartz\n')
inp.write('*Solid Section, elset=plagioclase, material=plagioclase\n')
inp.write('*End Part\n')
inp.write('*Part, name=outer\n')
inp.write('*INCLUDE, Input=outer_74_mesh_linear.inp\n')
inp.write('*Elset, elset=outer_element, generate\n')
inp.write('1, 281988, 1\n')
inp.write('*Solid Section, elset=outer_element, material=avg_granite\n')
inp.write('*End Part\n')
inp.write('*Assembly, name=Assembly\n')
inp.write('*Instance, name=poly-1, part=poly\n')
inp.write('          0.0,             0.,            0.0\n')
inp.write('*End Instance\n')
inp.write('*Instance, name=outer-1, part=outer\n')
inp.write(' 0.,           0.3,          -0.07\n')
inp.write('          0.,           0.3,          -0.07, 0.577350269189626, -0.277350269189626, 0.507350269189626,
      120.\n')
inp.write('*End Instance\n')
inp.write('*Nset, nset=FIX, instance=poly-1\n')
inp.write(' 1\n')
inp.write('*Nset, nset=ALL_poly, generate, instance=poly-1\n')
inp.write(' 1, 703553, 1\n')
inp.write('*Elset, elset=elementALL_poly, generate, instance=poly-1\n')
inp.write(' 1, 4060685, 1\n')
inp.write('*Nset, nset=ALL_outer, generate, instance=outer-1\n')
inp.write(' 1, 296620, 1\n')
inp.write('*Elset, elset=elementALL_outer, generate, instance=outer-1\n')
inp.write(' 1, 281988, 1\n')
inp.write('*INCLUDE, Input=j101_01_69_poly_surfaces.inp\n')
inp.write('*INCLUDE, Input=j101_01_74_outer_surfaces.inp\n')
inp.write('*TIE, name=tie_pos_y, TIED NSET=POLY-1.Y1, ADJUST=NO\n')
inp.write('posy, negy_inner\n')
inp.write('*TIE, name=tie_neg_x, TIED NSET=POLY-1.X0, ADJUST=NO\n')
inp.write('negx, posx_inner\n')
inp.write('*TIE, name=tie_neg_z, TIED NSET=POLY-1.Z0, ADJUST=NO\n')
inp.write('negz, posz_inner\n')
inp.write('** Section: plane_strain_basalt\n')
inp.write('*End Assembly\n')
inp.write('*INCLUDE, input=quartz_03.inp\n')
inp.write('*INCLUDE, input=plagioclase_03.inp\n')
inp.write('*INCLUDE, input=avg_granite_03.inp\n')
inp.write('*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23\n')
inp.write('*Initial Conditions, type=TEMPERATURE\n')
inp.write('ALL_outer, 25.\n')
inp.write('ALL_poly, 25.\n')
inp.write('** ----------------------------------------------------------------\n')
inp.write('**\n')
inp.write('** STEP: Heating\n')
inp.write('**\n')
inp.write('*Step, name=Heating\n')
inp.write('*Heat Transfer, end=PERIOD, deltmx=50.\n')
inp.write('0.5, 15., 0.002, 15.\n')
inp.write('*Dflux, OP=MOD\n')
inp.write('elementALL_poly, BFNU\n')
inp.write('elementALL_outer, BFNU\n')
inp.write('*Sfilm\n')
inp.write('negy, F , 25., 20.\n')
inp.write('negy_outer, F , 25., 20.\n')
inp.write('*Sradiate\n')
inp.write('negy, R, 25., 0.8\n')
inp.write('negy_outer, R, 25., 0.8\n')
inp.write('posy_outer, R, 25., 0.8\n')
inp.write('negx_outer, R, 25., 0.8\n')
inp.write('negz_outer, R, 25., 0.8\n')
inp.write('*Restart, write, frequency=1, overlay\n')
inp.write('*Output, field\n')
inp.write('*Element output\n')
inp.write('TEMP, IVOL, UVARM1\n')
inp.write('*Element Output, POSITION=NODES, directions=YES\n')
inp.write('HFL\n')
inp.write('*Node output\n')
inp.write('NT\n')
inp.write('*Output, history, variable=PRESELECT\n')
inp.write('*End Step\n')
inp.close()
```

Python script to calculate the thermal energy

Appendix2/script_calc_heat_aut.py

```
#######################################################################
# Script to calculate thermal energy in model
#######################################################################
#load packages
import math
import numpy
import argparse
from odbAccess import *
#######################################################################
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()            # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Script to calculate the thermal energy.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model', type=str, required=True, help='Name of the model.')
parser.add_argument('--heatout', type=str, required=True, help='Name of the output file in which the thermal energy is
      written.')
args = parser.parse_args(namespace=parameter)
#######################################################################
#initialization
aq=[]
bq=[]
cq=[]
gb=[]
hb=[]
gbo=[]
hbo=[]
dq=[]
kb=[]
toldq=[]
```

```
ap=[]
bp=[]
cp=[]
dp=[]
i_b=[]
i_bo=[]
toldp=[]
aa=[]
ba=[]
ca=[]
da=[]
i_a=[]
tolda=[]
kk=0
ll=[]
minus=0
tin=25.          # initial temperature in Celsius
eges=0.
frame=1
timeold=0.
e35=0.
Atriangle=5.32E-7   # Area of triangle of one side of tetrahedral element
Aquad=5.2E-6       # Area of rectangle of outer elements
######################################################################
#variables which have to be defined
denquartz=2649.           #density of quartz
denplagioclase=2703.       #density of plagioclase
denavg_granite=2667.36      #density of avg_granite
#
######################################################################
#input
inodb="j101_01_"+parameter.model+"_03_02_1000_poly_intpoint_heat"
inout=parameter.heatout
######################################################################
#Method to interpolate the cp value
def getcp (temp, cptable):
  z=1
  if temp<=cptable[0][1]:
    cp=cptable[0][0]
  else:
    if temp>=cptable[len(cptable)-1][1]:
      cp=cptable[len(cptable)-1][0]
    else:
      while z<len(cptable):
        if cptable[z-1][1]<=temp<=cptable[z][1]:
          cp=cptable[z-1][0]+(temp-cptable[z-1][1])*((cptable[z][0]-cptable[z-1][0])/(cptable[z][1]-cptable[z-1][1]))
          z=len(cptable)
        z=z+1
  return cp
#calculate thermal energy
def calcE (x, cp, d, tin, den):        #Q=roh*cp*V*deltaT
  if len(tin)==0:
    en=((d[1][x]-25)*cp*d[2][x]*den)  #d[1]=end Temperature of increment, d[2]=IVOL
  else:
    en=((d[1][x]-tin[x])*cp*d[2][x]*den)
  return en
#Method for calculating the average HFL on the boundary elements
def avgHFL (t, time):
  u=[[],[]]
  x=[[],[],[]]
  zz=0
  zx=0
  elementz=0
  #loop over all nodes which are on the front surface
  while zz<len(t[0]):
    elementz=t[0][zz]          #current element node ID
    zzz=0
    inx=0
    while zzz<len(x[0]):         #loop over already saved nodes
      if (x[0][zzz]==elementz):   #check if element ID is equal to saved element ID
        x[1][zzz]=x[1][zzz]+t[1][zz][1] #if yes add flux in y-direction
        x[2][zzz]=x[2][zzz]+1    #and update counter
        zzz=len(x[0])        #and end while
        inx=1
      zzz=zzz+1
    if inx==0:              #check if element ID is already saved
      x[0].append(t[0][zz])       #if not save the element ID
      x[1].append(t[1][zz][1])      #and the HFL in y-direction
      x[2].append(1)
    zz=zz+1
  while zx<len(x[0]):
    if (x[1][zx]<0):
      u[0].append(x[0][zx])
      u[1].append(x[1][zx]*(Atriangle/3)*time)  #and calculate Q=HFL2*deltatime*A
    zx=zx+1
  return u
#main
odb = openOdb(path=inodb+'.odb')              #open odb file
quartz=odb.rootAssembly.instances['POLY-1'].elementSets['QUARTZ']        #open element set quartz
plagioclase=odb.rootAssembly.instances['POLY-1'].elementSets['PLAGIOCLASE']    #open element set plagioclase
outer=odb.rootAssembly.instances['OUTER-1'].elementSets['OUTER_ELEMENT']     #open element set plagioclase
boundsurface=odb.rootAssembly.instances['POLY-1'].nodeSets['Y0']         #set containing front surface nodes
boundsurface_outer=odb.rootAssembly.instances['OUTER-1'].nodeSets['Y0']      #set containing front surface nodes of outer
      part
material_quartz=odb.materials['QUARTZ'].specificHeat              #read in Cp values of quartz
material_plagioclase=odb.materials['PLAGIOCLASE'].specificHeat          #read in Cp values of plagioclase
material_avg_granite=odb.materials['AVG_GRANITE'].specificHeat          #read in Cp values of plagioclase
cptable_quartz=material_quartz.table
cptable_plagioclase=material_plagioclase.table
cptable_avg_granite=material_avg_granite.table
lastFrame=odb.steps['Heating'].frames[-1]        #save last frame of the step
print "first frame ID: "+str(frame)
print "last frame ID: "+str(lastFrame.frameId)
f=open(inout+".txt",'w')                #open output file
f.write("Increment  Heating [J]  Flux over boundaries [J] Energy [J]  \n" )
while frame<=lastFrame.frameId:              #loop over all increments
#variable declaration
  yzp=0
  yxp=0
  yxb=0
  yxbo=0
  yhp=0
  yzq=0
  yxq=0
  yhq=0
  yza=0
  yxa=0
  yha=0
  einc=0
  minusinc=0
#main
  curFrame = odb.steps['Heating'].frames[frame]   #just load results from step "Heating"
  temp=curFrame.fieldOutputs['TEMP']           #get temperature values of all elements
  vol=curFrame.fieldOutputs['IVOL']          #get IVOL values of all elements
  heat=curFrame.fieldOutputs['HFL']          #get HFL values of all elements
  timec=curFrame.frameValue              #end time of current instance
  time=timec-timeold                  #increment of current instance
  field_plagioclase=temp.getSubset (region=plagioclase, position=INTEGRATION_POINT, elementType='DC3D4')     #TEMP values
      of elements in set plagioclase
  fieldvol_plagioclase=vol.getSubset (region=plagioclase, position=INTEGRATION_POINT, elementType='DC3D4')  #IVOL values
      of elements in set plagioclase
  fieldHFL_boundsurf=heat.getSubset (region=boundsurface, position=ELEMENT_NODAL, elementType='DC3D4')       #HFL values
      of nodes in set boundsurface
  fieldHFL_boundsurf_outer=heat.getSubset (region=boundsurface_outer, position=ELEMENT_NODAL, elementType='DC3D8')  #HFL
      values of nodes in set boundsurface_outer
  fieldValues_plagioclase=field_plagioclase.values  #load field values
  fieldvolVal_plagioclase=fieldvol_plagioclase.values
  fieldHFLVal_boundsurf=fieldHFL_boundsurf.values
  fieldHFLVal_boundsurf_outer=fieldHFL_boundsurf_outer.values
  field_quartz=temp.getSubset (region=quartz, position=INTEGRATION_POINT, elementType='DC3D4')       #TEMP values of
      elements in set quartz
  fieldvol_quartz=vol.getSubset (region=quartz, position=INTEGRATION_POINT, elementType='DC3D4')       #IVOL values of
      elements in set quartz
  fieldValues_quartz=field_quartz.values        #load field values
  fieldvolVal_quartz=fieldvol_quartz.values
  field_avg_granite=temp.getSubset (region=outer, position=INTEGRATION_POINT, elementType='DC3D8')     #TEMP values of
      elements in set avg_granite
  fieldvol_avg_granite=vol.getSubset (region=outer, position=INTEGRATION_POINT, elementType='DC3D8')       #IVOL values of
       elements in set avg_granite
  fieldValues_avg_granite=field_avg_granite.values  #load field values
  fieldvolVal_avg_granite=fieldvol_avg_granite.values
  #loop over all output lines
  for vq in fieldValues_quartz:           #index q stands for quartz
    aq.append(vq.elementLabel)            #save ID of the elements
    bq.append(vq.data)                #save TEMP value of the element
  for wq in fieldvolVal_quartz:
    cq.append(wq.data)
  #loop over all boundary nodes on poly surface
  for yb in fieldHFLVal_boundsurf:
    gb.append(yb.nodeLabel)             #index b stands for boundary
    hb.append(yb.data)
  #loop over all boundary nodes on outer surface
  for ybo in fieldHFLVal_boundsurf_outer:
    gbo.append(ybo.nodeLabel)           #index b stands for boundary
    hbo.append(ybo.data)              #index o stands for outer part
  dq.append(aq)
  dq.append(bq)
  dq.append(cq)
  i_b.append(gb)
  i_b.append(hb)
  i_bo.append(gbo)
  i_bo.append(hbo)
  for vp in fieldValues_plagioclase:          #index p stands for plagioclase
    ap.append(vp.elementLabel)
    bp.append(vp.data)
  for wp in fieldvolVal_plagioclase:
    cp.append(wp.data)
  dp.append(ap)
  dp.append(bp)
  dp.append(cp)
  #loop over avg_granite
  for va in fieldValues_avg_granite:          #index a stands for avg_granite
    aa.append(va.elementLabel)
    ba.append(va.data)
  for wa in fieldvolVal_avg_granite:
    ca.append(wa.data)
  da.append(aa)
  da.append(ba)
  da.append(ca)
        print "CP table plagioclase:"
  countcp=0
  while countcp<len(cptable_plagioclase):
          print str(cptable_plagioclase[countcp][0])+" "+str(cptable_plagioclase[countcp][1])
    countcp=countcp+1
#calculate energy in plagioclase per increment
  while yzp<len(dp[0]):             #loop over all elements in set plagioclase
    if (len(toldp)==0):
      cpp=getcp((dp[1][yzp]+tin)/2,cptable_plagioclase)
    else:
      cpp=getcp((dp[1][yzp]+toldp[yzp])/2,cptable_plagioclase)    #interpolate Cp values for certain temperatures
    einc=einc+calcE(yzp, cpp, dp, toldp, denplagioclase)  #sum energy over all elements
    yzp=yzp+1
#calculate energy in quartz per increment
  first=1
  while yzq<len(dq[0]):
    if (len(toldq)==0):
      cpq=getcp((dq[1][yzq]+tin)/2,cptable_quartz)
    else:
      cpq=getcp((dq[1][yzq]+toldq[yzq])/2,cptable_quartz)
    einc=einc+calcE(yzq, cpq, dq, toldq, denquartz)
    yzq=yzq+1
#calculate energy in avg_granite per element
  while yza<len(da[0]):
    if (len(tolda)==0):
      cpa=getcp((da[1][yza]+tin)/2,cptable_avg_granite)
    else:
      cpa=getcp((da[1][yza]+tolda[yza])/2,cptable_avg_granite)
    einc=einc+calcE(yza, cpa, da, tolda, denavg_granite)
    yza=yza+1
  eges=eges+einc             #cumulate energy over all increments
#delete old temperature field
  del toldp[:]
  del toldq[:]
  del tolda[:]
#save current temperature field on variable old temperature field
```

```
  while yhq<len(dq[0]):          #loop over quartz
    toldq.append(dq[1][yhq])
    yhq=yhq+1
  while yhp<len(dp[0]):          #loop over plagioclase
    toldp.append(dp[1][yhp])
    yhp=yhp+1
  while yha<len(da[0]):          #loop over plagioclase
    tolda.append(da[1][yha])
    yha=yha+1
#calculate HFL out of the grid
  kb=avgHFL(i_b, time)
  while yxb<len(kb[0]):
        minusinc=minusinc+kb[1][yxb]
    yxb=yxb+1
#calculate HFL out of the outer part
  kbo=avgHFL(i_bo, time)
  while yxbo<len(kbo[0]):
    minusinc=minusinc+kbo[1][yxbo]
    yxbo=yxbo+1
  minus=minus+minusinc
  del aq[:]
  del bq[:]
  del cq[:]
  del dq[:]
  del gb[:]
  del hb[:]
  del i_b[:]
  del i_bo[:]
  del ap[:]
  del bp[:]
  del cp[:]
  del dp[:]
  del aa[:]
  del ba[:]
  del ca[:]
  del da[:]
#write results to output file
  f.write(str(frame)+"    "+str(einc)+"    "+str(minusinc)+"    "+str(einc+minusinc)+"\n")
  frame=frame+1
  timeold=timec
print "eges "+str(eges)
print "minus "+str(minus)
print "(eges-minus) "+ str(eges-minus)
f.write("Heating: "+str(eges)+"\n")
f.write("Flux over boundaries: "+str(minus)+"\n")
f.write("Total energy input: "+str(eges-minus))
f.close()
```

Abaqus cooling input file

Appendix2/j101_01_cooling.inp

```
**Thermal 3D two component cooling model
*RESTART, READ, END STEP
**
** STEP: Heating
**
*Step, name=cooling
*Heat Transfer, end=PERIOD, deltmx=50.
2.e-2, 3600., 0.0000001, 3600.
**
** LOADS
**
**
*Sfilm
negy, F , 25., 20.
negy_outer, F , 25., 20.
*Sradiate
negy, R, 25., 0.8
```

```
negy_outer, R, 25., 0.8
posy_outer, R, 25., 0.8
negx_outer, R, 25., 0.8
negz_outer, R, 25., 0.8
*Restart, write, frequency=0
*Output, field
*Element output
TEMP, IVOL
*Element Output, POSITION=NODES, directions=YES
HFL
*Node output
NT
*Output, history, variable=PRESELECT
*End Step
```

FORTRAN cooling subroutine

Appendix2/BF_cooling.f

```
C
      SUBROUTINE DFLUX(FLUX,SOL,KSTEP,KINC,TIME,NOEL,NPT,COORDS,
     1 JLTYP,TEMP,PRESS,SNAME)
C
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION FLUX(2), TIME(2), COORDS(3)
      CHARACTER*80 SNAME
       integer :: I
C
      FLUX(1)=0
      RETURN
      END
C
C
      SUBROUTINE UVARM(UVAR,DIRECT,T,TIME,DTIME,CMNAME,ORNAME,
     1 NUVARM,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,NDI,NSHR,COORD,
     2 JMAC,JMATYP,MATLAYO,LACCFLA)
C
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME,ORNAME
      CHARACTER*3 FLGRAY(15)
      DIMENSION UVAR(NUVARM),DIRECT(3,3),T(3,3),TIME(2)
      DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)
C     The dimensions of the variables FLGRAY, ARRAY and JARRAY
C     must be set equal to or greater than 15.
      UVAR(1)=0
      RETURN
      END
```

Python script to create the *Abaqus* linear elastic model

Appendix2/create_stress_input.py

```
##########################################################################
# Script to automatically create the Abaqus stress input file
##########################################################################
import sys                              # better than:    import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
# Class containing all Parameters which are parsed
```

```
class Parameters:
    pass
parameter = Parameters()           # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created stress input file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('—model',    type=str, required=True, help='Name of the model.')
args = parser.parse_args(namespace=parameter)
######################################################################
# Open and write in Abaqus input file
inp=open('j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint.inp', 'w')
inp.write('**Automatic generated heat input file for j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint.inp\n')
inp.write('*Part, name=poly\n')
inp.write('*INCLUDE, Input=poly_74_mesh_static.inp\n')
inp.write('*INCLUDE, Input=quartz_elements_'+str(parameter.model)+'.inp\n')
inp.write('*INCLUDE, Input=plagioclase_elements_'+str(parameter.model)+'.inp\n')
inp.write('*Solid Section, elset=quartz, material=quartz\n')
inp.write('*Solid Section, elset=plagioclase, material=plagioclase\n')
inp.write('*End Part\n')
inp.write('*Part, name=outer\n')
inp.write('*INCLUDE, Input=outer_74_mesh_C3D8R.inp\n')
inp.write('*Elset, elset=outer_element, generate\n')
inp.write('1, 281988, 1\n')
inp.write('*Solid Section, elset=outer_element, material=avg_granite\n')
inp.write('*End Part\n')
inp.write('*Assembly, name=Assembly\n')
inp.write('*Instance, name=poly-1, part=poly\n')
inp.write('          0.0,              0.,              0.0\n')
inp.write('*End Instance\n')
inp.write('*Instance, name=outer-1, part=outer\n')
inp.write(' 0.,              0.3,          -0.07\n')
inp.write('       0.,              0.3,          -0.07, 0.577350269189626, -0.277350269189626, 0.507350269189626,          120.\n'
       )
inp.write('*End Instance\n')
inp.write('*Nset, nset=FIX, instance=poly-1\n')
inp.write(' 1\n')
inp.write('*Nset, nset=ALL_poly, generate, instance=poly-1\n')
inp.write(' 1, 703553, 1\n')
inp.write('*Elset, elset=elementALL_poly, generate, instance=poly-1\n')
inp.write(' 1, 4060685, 1\n')
inp.write('*Nset, nset=ALL_outer, generate, instance=outer-1\n')
inp.write(' 1, 296620, 1\n')
inp.write('*Elset, elset=elementALL_outer, generate, instance=outer-1\n')
inp.write(' 1, 281988, 1\n')
inp.write('*INCLUDE, Input=j101_01_69_poly_surfaces.inp\n')
inp.write('*INCLUDE, Input=j101_01_74_outer_surfaces.inp\n')
inp.write('*TIE, name=tie_pos_y, TIED NSET=POLY-1.Y1,  ADJUST=NO\n')
inp.write('posy, negy_inner\n')
inp.write('*TIE, name=tie_neg_x, TIED NSET=POLY-1.X0, ADJUST=NO\n')
inp.write('negx, posx_inner\n')
inp.write('*TIE, name=tie_neg_z, TIED NSET=POLY-1.Z0, ADJUST=NO\n')
inp.write('negz, posz_inner\n')
inp.write('** Section: plane_strain_basalt\n')
inp.write('*End Assembly\n')
inp.write('*INCLUDE, input=quartz_03_elastic.inp\n')
inp.write('*INCLUDE, input=plagioclase_03_elastic.inp\n')
inp.write('*INCLUDE, input=avg_granite_03_elastic.inp\n')
inp.write('*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23\n')
inp.write('*Boundary\n')
inp.write('z0_outer, ZSYMM\n')
inp.write('x0_outer, XSYMM\n')
inp.write('poly-1.z1, ZSYMM\n')
inp.write('poly-1.x1, XSYMM\n')
inp.write('*Initial Conditions, type=TEMPERATURE\n')
inp.write('ALL_outer, 25.\n')
inp.write('ALL_poly, 25.\n')
inp.write('** ----------------------------------------------------------------\n')
inp.write('**\n')
inp.write('** STEP: Heating\n')
inp.write('**\n')
inp.write('*Step, name=Heating\n')
inp.write('*Static\n')
inp.write('0.3, 15., 0.000001, 15.\n')
inp.write('*TEMPERATURE, FILE=j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat.odb, BSTEP=1\n')
inp.write('ALL_poly\n')
inp.write('ALL_outer\n')
inp.write('*Restart, write, frequency=0\n')
inp.write('*Output, field\n')
inp.write('*Element output\n')
inp.write('S, SP, IVOL, E\n')
inp.write('*Node output\n')
inp.write('NT, U, CF, RF\n')
inp.write('*Output, history, variable=PRESELECT\n')
inp.write('*End Step\n')
inp.write('** ----------------------------------------------------------------\n')
inp.write('**\n')
inp.write('** STEP: Cooling\n')
inp.write('**\n')
inp.write('*Step, name=Cooling\n')
inp.write('*Static\n')
inp.write('2.e-2, 3600., 0.0000001, 3600.\n')
inp.write('*TEMPERATURE, FILE=j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling.odb, BSTEP=1\n')
inp.write('ALL_poly\n')
inp.write('ALL_outer\n')
inp.write('*Restart, write, frequency=0\n')
inp.write('*Output, frequency=5, field\n')
inp.write('*Element output\n')
inp.write('S, SP, IVOL, E\n')
inp.write('*Node output\n')
inp.write('NT, U, CF, RF\n')
inp.write('*Output, history, variable=PRESELECT\n')
inp.write('*End Step\n')
inp.close()
```

Calculation of the submodel

Abaqus inputfile for the linear elastic submodel

Appendix2/j101_04_74_03_02_1000_poly_intpoint.inp

```
*Heading
** Submodel
** Generated by: Abaqus/CAE 6.12-1
*Part, name=poly
** Include only elements within a certain radius, created by write_poly_input_radius.cpp
*INCLUDE, Input=submodel_grains_input_1.inp
*INCLUDE, Input=quartz_elements_04_74.inp
*INCLUDE, Input=plagioclase_elements_04_74.inp
*Solid Section, elset=quartz, material=quartz
1.,
*Solid Section, elset=plagioclase, material=plagioclase
1.,
*End Part
*Assembly, name=Assembly
*Instance, name=poly-1, part=poly
          0.0,              0.,             0.0
*End Instance
*Nset, nset=FIX, instance=poly-1
 1
*INCLUDE, Input=j101_04_surfaces.inp
*Submodel, TYPE=NODE
BC_submodel_node_list
*End Assembly
*INCLUDE, input=quartz_03_elastic.inp
*INCLUDE, input=plagioclase_03_elastic.inp
*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23
*Boundary
z1, ZSYMM
x1, XSYMM
*Initial Conditions, type=TEMPERATURE
ALL_poly, 25.
```

```
*Step, name=Heating
*STATIC
0.3, 15., 0.0000000001, 15.
*BOUNDARY, STEP=1, SUBMODEL
BC_submodel_node_list, 1, 3
*TEMPERATURE, FILE=j101_01_74_03_02_1000_poly_intpoint.odb, BSTEP=1, MIDSIDE
ALL_poly
*Restart, write, frequency=0
*Output,frequency=1, field
*Element output
S, SP, IVOL, E
*Node output
NT, U, CF, RF
*Output, history, variable=PRESELECT
*End Step
*Step, name=Cooling
*STATIC
2.e-2, 3600., 0.0000001, 3600.
*BOUNDARY, STEP=2, SUBMODEL
BC_submodel_node_list, 1, 3
*TEMPERATURE, FILE=j101_01_74_03_02_1000_poly_intpoint.odb, BSTEP=2, MIDSIDE
ALL_poly
*Restart, write, frequency=0
*Output,frequency=5, field
*Element output
S, SP, IVOL, E
*Node output
NT, U, CF, RF
*Output, history, variable=PRESELECT
*End Step
```

C++ script to calculate the elements which are within a certain radius

Appendix2/write_poly_input_radius.cpp

```
//
//create input file of grains within a certain radius
//
#include <fstream>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#define _USE_MATH_DEFINES
#include <math.h>
#include <cmath>
#include <vector>
using std::cout;
using std::endl;
using std::string;
using std::ifstream;
using std::ofstream;
using std::istringstream;
using std::ostringstream;
using std::ios;
using std::istream_iterator;
using std::vector;
using std::back_inserter;
//VARIABLES which have to be defined
int amountnodes=703553;      //amount of nodes
int amountelements=4060685;    //amount of elements
int maxelement=1400000000;     //maximum elements
const char* inputfile_elementset="poly_74_mesh.inp";     //name of the input file containing the element sets
const char* quartz_elementset="quartz_elements_69.inp";    //name of the input file containing the quartz element sets
const char* plagioclase_elementset="plagioclase_elements_69.inp"; //name of the input file containing the plagioclase
     element sets
```

```
const char* new_quartz_elementset="quartz_elements_04_74.inp";  //name of the new input file containing the quartz
      element sets
const char* new_plagioclase_elementset="plagioclase_elements_04_74.inp";  //name of the new input file containing the
      plagioclase element sets
std::vector<std::vector<std::vector<double> > >dat; //variable to save the geometric data of the microstructure
//std::vector<std::vector<double> >boundCon;     //variable to save the bounding container of the different polyhedron
double radius=0.03;      //radius for searched elements in meter
double sfactor=0.4;      //factor which the Neper mesh is scaled
//INIT MEHTHODS
//
std::vector<std::vector<std::vector<int> > > getelemnode ();
std::vector<std::string> nodes;
//std::vector<std::vector<double> > getBS ();
std::vector<std::vector<std::vector<double> > >getpolyhedron ();
std::vector<int> readin(const char* input);
//MAIN
int main (void){
// variable declaration
  std::vector<std::vector<std::vector<int> > > element;
  ofstream o;
  ofstream p;
  ofstream q;
  int poly;
  int curelement;
  bool old;
  double curvector;      //length of the vector which point from origin to the middle point of the polyhedra
  string filename;
  std::vector<int> selement;  //list of elements which are within the radius
  std::vector<int> quartzelement; //element sets of quartz elements
  std::vector<int> plagioelement; //element sets of plagioclase elements
//body
  element=getelemnode();
  cout<<"Last element set= "<<element[1].size()<<endl;
  cout<<"Amount  of elements in last element set= "<<element[1][29999].size()<<endl;
  cout<<"Last element in last element set= "<<element[1][29999][element[1][29999].size()-1]<<endl;
// read in data of grains
  dat=getpolyhedron();              //vector which returns all the results
  quartzelement=readin(quartz_elementset);  //read in quartz elements
  plagioelement=readin(plagioclase_elementset); //read in plagioclase elements
//Loop over all polysets
  for (int i=0; i<dat[3].size(); i++){
    //search current element set
    curvector=sqrt(pow((0.08-dat[3][i][0]),2)+pow((dat[3][i][1]),2)+pow((0.08-dat[3][i][2]),2));
    if (curvector<=radius){
      selement.push_back(i+1);
      cout<<"Poly "<<i+1<<" is inside the radius"<<endl;
    }
  }
  filename="submodel_grains_radius_0030.inp";
  o.open(filename.c_str());
  for (int k=0; k<nodes.size(); k++){ //write node data to new input file
    o<<nodes[k]<<endl;
  }
// write element data in input file
  o<<"*Element, type=DC3D4"<<endl;
  for (int y=0; y<selement.size(); y++){
    poly=selement[y];
    for (int z=0; z<element[1][poly-1].size(); z++){
        curelement=element[1][poly-1][z];
        o<<curelement<<", "<<element[0][curelement-1][0]<<", "<<element[0][curelement-1][1]<<", "<<element[0][curelement
      -1][2]<<", "<<element[0][curelement-1][3]<<endl;
    }
  }
  for (int y=0; y<selement.size(); y++){
    poly=selement[y];
    o<<"*Elset, elset=poly"<<selement[y]<<endl;
    for (int z=1; z<=element[1][poly-1].size(); z++){
      if (z==(element[1][poly-1].size())){
        o<<element[1][poly-1][z-1]<<endl;
      }
      else{
        if (z%10==0){
          o<<element[1][poly-1][z-1]<<","<<endl;
        }
        else{
          o<<element[1][poly-1][z-1]<<", ";
        }
      }
    }
  }
  o.close();
  int lastID=0;
  p.open(new_quartz_elementset);
  p<<"*ELSET, Elset=quartz"<<endl;
  cout<<"quartzelement.size()="<<quartzelement.size()<<endl;
  for (int w=0; w<quartzelement.size(); w++){
    for (int z=lastID; z<selement.size(); z++){
      if (quartzelement[w]==selement[z]){
        p<<"poly"<<selement[z]<<endl;
        lastID=z;
        z=selement.size();
      }
      else{
        if(quartzelement[w]<selement[z]){
          z=selement.size();
        }
      }
    }
  }
  p.close();
  lastID=0;
  q.open(new_plagioclase_elementset);
  q<<"*ELSET, Elset=plagioclase"<<endl;
  cout<<"plagioelement.size()="<<plagioelement.size()<<endl;
  for (int w=0; w<plagioelement.size(); w++){
    for (int z=lastID; z<selement.size(); z++){
      if (plagioelement[w]==selement[z]){
        q<<"poly"<<selement[z]<<endl;
        lastID=z;
        z=selement.size();
      }
      else{
        if(plagioelement[w]<selement[z]){
          z=selement.size();
        }
      }
    }
  }
  q.close();
  return 0;
}
// Method to read in and save the elements and the corresponding nodes
std::vector<std::vector<std::vector<int> > > getelemnode (){
  std::vector<std::vector<std::vector<int> > > elementdata;
  int deletlines=1;    //lines which should be omitted
  ifstream f;
  string s;
  std::vector<std::string> v;
  std::vector<std::vector<int> > elements;
  std::vector<std::vector<int> > polydata;
  std::vector<std::vector<std::vector<int> > > retout;
  f.open(inputfile_elementset, ios::in);
  if (f.good()==false){        //check if input file exist
              f.close();         //if not close file
              cout<<"inp file is missing!!!"<<endl; //print error message and
              exit(1);          //exit program
        }
  while (!f.eof())
  { getline (f,s);
    v.push_back(s);
  }
  f.close();
  nodes.push_back(v[0]);
  v.erase(v.begin());
  for (int i=0; i<amountnodes; i++){  //save all nodes including the coordinates
    nodes.push_back(v[i]);
  }
  for (int h=amountnodes+1;h<(amountnodes+amountelements+1);h++){ //save the elements and the corresponding nodes numbers
    std::vector<std::string> temp2;
    istringstream iss2(v[h]);
    copy(istream_iterator<string>(iss2),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp2));
    std::vector<int> row2;
    for (int j=1; j<5; j++){
      string str2=temp2[j];
        if ((str2.length()-1)==',')
          str2.erase(str2.length()-1);
        row2.push_back(atof(str2.c_str()));
    }
    elements.push_back(row2);
  }
  bool newpoly=true;
  unsigned int loc=6;
  bool em;
  bool firstem=false;
  int skip=0;
  std::vector<int> row3;
  // save the polyhedra and the corresponding element number
  for (unsigned int k=amountnodes+amountelements; k<v.size(); k++){      //loop over whole file
    std::vector<std::string> temp3;
    istringstream iss3(v[k]);
    copy(istream_iterator<string>(iss3),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp3));
    if (firstem==true){
      skip++;
      em=temp3.empty();            //em=true if the line is empty
    if ((em==false)&&(newpoly==true)){       //check if line is not empty and a new poly is detected
        loc=temp3[1].find("poly");
      }
      if (newpoly==true){          //check if a new poly is required
        if (loc != 6)           //if "poly" is not on the 7th digit end loop
          k=v.size()+2;
        else
          newpoly=false;
      }
      else{
        if (em==true){            //if the line is empty erase the certain line
          polydata.push_back(row3);
          row3.erase(row3.begin(), row3.end());
          newpoly=true;
        }
        else{
          for (unsigned int j=0; j<temp3.size(); j++){     //
          string str3=temp3[j];          // save each element
            if ((str3.length()-1)==',')      // if element contain , delete ,
              str3.erase(str3.length()-1);
            if (str3.empty()==false)
               row3.push_back(atof(str3.c_str()));
          }
        }
      }
    }
    if (skip==0){
      firstem=temp3.empty();
    }
  }
  retout.push_back(elements);
  retout.push_back(polydata);
  return retout;
}
// Method which read data from the Neper file
std::vector<std::vector<std::vector<double> > >getpolyhedron () {
  int omittedlines=6; // insert the line which should be omitted
  ifstream f;
  string s;
  std::vector<std::string> v;
  int amountvertex; //amount of vertex
  int amountedge;   //amount of edges
  int amountface;   //amount of faces
  int amountpolyhedron; //amount of polyhedron
  std::vector<std::vector<double> > vertex; // index of vertex is the column; row is in the form: Amount of edges, edge
     number 1, edge number 2, ..., x- , y-, z- value of vertex
  std::vector<std::vector<double> > edges; // index of edge is the column; row is in the form: vertex 1, vertex 2, number
      of faces, face 1, face 2, ...
  std::vector<std::vector<double> > faces; // index of face is the column; row is in the form: polyhedron 1, polyhedron
     2, number of vertices, ver_1, ver_2, ..., edge_1, edge_2, ...
  std::vector<std::vector<double> > polyhedron; // poly_centre_x, poly_centre_y, poly_centre_z, number of faces, face 1,
     face 2, ...
  std::vector<std::vector<std::vector<double> > >ret; //vector which returns all the results
  f.open("poly_69_mesh.tess", ios::in);
  if (f.good()==false){        //check if tess file exist
    f.close();          //if not close file
    cout<<"tess file is missing!!!"<<endl;  //write error message
    exit(1);          //end program
  }
  while (!f.eof())
  { getline (f,s);
    v.push_back(s);
  }
  f.close();
  for (int j=1; j<=omittedlines; j++){
    v.erase(v.begin());
  }
// save vertex
  int hv=0;
  amountvertex=atoi(v[hv].c_str());
  v.erase(v.begin());
  for (int j=0; j<3*amountvertex; j=j+3){
    std::vector<std::string> temp1; // amount of edges, edge number 1, edge number 2, ...
    std::vector<std::string> temp2;
    istringstream iss1(v[j+1]);
    copy(istream_iterator<string>(iss1), // split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp1));
    std::vector<double> row;
    istringstream iss2(v[j+2]);
    copy(istream_iterator<string>(iss2),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp2));
    for (int n=0; n<=2; n++){  // insert coordinates of vertex
      row.push_back(sfactor*atof(temp2[n].c_str()));  //save component and scale it with factor
    }
    vertex.push_back(row);  //insert row in 2dimensional vector
  }
  cout<<"z koordinate von vertex 1:"<<vertex[0][2]<<endl;
  ret.push_back(vertex);
  int he=hv+amountvertex*3+1;
  amountedge=atoi(v[he].c_str());
  for (int j=he+1; j<he+1+4*amountedge; j=j+4){
    std::vector<std::string> temp3;
    std::vector<std::string> temp4;
    istringstream iss3(v[j+1]);
    copy(istream_iterator<string>(iss3), //split of the single lines
        istream_iterator<string>(),
      back_inserter<vector<string> >(temp3));
    std::vector<double> row2;
    for (int n=0; n<2;n++){ //insert vertex 1, vertex 2
      row2.push_back(atof(temp3[n].c_str()));
    }
    istringstream iss4(v[j+2]);
    copy(istream_iterator<string>(iss4),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp4));
    row2.push_back(atof(temp4[0].c_str())); // insert amount of faces
    for (int n=1; n<=atoi(temp4[0].c_str()); n++){
      row2.push_back(atof(temp4[n].c_str()));
    }
    edges.push_back(row2); //insert row2 in 2dimemsional vector
  }
  ret.push_back(edges);
// save faces
  int hf=he+amountedge*4+2;
  amountface=atoi(v[hf].c_str());
  for (int j=hf+1; j<hf+1+7*amountface; j=j+7){
    std::vector<std::string> temp5;
    std::vector<std::string> temp6;
    std::vector<std::string> temp7;
    istringstream iss5(v[j+2]);
    copy(istream_iterator<string>(iss5), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp5));
    std::vector<double> row3;
    for (int n=0; n<4;n++){ //insert face eq_a, face eq_b, face eq_c, face eq_d
      row3.push_back(atof(temp5[n].c_str()));
    }
    istringstream iss6(v[j+3]);
    copy(istream_iterator<string>(iss6),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp6));
    row3.push_back(atof(temp6[0].c_str())); // insert amount of vertex
    for (int n=1; n<=atoi(temp6[0].c_str()); n++){  //insert ver_1, ver_2, ...
      row3.push_back(atof(temp6[n].c_str()));
    }
    faces.push_back(row3); //insert row3 in 2dimemsional vector
  }
  ret.push_back(faces);
// save polyhedron
  int hp=hf+amountface*7+2;
  amountpolyhedron=atoi(v[hp].c_str());
  for (int j=hp+1; j<hp+1+3*amountpolyhedron; j=j+3){
    std::vector<std::string> temp8;
    std::vector<std::string> temp9;
    istringstream iss8(v[j]);
    copy(istream_iterator<string>(iss8), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp8));
    std::vector<double> row4;
    for (int n=1; n<4;n++){ //insert coordinates of the middle point of the polyhedron
      row4.push_back(sfactor*atof(temp8[n].c_str()));   //save component and scale it with factor
    }
    istringstream iss9(v[j+2]);
    copy(istream_iterator<string>(iss9),
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp9));
    row4.push_back(atof(temp9[0].c_str())); // insert amount of faces
    for (int n=1; n<=atoi(temp9[0].c_str()); n++){  //insert face_1, face_2, ...
      row4.push_back(atof(temp9[n].c_str()));
//    cout<<"fist face" << row4[0]<<endl;
    }
    polyhedron.push_back(row4); //insert row 4 in 2dimensionl vector
  }
  ret.push_back(polyhedron);
 return ret;
}
//Method to read in the element sets
std::vector<int> readin(const char* input){
  std::vector<int> elsets;
  ifstream f;
  std::vector<std::string> v;
  string s;
  f.open(input, ios::in);
  if (f.good()==false){   //check if tess file exist
    f.close();        //if not close file
    cout<<"Element set file is missing!!!"<<endl; //write error message
    exit(1);        //end program
  }
  while (!f.eof())
  { getline (f,s);
    v.push_back(s);
  }
  f.close();
  v.erase(v.begin());
  for (int j=0; j<v.size(); j++){
    v[j].erase(0,4);
    elsets.push_back(atoi(v[j].c_str()));
  }
  return elsets;
}
```

Anisotropic model

Abaqus input file for the heat transfer model with anisotropic quartz material behavior

Appendix2/j101_01_76_06_02_2000_poly_intpoint.inp

```
*Heading
** Thermal model for anisotropic material
** Generated by: Abaqus/CAE 6.12-1
*Part, name=poly
*INCLUDE, Input=poly_74_mesh.inp
*INCLUDE, Input=quartz_elements_69.inp
*INCLUDE, Input=plagioclase_elements_69.inp
** include orientation created by create_orientation.cpp
*INCLUDE, Input=orientation_1_76.inp
*End Part
*Part, name=outer
*INCLUDE, Input=outer_74_mesh_linear.inp
*Elset, elset=outer_element, generate
1, 281988, 1
*Solid Section, elset=outer_element, material=avg_granite
1.,
*End Part
*Assembly, name=Assembly
*Instance, name=poly-1, part=poly
        0.0,            0.,             0.0
*End Instance
*Instance, name=outer-1, part=outer
  0.,           0.3,         -0.07
        0.,           0.3,         -0.07, 0.577350269189626, -0.277350269189626, 0.507350269189626,          120.
*End Instance
*Nset, nset=FIX, instance=poly-1
 1
*Nset, nset=ALL_poly, generate, instance=poly-1
 1, 703553, 1
*Elset, elset=elementALL_poly, generate, instance=poly-1
 1, 4060685, 1
*Nset, nset=ALL_outer, generate, instance=outer-1
 1, 296620, 1
*Elset, elset=elementALL_outer, generate, instance=outer-1
 1, 281988, 1
*INCLUDE, Input=j101_01_69_poly_surfaces.inp
*INCLUDE, Input=j101_01_74_outer_surfaces.inp
*TIE, name=tie_pos_y, TIED NSET=POLY-1.Y1, ADJUST=NO
posy, negy_inner
*TIE, name=tie_neg_x, TIED NSET=POLY-1.X0, ADJUST=NO
negx, posx_inner
*TIE, name=tie_neg_z, TIED NSET=POLY-1.Z0, ADJUST=NO
negz, posz_inner
*End Assembly
**Include materials
*INCLUDE, input=quartz_06.inp
*INCLUDE, input=plagioclase_03.inp
*INCLUDE, input=avg_granite_03.inp
*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23
*Initial Conditions, type=TEMPERATURE
ALL_outer, 25.
ALL_poly, 25.
*Step, name=Heating
*Heat Transfer, end=PERIOD, deltmx=50.
0.5, 15., 0.002, 15.
*Dflux, OP=MOD
elementALL_poly, BFNU
elementALL_outer, BFNU
*Sfilm
negy, F , 25., 20.
negy_outer, F , 25., 20.
*Sradiate
negy, R, 25., 0.8
negy_outer, R, 25., 0.8
posy_outer, R, 25., 0.8
```

```
negx_outer, R, 25., 0.8
negz_outer, R, 25., 0.8
*Output, field
*Element output
TEMP, IVOL, UVARM1
*Element Output, POSITION=NODES, directions=YES
HFL
*Node output
NT
*Output, history, variable=PRESELECT
*End Step
*Step, name=Cooling
*Heat Transfer, end=PERIOD, deltmx=200.
10., 3600., 0.036, 3600.
*Dflux, op=NEW
*Sfilm
negy, F , 25., 20.
negy_outer, F , 25., 20.
*Sradiate
negy, R, 25., 0.8
negy_outer, R, 25., 0.8
posy_outer, R, 25., 0.8
negx_outer, R, 25., 0.8
negz_outer, R, 25., 0.8
*Restart, write, frequency=0
*Output,frequency=5, field
*Element output
TEMP, IVOL, HFL
*Node output
NT
*Output, history, variable=PRESELECT
*End Step
```

C++ script to create a random orientation of quartz grains

Appendix2/create_orientation.cpp

```
//
//Script which creates random orientations for the Abaqus model
//
#include <fstream>
#include <vector>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#define _USE_MATH_DEFINES
#include <math.h>
using namespace std;
//
//global variable
//
ofstream myfile;
//
//Method which creates a vector containing the polyhedra and the corresponding phase
void getori (int amountofori){
  int alpha, beta, gamma;
  time_t t;
  double M[3][3];
  double a[3];
  double b[3];
  double x[3]={1,0,0};
  double y[3]={0,1,0};
  time(&t);
  srand((unsigned int)t);
  for (int i=0; i<amountofori; i++){
    alpha= int(rand() % 360 + 1);    //create alpha angle between 1 and 360
    beta= int(rand() % 180 + 1);     //create beta angle between 1 and 180
    gamma= int(rand() % 360 + 1);    //create gamma angle between 1 and 360
    myfile<<"**alpha= "<<alpha<<" beta= "<<beta<<" gamma= "<<gamma<<endl;
    M[0][0]=cos(alpha)*cos(gamma)-sin(alpha)*cos(beta)*sin(gamma);
    M[0][1]=sin(alpha)*cos(gamma)+cos(alpha)*cos(beta)*sin(gamma);
    M[0][2]=sin(beta)*sin(gamma);
    M[1][0]=-cos(alpha)*sin(gamma)-sin(alpha)*cos(beta)*cos(gamma);
    M[1][1]=-sin(alpha)*sin(gamma)+cos(alpha)*cos(beta)*cos(gamma);
    M[1][2]=sin(beta)*cos(gamma);
    M[2][0]=sin(alpha)*sin(beta);
    M[2][1]=-cos(alpha)*sin(beta);
    M[2][2]=cos(beta);
    for (int k=0; k<3; k++){
      a[k]=0;
      b[k]=0;
    }
    for (int k=0; k<3; k++){
      for (int j=0; j<3; j++){
        a[k]+=(M[k][j]*x[j]);
        b[k]+=(M[k][j]*y[j]);
      }
    }
    myfile<<"*ORIENTATION, name=ori"<<i+1<<", DEFINITION=COORDINATES"<<endl;
    myfile<<a[0]<<", "<<a[1]<<", "<<a[2]<<", "<<b[0]<<", "<<b[1]<<", "<<b[2]<<endl;
  }
}
void assignpoly (int amountpoly, int amountori){
  ifstream f;
  time_t t;
  string s;
  std::vector<std::string> v;
  std::vector<int> phases;
  f.open("phases_69.txt", ios::in);
  if (f.good()==false){        //check if tess file exist
    f.close();          //if not close file
    cout<<"tess file is missing!!!"<<endl;  //write error message
    exit(1);          //end program
  }
  while (!f.eof())
  { getline (f,s);
    v.push_back(s);
  }
  f.close();
  v.erase(v.begin());
  for (int l=0; l<amountpoly; l++){
    std::vector<std::string> temp1;
    istringstream iss1(v[l]);
    copy(istream_iterator<string>(iss1), // split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp1));
    phases.push_back(atof(temp1[1].c_str()));
  }
  cout<<"phase of poly1: "<<phases[0]<<endl;
  time(&t);
  srand((unsigned int)t);
  int ori;
  for (int i=0; i<amountpoly; i++){
    ori= int(rand() % amountori + 1);    //create number between 1 and amount of orientations
    if (phases[i]==0){
      myfile<<"*SOLID SECTION, elset=poly"<<i+1<<", material=quartz, orientation=ori"<<ori<<endl;
    }
    else{
      if (phases[i]==1){
        myfile<<"*SOLID SECTION, elset=poly"<<i+1<<", material=plagioclase"<<endl;
      }
      else{
        cout<<"ERROR!!!"<<endl;
      }
    }
  }
}
//
```

```
//main
int main() {
  int amountofori;
  int amountofpoly;
  myfile.open("orientation_1_76.inp");
  cout<<"Insert amount of orientations: ";
  cin>>amountofori;
  getori (amountofori);
  cout<<"Insert amount of polyhedra: ";
  cin>>amountofpoly;
  assignpoly (amountofpoly, amountofori);
  myfile.close();
  return 0;
}
```

Quartz anisotropic input file

Appendix2/quartz_06_elastic.inp

```
*Material, name=quartz
*CONDUCTIVITY, TYPE=ORTHO
6.44, 6.44, 12.31,  25.
6.00, 6.00, 10.84,  50.
5.30, 5.30, 8.80, 100.
4.78, 4.78, 7.53, 150.
4.40, 4.40, 6.62, 200.
4.05, 4.05, 5.94, 250.
3.76, 3.76, 5.42, 300.
3.53, 3.53, 4.98, 350.
3.34, 3.34, 4.67, 400.
3.15, 3.15, 4.37, 450.
3.02, 3.02, 4.13, 500.
2.95, 2.95, 4.04, 520.
2.89, 2.89, 3.94, 540.
2.82, 2.82, 3.86, 560.
2.93, 2.93, 3.88, 572.
3.00, 3.00, 3.90, 573.
3.06, 3.06, 3.92, 574.
3.21, 3.21, 3.94, 580.
3.24, 3.24, 3.96, 600.
3.30, 3.30, 4.01, 650.
3.37, 3.37, 4.07, 700.
3.45, 3.45, 4.13, 750.
3.51, 3.51, 4.17, 800.
*Density
 2649.,
*ELASTIC,  TYPE=ANISOTROPIC
87.66E+9, 6.84E+9,  87.66E+9, 13.72E+9, 13.72E+9, 106.33E+9,  0.00, 0.00
0.00, 40.91E+9, 0.00, 0.00, 0.00, 18.18E+9, 56.69E+9, 18.18E+9
-18.18E+9,  0.00, 0.00, 0.00, 56.69E+9, 25.
87.13E+9, 5.27E+9,  87.13E+9, 12.75E+9, 12.75E+9, 103.92E+9,  0.00, 0.00
0.00, 41.48E+9, 0.00, 0.00, 0.00, 18.24E+9, 55.62E+9, 18.24E+9
-18.24E+9,  0.00, 0.00, 0.00, 55.62E+9, 100.
86.98E+9, 3.19E+9,  86.98E+9, 11.25E+9, 11.25E+9, 101.00E+9,  0.00, 0.00
0.00, 42.36E+9, 0.00, 0.00, 0.00, 17.99E+9, 53.55E+9, 17.99E+9
-17.99E+9,  0.00, 0.00, 0.00, 53.55E+9, 200.
86.89E+9, 0.57E+9,  86.89E+9, 9.54E+9,  9.54E+9,  98.02E+9, 0.00, 0.00
0.00, 43.66E+9, 0.00, 0.00, 0.00, 17.90E+9, 51.78E+9, 17.90E+9
-17.90E+9,  0.00, 0.00, 0.00, 51.78E+9, 300.
85.02E+9, -4.90E+9, 85.02E+9, 7.11E+9,  7.11E+9,  93.83E+9, 0.00, 0.00
0.00, 44.99E+9, 0.00, 0.00, 0.00, 18.17E+9, 49.09E+9, 18.17E+9
-18.17E+9,  0.00, 0.00, 0.00, 49.09E+9, 400.
83.69E+9, -8.35E+9, 83.69E+9, 4.98E+9,  4.98E+9,  91.13E+9, 0.00, 0.00
0.00, 45.73E+9, 0.00, 0.00, 0.00, 17.37E+9, 47.74E+9, 17.37E+9
-17.37E+9,  0.00, 0.00, 0.00, 47.74E+9, 450.
81.16E+9, -12.72E+9,  81.16E+9, 1.74E+9,  1.74E+9,  87.97E+9, 0.00, 0.00
0.00, 46.96E+9, 0.00, 0.00, 0.00, 16.58E+9, 45.97E+9, 16.58E+9
-16.58E+9,  0.00, 0.00, 0.00, 45.97E+9, 500.
78.98E+9, -15.40E+9,  78.98E+9, 0.46E+9,  0.46E+9,  86.43E+9, 0.00, 0.00
```

```
0.00, 47.52E+9, 0.00, 0.00, 0.00, 15.88E+9, 45.17E+9, 15.88E+9
−15.88E+9,  0.00, 0.00, 0.00, 45.17E+9, 520.
76.67E+9, −18.43E+9,  76.67E+9, −1.86E+9, −1.86E+9, 83.70E+9, 0.00, 0.00
0.00, 48.11E+9, 0.00, 0.00, 0.00, 15.18E+9, 44.47E+9, 15.18E+9
−15.18E+9,  0.00, 0.00, 0.00, 44.47E+9, 540.
72.77E+9, −23.86E+9,  72.77E+9, −5.64E+9, −5.64E+9, 79.13E+9, 0.00, 0.00
0.00, 48.85E+9, 0.00, 0.00, 0.00, 12.78E+9, 42.22E+9, 12.78E+9
−12.78E+9,  0.00, 0.00, 0.00, 42.22E+9, 560.
66.18E+9, −32.32E+9,  66.18E+9, −5.65E+9, −5.65E+9, 75.16E+9, 0.00, 0.00
0.00, 49.87E+9, 0.00, 0.00, 0.00, 9.77E+9,  38.68E+9, 9.77E+9
−9.77E+9, 0.00, 0.00, 0.00, 38.68E+9, 572.
65.39E+9, −29.73E+9,  65.39E+9, −2.45E+9, −2.45E+9, 74.80E+9, 0.00, 0.00
0.00, 50.05E+9, 0.00, 0.00, 0.00, 0.00, 38.09E+9, 0.00
0.00, 0.00, 0.00, 0.00, 38.09E+9, 573.
64.60E+9, −24.61E+9,  64.60E+9, 0.74E+9,  0.74E+9,  76.80E+9, 0.00, 0.00
0.00, 50.30E+9, 0.00, 0.00, 0.00, 0.00, 37.66E+9, 0.00
0.00, 0.00, 0.00, 0.00, 37.66E+9, 574.
96.41E+9, −1.04E+9, 96.41E+9, 18.52E+9, 18.52E+9, 91.30E+9, 0.00, 0.00
0.00, 50.67E+9, 0.00, 0.00, 0.00, 0.00, 36.44E+9, 0.00
0.00, 0.00, 0.00, 0.00, 36.44E+9, 580.
118.96E+9,  16.99E+9, 118.96E+9,  33.87E+9, 33.87E+9, 110.78E+9,  0.00, 0.00
0.00, 50.65E+9, 0.00, 0.00, 0.00, 0.00, 36.17E+9, 0.00
0.00, 0.00, 0.00, 0.00, 36.17E+9, 600.
128.14E+9,  26.04E+9, 128.14E+9,  42.78E+9, 42.78E+9, 120.43E+9,  0.00, 0.00
0.00, 51.12E+9, 0.00, 0.00, 0.00, 0.00, 36.45E+9, 0.00
0.00, 0.00, 0.00, 0.00, 36.45E+9, 650.
131.52E+9,  29.03E+9, 131.52E+9,  45.82E+9, 45.82E+9, 122.89E+9,  0.00, 0.00
0.00, 51.58E+9, 0.00, 0.00, 0.00, 0.00, 36.72E+9, 0.00
0.00, 0.00, 0.00, 0.00, 36.72E+9, 700.
136.41E+9,  32.11E+9, 136.41E+9,  48.78E+9, 48.78E+9, 125.88E+9,  0.00, 0.00
0.00, 52.16E+9, 0.00, 0.00, 0.00, 0.00, 38.21E+9, 0.00
0.00, 0.00, 0.00, 0.00, 38.21E+9, 800.
*Specific Heat
786.1, 48.8
836.5, 85.1
918.8, 150.6
982.5, 210.6
1030.2, 267.7
1093.8, 345.7
1136.3, 405.6
1189.3, 469.7
1226.4, 507.3
1263.6, 531.
1324.7, 550.7
1468.2, 567.7
1138.6, 573.8
1141.2, 623.9
1149.0, 671.2
1162.1, 776.9
1180.5, 928.5
1182.9, 1027.2
*EXPANSION, TYPE=ORTHO, ZERO=25
1.267E−05,  1.267E−05,  9.557E−06,  25.
1.267E−05,  1.267E−05,  9.557E−06,  35.
1.412E−05,  1.412E−05,  8.522E−06,  100.
1.521E−05,  1.521E−05,  9.143E−06,  150.
1.536E−05,  1.536E−05,  9.604E−06,  200.
1.618E−05,  1.618E−05,  1.029E−05,  250.
1.677E−05,  1.677E−05,  1.069E−05,  300.
1.711E−05,  1.711E−05,  1.106E−05,  350.
1.837E−05,  1.837E−05,  1.180E−05,  400.
1.941E−05,  1.941E−05,  1.264E−05,  450.
2.146E−05,  2.146E−05,  1.339E−05,  500.
2.293E−05,  2.293E−05,  1.458E−05,  543.
2.353E−05,  2.353E−05,  1.500E−05,  550.
2.451E−05,  2.451E−05,  1.556E−05,  558.
2.607E−05,  2.607E−05,  1.640E−05,  567.
2.933E−05,  2.933E−05,  1.796E−05,  574.
2.980E−05,  2.980E−05,  1.839E−05,  576.
2.993E−05,  2.993E−05,  1.864E−05,  579.
2.912E−05,  2.912E−05,  1.803E−05,  600.
2.670E−05,  2.670E−05,  1.654E−05,  650.
```

Appendix C

Scripts and input files for 3D inhomogeneous three component models

Global *Python* script for a parameter study

Appendix3/3D_poly_model_v3_2_3comp.py

```
########################################################################
# This Python script automatically calculate temperature and stress fields for the
# 3D three component parameter study
# requires: electromagnetic field, absorbed power density at integration points, phase elements
########################################################################
#
#load packages
#
import sys                              # better than:     import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
from datetime import datetime
os.system('python -V')
start_time = datetime.now()
print('Starting time = '+str(start_time))
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()                # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic temperaturefield calculation.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',    type=str, required=True, help='Name of the model.')
parser.add_argument('--initialBF',  type=float, required=True, help='Initial constant body heat multiplier.')
parser.add_argument('--totaltime',  type=float, required=True, help='Time of microwave heating.')
parser.add_argument('--abaquspath', type=str, required=True, help='Name of the abaqus executable.')
parser.add_argument('--cpus',       type=int, required=True, help='Number of cpus which shall be used for the Abaqus
      calculation.')
parser.add_argument('--provpower', type=float, required=True, help='Provided microwave power.')
parser.add_argument('--xd', type=float, required=True, help='Dimension in x direction of numerical domain')
parser.add_argument('--yd', type=float, required=True, help='Dimension in y direction of numerical domain')
parser.add_argument('--zd', type=float, required=True, help='Dimension in z direction of numerical domain')
parser.add_argument('--tess',    type=str, required=True, help='Name of the tess file containing the grains.')
parser.add_argument('--phfraction_quartz',    type=float, required=True, help='Phase fraction of the quartz phase.')
parser.add_argument('--phfraction_plagioclase',    type=float, required=True, help='Phase fraction of the plagioclase
      phase.')
parser.add_argument('--amountpoly',    type=int, required=True, help='Amount of polyhedra in the model.')
parser.add_argument('--inpintegrationpoints', type=str, required=True, help='Name of the file containing the coordinates
     of the intergrationpoints.')
parser.add_argument('--inpelementset', type=str, required=True, help='Name of the file containing the elementsets.')
parser.add_argument('--resolution', type=int, required=True, help='Spatial resolution of the FDTD grid')
args = parser.parse_args(namespace=parameter)
#Variable initialization
fine=0
count=0
finished=0
currentBF=0.
######################################################################
#main
#
print('Start of automatic 3D polyhedron calculation\n')
path_working_dir = os.path.dirname(os.path.abspath(__file__))      # The directory of the current __file__ is cut down
     to the directory without the filename.
print('The current working direcotry is: '+str(path_working_dir))
currentBF=parameter.initialBF     #initialize current body flux
provenergy=0.7*parameter.provpower*parameter.totaltime/4. #calculate provided energy (with 30 percent of loss)
print('Provided Enerergy= '+str(provenergy))
while (finished!=1):
#copy files
  os.system("cp output3D_granite_hartlieb_inhom_v1_2_E2.h5 output3D_"+str(parameter.model)+"_E2.h5")
  os.system("cp absorption_101_granite_hartlieb_inhom_v1_2_03_02_1000_intpoints.txt absorption_101_"+str(parameter.model)
     +".txt")
  os.system("cp quartz_elements_granite_hartlieb_inhom_v1_2.inp quartz_elements_"+str(parameter.model)+".inp")
  os.system("cp plagioclase_elements_granite_hartlieb_inhom_v1_2.inp plagioclase_elements_"+str(parameter.model)+".inp")
  os.system("cp mica_elements_granite_hartlieb_inhom_v1_2.inp mica_elements_"+str(parameter.model)+".inp")
#Create Abaqus input file
  if fine==0:
    if 0 != os.system(str(parameter.abaquspath)+' python -u create_heat_input_3comp_v2.py --model '+str(parameter.model)+
     ' --totaltime '+str(parameter.totaltime)+' | tee create_heat_input_3_comp_v2'+str(parameter.model)+'_'+str(count)+'
     .log'):
        sys.exit('Error during create_heat_input_v2.py.')
      fine=1
    else:
      print ('Heat abaqus input file written!')
#Create DFLUX subroutine
  if fine==0:
    print('Start the creation of the DFLUX usersubroutine with a BF of '+str(parameter.initialBF)+'\n')
    if 0 != os.system(str(parameter.abaquspath)+' python -u create_BF_subroutine.py --model '+str(parameter.model)+' --
     initialBF '+str(currentBF)+' --nameBF BF_intpoint_server_'+str(parameter.model)+'_'+str(count)+'.f --work '+
     path_working_dir):
          sys.exit('Error during create_BF_subroutine.py.')
      fine=1
    else:
      print('Heat input file generated')
# Start first Abaqus heating job
  if fine==0:
    print('Start first NT11 heat calculation \n')
    if 0 != os.system(str(parameter.abaquspath)+' job=j101_01_"+str(parameter.model)+"_03_02_1000_poly_intpoint_heat inp=
     j101_01_"+str(parameter.model)+"_03_02_1000_poly_intpoint_heat.inp"+" user=BF_intpoint_server_"+str(parameter.model
     )+'_'+str(count)+".f cpus=1 interactive | tee j101_01_"+str(parameter.model)+"03_02_1000_poly_intpoint_1.log"):
      sys.exit("Error during first NT11 calculation")
      fine=1
      print("NT11 Abaqus job finished and start of heat calculation. \n")
  if fine==0:
    if 0 != os.system(str(parameter.abaquspath)+' python -u script_calc_heat_aut_3comp.py --model '+str(parameter.model)+
     ' --heatout energy_j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat_'+str(count)):
          sys.exit('Error during heat calculation.')
      fine=1
#read in total energy after first loop
  if fine==0:
    print('Compare energies')
    f=open('energy_j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_heat_'+str(count)+'.txt', 'r')
    for line in f:
      line1=line.rstrip()
      if 'Total' in line1:
        temp=line1.split(': ')
        curenergy=float(temp[1])
    f.close()
#check if error in energy is less than 3 percent
  if abs((curenergy-provenergy)/provenergy)>0.03 and fine==0:
    print('Energydifference= '+str(((curenergy-provenergy)/provenergy)*100.)+'% after '+str(count+1)+'. thermal
     calculation is bigger than 3% --> recalculation')
  #calculate new BF
    currentBF=currentBF*(provenergy/curenergy)
    print('New BF= '+str(currentBF))
    count=count+1
  else:
    print('Temperaturefield calculation of heating step finished with an error of '+str((curenergy-provenergy)/provenergy
      ))
    finished=1
#Start Abaqus cooling job
if fine==0:
  os.system('cp j101_01_cooling.inp j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling.inp')
  if 0 != os.system(str(parameter.abaquspath)+' job=j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling inp
     =j101_01_'+str(parameter.model)+'_03_02_1000_poly_intpoint_cooling.inp oldjob=j101_01_'+str(parameter.model)+'
     _03_02_1000_poly_intpoint_heat user=subroutine_cooling.f cpus='+str(parameter.cpus)+' interactive | tee j101_01_'+
     str(parameter.model)+'_03_02_1000_poly_intpoint_cooling.log'):
    sys.exit("Error during cooling NT11 calculation")
    fine=1
  else:
    print("Temperaturefield calculation of cooling step finished. \n")
#Create stress Abaqus input file
if fine==0:
  if 0 != os.system(parameter.abaquspath+' python -u create_stress_input.py --model '+str(parameter.model)+' --totaltime
     '+str(parameter.totaltime)):
          sys.exit('Error during create_stress_input.py.')
    fine=1
  else:
    print("Stress input file successfully created.")
#Start stress calculation
if fine==0:
  print('Start of stress calculation')
  if 0 != os.system(str(parameter.abaquspath)+' job=j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint inp=
     j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint.inp cpus='+str(parameter.cpus)+' interactive | tee
     j101_03_'+str(parameter.model)+'_03_02_1000_poly_intpoint.log'):
    sys.exit("Error during stress calculation")
    fine=1
  else:
    print("Stress calculation finished.")
exit()
```

Abaqus CDP model

FORTRAN subroutine to adapt dilation angle

Appendix3/Granite_arzua2013_v2.f

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       Subroutine USDFLD to update field variables
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE USDFLD(FIELD,STATEV,PNEWDT,DIRECT,T,CELENT,
     1 TIME,DTIME,CMNAME,ORNAME,NFIELD,NSTATV,NOEL,NPT,LAYER,
     2 KSPT,KSTEP,KINC,NDI,NSHR,COORD,JMAC,JMATYP,MATLAYO,LACCFLA)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME,ORNAME
      CHARACTER*3  FLGRAY(15)
      DIMENSION FIELD(NFIELD),STATEV(NSTATV),DIRECT(3,3),
     1 T(3,3),TIME(2)
      DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)
      DOUBLE PRECISION EP1, EP3, GP
      DOUBLE PRECISION a, b, c
C     EP1   maximum principal plastic strain
C     EP2   minimum principal plastic strain
```

```
C      GP      plastic shear strain in percent
C      a,b,c coefficients of the dilation angle function
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      Initialize variables
C
       EP1=0.0
       EP3=0.0
       GP=0.0
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C Define coefficients of dilation function
C
       a=30.95
       b=8.97
       c=0.654
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C get principal plastic strain vector
C
       CALL GETVRM('PEP',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,
      1 MATLAYO,LACCFLA)
       EP1 = ARRAY(1)
       EP3 = ARRAY(3)
C
C calculate plastic shear strain
C
       GP = ABS((EP1-EP3)*100.)
C
C calculate the dilation angle and use it as a field variable
C
       IF(GP.GT.0.314878981)THEN
        FIELD(1) = ((a*b*(EXP(-b*GP)-EXP(-c*GP)))/(c-b))
       ELSE
        FIELD(1) = 25.189875
       END IF
       IF (NOEL.EQ.1000)THEN
        WRITE(6,*) 'DILATION ANGLE at INC ',KINC,'is ',FIELD(1),
      1      ' and GP of ',GP, ' EP1: ',EP1,' EP3:',EP3
       END IF
C
C update the expansion behaviour according to the current step
C
       IF(KSTEP.EQ.4)THEN
        FIELD(2)=3
       ELSE
        FIELD(2)=1
       END IF
C
C If error, write comment to .log file:
C
       IF(JRCD.NE.0)THEN
        WRITE(6,*) 'REQUEST ERROR IN USDFLD FOR ELEMENT NUMBER ',
      1      NOEL,'INTEGRATION POINT NUMBER ',NPT
       END IF
C
       RETURN
       END
```

Quartz CDP model

Appendix3/quartz_246_conc.inp

```
*Material , name=quartz
*Conductivity
8.399,  25.
7.613,  50.
6.467,  100.
5.698,  150.
5.139,  200.
4.679,  250.
4.314,  300.
4.015,  350.
3.783,  400.
3.557,  450.
3.387,  500.
3.316,  520.
3.237,  540.
3.167,  560.
3.250,  572.
3.296,  573.
3.343,  574.
3.453,  580.
3.483,  600.
3.537,  650.
3.602,  700.
3.676,  750.
3.731,  800.
*Density
 2649.,
*Elastic
9.59E+10, 0.09, 25.
9.48E+10, 0.08, 100.
9.32E+10, 0.07, 200.
9.18E+10, 0.06, 300.
8.80E+10, 0.03, 400.
8.58E+10, 0.01, 450.
8.22E+10, −0.04,  500.
7.99E+10, −0.06,  520.
7.65E+10, −0.10,  540.
6.92E+10, −0.18,  560.
5.89E+10, −0.27,  572.
6.28E+10, −0.23,  573.
6.75E+10, −0.16,  574.
9.13E+10, 0.11, 580.
1.01E+11, 0.21, 600.
1.04E+11, 0.24, 650.
1.05E+11, 0.25, 700.
1.08E+11, 0.25, 800.
*Specific Heat
786.1, 48.8
836.5, 85.1
918.8, 150.6
982.5, 210.6
1030.2, 267.7
1093.8, 345.7
1136.3, 405.6
1189.3, 469.7
1226.4, 507.3
1263.6, 531.
1324.7, 550.7
1468.2, 567.7
1138.6, 573.8
1141.2, 623.9
1149.0, 671.2
1162.1, 776.9
1180.5, 928.5
1182.9, 1027.2
*EXPANSION, TYPE=ISO, ZERO=20., DEPENDENCIES=2
1.0134E−05, 20.00, ,1
1.0134E−05, 36.95, ,1
1.2531E−05, 105.40, ,1
```

```
1.2951E−05, 182.40, ,1
1.3782E−05, 250.86, ,1
1.4667E−05, 304.37, ,1
1.5260E−05, 372.85, ,1
1.6324E−05, 443.51, ,1
1.7813E−05, 486.39, ,1
1.9599E−05, 535.74, ,1
2.0050E−05, 543.00, ,1
2.0585E−05, 550.00, ,1
2.1421E−05, 558.00, ,1
2.2734E−05, 567.00, ,1
2.5400E−05, 574.00, ,1
2.5856E−05, 576.00, ,1
2.6021E−05, 579.00, ,1
2.5924E−05, 581.08, ,1
2.5726E−05, 585.41, ,1
2.4088E−05, 623.85, ,1
2.1352E−05, 700., ,1
1.8614E−05, 800., ,1
**adapted
2.0489E−05,   830., ,1
2.7065E−05,   875., ,1
2.8840E−05, 900.00, ,1
3.0000E−05,   1000., ,1
7.1699E−04, 20.00, ,2
8.8807E−04, 36.95, ,2
1.8678E−04, 105.40, ,2
1.0458E−04, 182.40, ,2
7.8241E−05, 250.86, ,2
6.6997E−05, 304.37, ,2
5.7434E−05, 372.85, ,2
5.1461E−05, 443.51, ,2
4.9720E−05, 486.39, ,2
4.8453E−05, 535.74, ,2
4.8503E−05, 543.00, ,2
4.8663E−05, 550.00, ,2
4.9081E−05, 558.00, ,2
4.9939E−05, 567.00, ,2
5.2261E−05, 574.00, ,2
5.2621E−05, 576.00, ,2
5.2642E−05, 579.00, ,2
5.2446E−05, 581.08, ,2
5.2045E−05, 585.41, ,2
4.8732E−05, 623.85, ,2
4.3236E−05, 700.00, ,2
3.7692E−05, 800.00, ,2
3.6296E−05, 830.00, ,2
3.4386E−05, 875.00, ,2
3.3409E−05, 900.00, ,2
3.0000E−05, 1000.00, ,2
1.0134E−05, 20.00, ,3
1.0134E−05, 36.95, ,3
1.2531E−05, 105.40, ,3
1.2951E−05, 182.40, ,3
1.3782E−05, 250.86, ,3
1.4667E−05, 304.37, ,3
1.5260E−05, 372.85, ,3
1.6324E−05, 443.51, ,3
1.7813E−05, 486.39, ,3
1.9599E−05, 535.74, ,3
2.0050E−05, 543.00, ,3
2.0585E−05, 550.00, ,3
2.1421E−05, 558.00, ,3
2.2734E−05, 567.00, ,3
2.5400E−05, 574.00, ,3
2.5856E−05, 576.00, ,3
2.6021E−05, 579.00, ,3
2.5924E−05, 581.08, ,3
2.5725E−05, 585.41, ,3
2.4088E−05, 623.85, ,3
2.1390E−05, 700.00, ,3
1.8648E−05, 800.00, ,3
1.7957E−05, 830.00, ,3
1.7012E−05, 875.00, ,3
1.6529E−05, 900.00, ,3
```

```
1.4842E-05, 1000.00, ,3
1.0134E-05, 20.00, ,4
1.0134E-05, 36.95, ,4
1.2531E-05, 105.40, ,4
1.2951E-05, 182.40, ,4
1.3782E-05, 250.86, ,4
1.4667E-05, 304.37, ,4
1.5260E-05, 372.85, ,4
1.6324E-05, 443.51, ,4
1.7813E-05, 486.39, ,4
1.9599E-05, 535.74, ,4
2.0050E-05, 543.00, ,4
2.0585E-05, 550.00, ,4
2.1421E-05, 558.00, ,4
2.2734E-05, 567.00, ,4
2.5400E-05, 574.00, ,4
2.5856E-05, 576.00, ,4
2.6021E-05, 579.00, ,4
2.5924E-05, 581.08, ,4
2.5726E-05, 585.41, ,4
2.4088E-05, 623.85, ,4
2.1352E-05, 700.00, ,4
*Concrete damaged plasticity , DEPENDENCIES=2
0.01, 0.15, 2.34, 0.56, 0.0001, , 0.01,
50., 0.15, 2.34, 0.56, 0.0001, , 50.0,
*Concrete tension stiffening , TYPE=GFI
 25.e6, 52.68
*Concrete compression hardening
 85.45e6, 0.
 20.71e6, 0.003281298
 20.71e6, 0.1
*Concrete compression damage
 0.0, 0.0
 0.757636, 0.003281298
 0.757636, 0.1
*CONCRETE TENSION DAMAGE, TYPE=DISPLACEMENT, COMPRESSION RECOVERY=0.
 0.0, 0.0
 0.57, 4.214e-6
*USER DEFINED FIELD
**DEPVAR
**1
```

Appendix D

Scripts and input files for 3D homogeneous SCM

Global *Python* script

Appendix4/3D_hom_model_coupled_v2_1.py

```
#######################################################################
# This Python script calculates the SCM for homogeneous rocks
#######################################################################
#
#load packages
import sys                          # better than:    import os.sys
import os                           # os = operating system
import argparse                     # to parse arguments
from datetime import datetime
os.system('python -V')
start_time = datetime.now()
print('Starting time = '+str(start_time))
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()            # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic temperaturefield calculation.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',    type=str, required=True, help='Name of the model.')
parser.add_argument('--initialBF',  type=float, required=True, help='Initial constant body heat multiplier.')
parser.add_argument('--abaquspath', type=str, required=True, help='Name of the abaqus executable.')
parser.add_argument('--cpus',      type=int, required=True, help='Number of cpus which shall be used for the Abaqus
     calculation.')
parser.add_argument('--provpower', type=float, required=True, help='Provided microwave power.')
parser.add_argument('--xd', type=float, required=True, help='Dimension in x direction of numerical domain')
parser.add_argument('--yd', type=float, required=True, help='Dimension in y direction of numerical domain')
parser.add_argument('--yf', type=float, required=True, help='Dimension in y direction of refined mesh')
parser.add_argument('--zd', type=float, required=True, help='Dimension in z direction of numerical domain')
parser.add_argument('--avgeps', type=str, required=True, help='Name of the file containing the averaged relative
     dielectric constant')
parser.add_argument('--resolution', type=float, required=True, help='Spatial resolution of the FDTD grid')
parser.add_argument('--meshconstref', type=float, required=True, help='Edge length of the refined mesh')
parser.add_argument('--meshconstcoarse', type=float, required=True, help='Edge length of the coarse mesh')
parser.add_argument('--maxdeltatemp',    type=float, required=True, help='Maximum amount of temperature change during one
      increment.')
parser.add_argument('--timeheating',    type=float, required=True, help='Time to heat sample.')
parser.add_argument('--material', type=str, required=True, help='Name of the material.')
parser.add_argument('--materialfile', type=str, required=True, help='Name of the material file.')
args = parser.parse_args(namespace=parameter)
########################################################################
#Variable initialization
fine=0
count=0
finished=0
currentBF=0.
epsreal=0.
epsim=0.
initialtemp=25.
currentBF=0.
currentmaxE=0.
a=[]
b=[]
y=0
z=0
floatq=[]
floatp=[]
eps=[]
oldtime=0.    #old time
########################################################################
#main
print('Start of coupled 3D homogeneous model calculation\n')
path_working_dir = os.path.dirname(os.path.abspath(__file__))      # The directory of the current __file__ is cut down
     to the directory without the filename.
print('The current working direcotry is: '+str(path_working_dir))
#Start global loop
currentBF=parameter.initialBF     #initialize current body flux
provenergy=0.7*parameter.provpower*parameter.timeheating/4. #calculate provided energy (with 30 percent of loss)
print('Provided Energy= '+str(provenergy))
#Create abaqus input file
if 0 != os.system(str(parameter.abaquspath)+' python -u create_mesh_hom_v7.py --model '+str(parameter.model)+' --horc 0
     --xd '+str(parameter.xd)+' --yd '+str(parameter.yd)+' --zd '+str(parameter.zd)+' --yf '+str(parameter.yf)+' --
     meshconstref '+str(parameter.meshconstref)+' --meshconstcoarse '+str(parameter.meshconstcoarse)+' | tee
     create_mesh_hom_v7_'+str(parameter.model)+'.log'):
  sys.exit('Error during create_mesh_hom_v7.py!')
  fine=1
else:
  print ('Heat abaqus input file written!')
#read in eps values
epsread=open(parameter.avgeps, 'r')     #open eps data of homogeneous material
for lineq in epsread:
  a.append(lineq.rstrip())
epsread.close()
while y<len(a):          #split
  temp=[float(x) for x in a[y].split("; ")] #0...temp, 1...real, 2... imaginary part
  eps.append(temp)
  y=y+1
print('second line of eps avg vector is'+str(eps[1][0])+' '+str(eps[1][1])+' '+str(eps[1][2]))
while (finished!=1):
  inc=1
  curtime=0.  #current time
  totalenergy=0
  while curtime<parameter.timeheating:
#start Meep FDTD calculation
    if fine==0:
      if 0 != os.system("source /export/opt/gcc-4.6.4/bin/setvars.sh; source /export/opt/2014-05-12_openmpi-1.8.1-gcc
     -4.6.4/bin/setvars.sh; mpirun ./hom_model_coupled_v7.exe "+str(parameter.xd)+" "+str(parameter.yd)+" "+str(
     parameter.yf)+" "+str(parameter.zd)+" "+str(parameter.avgeps)+" "+str(parameter.resolution)+" "+str(inc)+" "+str(
     parameter.meshconstref)+" "+str(parameter.meshconstcoarse)+" "+str(parameter.model)+"| tee hom_model_coupled_v7"+
     str(parameter.model)+"_"+str(inc)+".log"):
            sys.exit('Error during meep electric field calculation')
        fine=1
      else:
        print('Meep electric field calculated')
#start hdf5 E2 calculation
    if fine==0:
      if 0 != os.system("source /export/opt/gcc-4.6.4/bin/setvars.sh; "+str(path_working_dir)+"/
     hdf5_aut_hom_model_coupled_v7.exe "+str(parameter.model)+" "+str(parameter.xd)+" "+str(parameter.yd)+" "+str(
     parameter.zd)+" "+str(parameter.resolution)+" "+str(inc)+" | tee hdf5_aut_hom_model_coupled_v7"+str(parameter.model
     )+"_"+str(inc)+".log"):
            sys.exit('Error during hdf5_aut_v7.exe calculation!')
        fine=1
      else:
        print ('Hdf5 files successfully calculated!')
        os.system("mv eps-000000000.h5 eps_"+str(parameter.timeheating)+"s_"+str(parameter.provpower)+"W_inc"+str(inc)+".
     h5")
        os.system("rm ex-*; rm ey-*; rm ez-*")
#calculate absorbed power density
    if fine==0:
      if 0 != os.system("source /export/opt/gcc-4.6.4/bin/setvars.sh; "+str(path_working_dir)+"/
     hdf5_int_v7_coupled_hom_model.exe "+str(parameter.model)+" "+str(parameter.xd)+" "+str(parameter.yd)+" "+str(
     parameter.yf)+" "+str(parameter.zd)+" "+str(parameter.avgeps)+" "+str(parameter.resolution)+" "+str(parameter.
     meshconstref)+" "+str(parameter.meshconstcoarse)+" "+str(inc)+" | tee hdf5_int_v7_coupled_hom_model"+str(parameter.
     model)+"_"+str(inc)+".log"):
              sys.exit('Error during hdf5_int_v7_coupled_hom_model.exe calculation!')
        fine=1
      else:
        print ('Hdf5_int_v7_coupled_hom_model files successfully calculated!')
#Create DFLUX subroutine
    if fine==0:
      print('Start the creation of the DFLUX usersubroutine with a BF of '+str(currentBF)+'\n')
      if 0 != os.system(str(parameter.abaquspath)+' python -u create_BF_subroutine_hom_model_coupled_v7.py --model '+str(
     parameter.model)+' --initialBF '+str(currentBF)+' --nameBF BF_intpoint_server_'+str(parameter.model)+'_'+str(inc)+'
     _'+str(count)+'.f --work '+path_working_dir+' --xd '+str(parameter.xd)+' --yd '+str(parameter.yd)+' --yf '+str(
     parameter.yf)+' --zd '+str(parameter.zd)+' --meshconstref '+str(parameter.meshconstref)+' --meshconstcoarse '+str(
     parameter.meshconstcoarse)+' --inc '+str(inc)):
              sys.exit('Error during create_BF_subroutine_hom_model_coupled_v7.py.')
        fine=1
      else:
        print('Heat input file generated')
#Create Abaqus input file
    if fine==0:
      if 0 != os.system(str(parameter.abaquspath)+' python -u create_abaqus_input_file_coupled_v2.py --model '+str(
     parameter.model)+' --horc 0 --inc '+str(inc)+' --maxdeltatemp '+str(parameter.maxdeltatemp)+' --material '+str(
     parameter.material)+' --materialfile '+str(parameter.materialfile)+' --timeheating '+str(parameter.timeheating)+' |
      tee create_heat_input_coupled_v2_1'+str(parameter.model)+'_'+str(inc)+'.log'):
              sys.exit('Error during create_abaqus_input_file_coupled_v2.py.')
        fine=1
      else:
        print ('Heat abaqus input file for inc'+str(inc)+' written!')
# Start first Abaqus heating job
    if (fine==0 and inc==1):
      print('Start NT11 heat calculation \n')
      if 0 != os.system(str(parameter.abaquspath)+" job=j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(inc)+
     " inp=j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(inc)+".inp"+" user=BF_intpoint_server_"+str(
     parameter.model)+'_'+str(inc)+'_'+str(count)+".f cpus=1 interactive | tee j101_01_"+str(parameter.model)+"
     _hom_model_heating_"+str(inc)+"_"+str(count)+".log"):
        sys.exit("Error during "+str(inc)+" NT11 calculation")
        fine=1
      else:
        print("NT11 Abaqus job inc"+str(inc)+" finished and start of heat calculation. \n")
    if (fine==0 and inc!=1):
      print('Start '+str(inc)+' NT11 heat calculation \n')
      if 0 != os.system(str(parameter.abaquspath)+" job=j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(inc)+
     " inp=j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(inc)+".inp"+" user=BF_intpoint_server_"+str(
     parameter.model)+'_'+str(inc)+'_'+str(count)+".f oldjob=j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str
     (inc-1)+" cpus=1 interactive | tee j101_01_"+str(parameter.model)+"_hom_model_heating_"+str(inc)+"_"+str(count)+".
     log"):
        sys.exit("Error during "+str(inc)+" NT11 calculation")
        fine=1
      else:
        print("NT11 Abaqus job inc"+str(inc)+" finished and start of heat calculation. \n")
#
#read odb data and save to external file
    if fine==0:
      if 0 != os.system(str(parameter.abaquspath)+' python -u read_out_odb_hom_model_coupled_v1.py --model '+str(
     parameter.model)+' --inc '+str(inc)+' | tee read_out_odb_hom_model_coupled_'+str(inc)+'_v1.log'):
              sys.exit('Error during read out of odb thermal data')
        fine=1
      else:
        print('Read out of thermal odb data of inc'+str(inc)+' finished')
    #read in current time
    if fine==0:
      t=open('currenttime_'+str(parameter.model)+'_'+str(inc)+'.txt', 'r')
      oldtime=curtime
      curtime=float(t.readline())
      deltatime=curtime-oldtime
      if deltatime==0:
        print ('ERROR delta time equals zero!!!')
        fine=1
      t.close()
      print('Current time= '+str(curtime)+' and delta time= '+str(deltatime))
    #adapt energy
    currentmaxE=(provenergy*deltatime)/parameter.timeheating
    #calculate heat
    if fine==0:
      if 0 != os.system(str(parameter.abaquspath)+' python -u script_calc_heat_coupled_hom_model_v2.py --model '+str(
      parameter.model)+' --heatout energy_j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(inc)+'_'+str(count)
      +' --inc '+str(inc)+' --timeold '+str(oldtime)+' --aquad '+str(parameter.meshconstref**2)+' --material '+str(
      parameter.material)+' | tee script_calc_heat_coupled_hom_model_v2_'+str(inc)+'_'+str(count)+'.log'):
              sys.exit('Error during heat calculation.')
        fine=1
      else:
        print("Energy of inc"+str(inc)+" and loop "+str(count)+" calculated!")
    #read in total energy after first loop
    if fine==0:
      print('Compare energies')
      f=open('energy_j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(inc)+'_'+str(count)+'.txt', 'r')
      for line in f:
        line1=line.rstrip()
        if 'Total' in line1:
          temp=line1.split(': ')
          curenergy=float(temp[1])
      f.close()
    if abs((curenergy-currentmaxE)/currentmaxE)>0.10:
      print("WARNING after "+str(curtime)+"s of coupled heat calculation achieved energy balance is unlikely")
    inc=inc+1
    totalenergy=totalenergy+curenergy
#check if totalenergy is in the range of provided energy
  if abs((totalenergy-provenergy)/provenergy)>0.05 and fine==0:
    print('Totalenergy= '+str(totalenergy)+' which results in a energydifference= '+str(((totalenergy-provenergy)/
      provenergy)*100.)+'% in j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(inc)+'.odb after '+str(count)+'
      . loop is bigger than 5% --> recalculation')
    #calculate new BF
    currentBF=currentBF*(provenergy/totalenergy)
    print('New BF= '+str(currentBF))
    count=count+1
  else:
    print('Temperaturefield calculation of heating step finished with an error of '+str((totalenergy-provenergy)/
      provenergy))
    finished=1
#Create Abaqus cooling job
if fine==0:
  if 0 != os.system(str(parameter.abaquspath)+' python -u create_abaqus_input_file_coupled_v2.py --model '+str(parameter.
      model)+' --horc 1 --inc 1 --maxdeltatemp '+str(parameter.maxdeltatemp)+' --material '+str(parameter.material)+' --
      materialfile '+str(parameter.materialfile)+' --timeheating '+str(parameter.timeheating)+' | tee
      create_heat_input_coupled_v2_1'+str(parameter.model)+'_'+str(inc)+'.log'):
    sys.exit('Error during create_abaqus_input_file_coupled_v2.py.')
    fine=1
  else:
    print ('Cooling abaqus input file written!')
#Start Abaqus cooling job
if fine==0:
  os.system('cp j101_01_cooling_hom_model.inp j101_01_'+str(parameter.model)+'_hom_model_cooling.inp')
  if 0 != os.system(str(parameter.abaquspath)+' job=j101_01_'+str(parameter.model)+'_hom_model_cooling inp=j101_01_'+str(
      parameter.model)+'_hom_model_cooling.inp oldjob=j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(inc-1)+
      ' cpus='+str(parameter.cpus)+' interactive user=BF_cooling.f | tee j101_01_'+str(parameter.model)+'
      _hom_model_cooling.log'):
    sys.exit("Error during cooling NT11 calculation")
    fine=1
  else:
    print("Temperaturefield calculation of cooling step finished. \n")
#Create stress Abaqus input file
if fine==0:
  if 0 != os.system(str(parameter.abaquspath)+' python -u create_mesh_hom_v1.py --model '+str(parameter.model)+' --horc 1
       --xd '+str(parameter.xd)+' --yd '+str(parameter.yd)+' --zd '+str(parameter.zd)+' --yf '+str(parameter.yf)+' --
      meshconstref '+str(parameter.meshconstref)+' --meshconstcoarse '+str(parameter.meshconstcoarse)+' | tee
      create_mesh_hom_v1'+str(parameter.model)+'.log'):
    sys.exit('Error during create_mesh_hom_v1.py!')
    fine=1
  else:
    print ('Heat abaqus input file written!')
```

```
#Create Abaqus cooling job
if fine==0:
  if 0 != os.system(str(parameter.abaquspath)+' python -u create_stress_file_coupled_v5_1.py --model '+str(parameter.
      model)+' --inc '+str(inc)+' --material '+str(parameter.material)+' --materialfile '+str(parameter.materialfile)+' |
       tee create_stress_file_coupled_v5_1.log'):
    sys.exit('Error during create_stress_file_coupled_v5_1')
    fine=1
  else:
    print ('Cooling abaqus input file written!')
    fine=1
#Start stress calculation
if fine==0:
  print('Start of stress calculation')
  if 0 != os.system(str(parameter.abaquspath)+' job=j101_03_'+str(parameter.model)+'_hom_model inp=j101_03_'+str(
      parameter.model)+'_hom_model.inp cpus='+str(parameter.cpus)+' interactive | tee j101_03_'+str(parameter.model)+'
      _hom_model.log'):
    sys.exit("Error during stress calculation")
    fine=1
  else:
    print("Stress calculation finished.")
exit()
```

Python script to generate geometry

Appendix4/create_mesh_hom_v7.py

```
######################################################################
# Python script which automatically creates the mesh for the Abaqus homogeneous model
######################################################################
#
import sys                              # better than:     import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()                # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created Dflux file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',    type=str, required=True, help='Name of the model.')
parser.add_argument('--horc',  type=int, required=True, help='0...create heat model, 1...create stress model.')
parser.add_argument('--xd', type=float, required=True, help='Dimension in x direction of numerical domain')
parser.add_argument('--yd', type=float, required=True, help='Dimension in y direction of numerical domain')
parser.add_argument('--zd', type=float, required=True, help='Dimension in z direction of numerical domain')
parser.add_argument('--yf', type=float, required=True, help='Dimension in y direction of refined mesh')
parser.add_argument('--meshconstref', type=float, required=True, help='Edge length of the refined mesh')
parser.add_argument('--meshconstcoarse', type=float, required=True, help='Edge length of the coarse mesh')
args = parser.parse_args(namespace=parameter)
# Variables to be defined
pmlt=0.1       #thickness of the pml layer
airt=0.1       #thickness of the air layer
numlim=1.E-8     #numerical limit
######################################################################
# Open and write in DFLUX file
if (parameter.horc==0):
  df=open('model_information_'+str(parameter.model)+'_heating.inp', 'w')
  df.write('** Automatic created abaqus model for heat transfer calculation\n')
else:
  if (parameter.horc==1):
    df=open('model_information_'+str(parameter.nameBF)+'_static.inp', 'w')
    df.write('** Automatic created abaqus model for stress calculation\n')
  else:
    print ("ERROR wrong horc integer value!!!")
elementx=int((parameter.xd-2.*pmlt)/(2*parameter.meshconstref)+numlim)
elementz=int((parameter.zd-2.*pmlt)/(2*parameter.meshconstref)+numlim)
elementy=int(parameter.yf/parameter.meshconstref+numlim)
```

```
elementycoarse=int((parameter.yd-parameter.yf-2.*pmlt-airt)/parameter.meshconstcoarse+numlim)
print('elementx= '+str(elementx)+' elementy= '+str(elementy)+' elementz= '+str(elementz)+' elementycoarse= '+str(
      elementycoarse))
df.write('*Node\n')
df.write('1, 0., 0., 0.\n')
df.write(str(elementx+1)+', '+str((parameter.xd-2.*pmlt)/2.)+', 0., 0.\n')
df.write(str(((elementx+1)*elementz)+1)+', 0., 0., '+str((parameter.zd-2.*pmlt)/2.)+'\n')
df.write(str((elementx+1)*(elementz+1))+', '+str((parameter.xd-2.*pmlt)/2.)+', 0., '+str((parameter.zd-2.*pmlt)/2.)+'\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy))+1)+', 0., '+str(parameter.yf)+', 0.\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy))+elementx+1)+', '+str((parameter.xd-2.*pmlt)/2.)+', '+str(parameter.yf
      )+', 0.\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy))+((elementx+1)*elementz+1))+', 0., '+str(parameter.yf)+', '+str((
      parameter.zd-2.*pmlt)/2.)+'\n')
df.write(str((elementx+1)*(elementz+1)*(elementy+1))+', '+str((parameter.xd-2.*pmlt)/2.)+', '+str(parameter.yf)+', '+str
      ((parameter.zd-2.*pmlt)/2.)+'\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy+elementycoarse))+1)+', 0., '+str(parameter.yd-2.*pmlt-airt)+', 0.\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy+elementycoarse))+elementx+1)+', '+str((parameter.xd-2.*pmlt)/2.)+', '+
      str(parameter.yd-2.*pmlt-airt)+', 0.\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy+elementycoarse))+(elementx+1)*(elementz)+1)+', 0., '+str(parameter.yd
      -2.*pmlt-airt)+',' +str((parameter.zd-2.*pmlt)/2.)+'\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy+elementycoarse+1)))+', '+str((parameter.xd-2.*pmlt)/2.)+', '+str(
      parameter.yd-2.*pmlt-airt)+', '+str((parameter.zd-2.*pmlt)/2.)+'\n')
df.write('*Ngen, Nset=y0bot\n')
df.write('1, '+str(elementx+1)+', 1\n')
df.write('*Ngen, Nset=y0top\n')
df.write(str((elementx+1)*elementz+1)+', '+str((elementx+1)*(elementz+1))+', 1\n')
df.write('*Ngen, Nset=yrefbot\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy))+1)+', '+str(((elementx+1)*(elementz+1)*(elementy))+elementx+1)+', 1\n
      ')
df.write('*Ngen, Nset=yreftop\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy))+((elementx+1)*elementz+1))+', '+str((elementx+1)*(elementz+1)*(
      elementy+1))+', 1\n')
df.write('*Nfill, Nset=y0\n')
df.write('y0bot, y0top, '+str(elementx)+', '+str(elementx+1)+'\n')
df.write('*Nfill, Nset=yref\n')
df.write('yrefbot, yreftop, '+str(elementx)+', '+str(elementx+1)+'\n')
df.write('*Nfill, Nset=nodes_ref\n')
df.write('y0, yref, '+str(elementy)+', '+str((elementx+1)*(elementz+1))+'\n')
df.write('*Ngen, Nset=y1bot\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy+elementycoarse))+1)+', '+str(((elementx+1)*(elementz+1)*(elementy+
      elementycoarse))+elementx+1)+', 1\n')
df.write('*Ngen, Nset=y1top\n')
df.write(str(((elementx+1)*(elementz+1)*(elementy+elementycoarse))+(elementx+1)*(elementz)+1)+', '+str(((elementx+1)*(
      elementz+1)*(elementy+elementycoarse+1)))+', 1\n')
df.write('*Nfill, Nset=y1\n')
df.write('y1bot, y1top, '+str(elementx)+', '+str(elementx+1)+'\n')
df.write('*Nfill, Nset=nodes_coarse\n')
df.write('yref, y1, '+str(elementycoarse)+', '+str((elementx+1)*(elementz+1))+'\n')
if (parameter.horc==0):
  df.write('*Element, Type=DC3D8\n')
else:
  df.write('*Element, Type=C3D8R\n')
df.write('1, 1, 2, '+str((elementx+1)*(elementz+1)+2)+', '+str((elementx+1)*(elementz+1)+1)+', '+str(elementx+2)+', '+str
      (elementx+3)+', '+str((elementx+1)*(elementz+1)+elementx+3)+', '+str((elementx+1)*(elementz+1)+elementx+2)+'\n')
df.write('*Elgen, Elset=refined_elements\n')
df.write('1, '+str(elementx)+', 1, 1, '+str(elementz)+', '+str(elementx+1)+', '+str(elementx)+', '+str(elementy)+', '+str
      ((elementx+1)*(elementz+1))+', '+str(elementx*elementz)+'\n')
if (parameter.horc==0):
  df.write('*Element, Type=DC3D8\n')
else:
  df.write('*Element, Type=C3D8R\n')
df.write(str(elementx*elementz*elementy+1)+', '+str(((elementx+1)*(elementz+1)*(elementy))+1)+', '+str(((elementx+1)*(
      elementz+1)*(elementy))+2)+', '+str((elementx+1)*(elementz+1)*(elementy+1)+2)+', '+str((elementx+1)*(elementz+1)*(
      elementy+1)+1)+', '+str(((elementx+1)*(elementz+1)*(elementy))+2+elementx)+', '+str(((elementx+1)*(elementz+1)*(
      elementy))+3+elementx)+', '+str((elementx+1)*(elementz+1)*(elementy+1)+elementx+3)+', '+str((elementx+1)*(elementz
      +1)*(elementy+1)+elementx+2)+'\n')
df.write('*Elgen, Elset=coarse_elements\n')
df.write(str(elementx*elementz*elementy+1)+', '+str(elementx)+', 1, 1, '+str(elementz)+', '+str(elementx+1)+', '+str(
      elementx)+', '+str(elementycoarse)+', '+str((elementx+1)*(elementz+1))+', '+str(elementx*elementz)+'\n')
df.write('*Elset, Elset=negy_el, generate\n')
df.write('1, '+str(elementx*elementz)+', 1\n')
df.write('*Elset, Elset=posy_el, generate\n')
df.write(str(elementx*elementz*(elementy+elementycoarse-1)+1)+', '+str(elementx*elementz*(elementy+elementycoarse))+', 1\
      n')
df.write('*Elset, Elset=posx_el, generate\n')
```

```
for z in range(1,elementz+1):
  df.write(str(elementx*z)+', '+str((elementx*elementz*(elementy-1)+(elementx*z)))+', '+str(elementx*elementz)+'\n')
  df.write(str(elementx*elementz*elementy+(elementx)*z)+', '+str((elementx*elementz*(elementy+elementycoarse-1)+(elementx
      *z)))+', '+str(elementx*elementz)+'\n')
df.write('*Elset, Elset=negx_el, generate\n')
for z in range(0,elementz):
  df.write(str(1+elementx*z)+', '+str((elementx*elementz*(elementy+elementycoarse-1)+1+elementx*z))+', '+str(elementx*
      elementz)+'\n')
df.write('*Elset, Elset=negz_el, generate\n')
for x in range(0,elementx):
  df.write(str(1+x)+', '+str((elementx*elementz*(elementy+elementycoarse-1)+1+x))+', '+str(elementx*elementz)+'\n')
df.write('*Elset, Elset=posz_el, generate\n')
for x in range(0,elementx):
  df.write(str(elementx*(elementz-1)+1+x)+', '+str((elementx*elementz*(elementy+elementycoarse-1)+elementx*(elementz-1)
      +1+x))+', '+str(elementx*elementz)+'\n')
df.write('*Nset, Nset=fix\n')
df.write(str(elementx+1)+'\n')
df.write('*Nset, Nset=x1, generate\n')
for z in range(1,elementz+2):
  df.write(str((elementx+1)*z)+', '+str(((elementx+1)*(elementz+1)*(elementy+elementycoarse))+(elementx+1)*z)+', '+str((
      elementx+1)*(elementz+1))+'\n')
df.write('*Nset, Nset=z1, generate\n')
for x in range(0,elementx+1):
  df.write(str(((elementx+1)*elementz)+1+x)+', '+str((((elementx+1)*(elementz+1)*(elementy+elementycoarse))+(elementx+1)*(
      elementz)+1+x)+', '+str((elementx+1)*(elementz+1))+'\n')
df.write('*Surface, name=posx, type=element\n')
df.write('posx_el, S4\n')
df.write('*Surface, name=posy, type=element\n')
df.write('posy_el, S5\n')
df.write('*Surface, name=posz, type=element\n')
df.write('posz_el, S2\n')
df.write('*Surface, name=negx, type=element\n')
df.write('negx_el, S6\n')
df.write('*Surface, name=negy, type=element\n')
df.write('negy_el, S3\n')
df.write('*Surface, name=negz, type=element\n')
df.write('negz_el, S1\n')
df.close()
exit()
```

C++ script for the *Meep* FDTD calculation

Appendix4/hom_model_coupled_v7.cpp

```
//
//Meep file for SCM
//
#include <meep.hpp>
#include <fstream>
#include <vector>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#define _USE_MATH_DEFINES
#include <math.h>
#include <mpi.h>
using namespace meep;
//variable definition
//
//variables which have to be defined by the USER
//
```

```
double pmlt=0.1;                        //PML thickness
double airt=0.1;                        //air thickness
double frequencyHz=2.45e9;              //frequency of microwaves
double c_m=299792458;                   //speed of light
double period=24.;                      //evaluation period
double ysource=0.19;                    //y position of source
double w0=0.043;              //waist radius of Gaussian beam
double e0=1;               //standard value of E
double numlim=1.E-8;           //numerical accuracy
//
//global variables for calculation
//
int mynode, totalnodes;
MPI_Status status;
double xd;                              //x calculation domain dimension
double yd;                            //y calculation domain dimension
double yf;                            //y value of the mesh refined area
double zd;                            //z calculation domain dimension
double resolution;                      //amount of pixels per unit distance
double meshconstref;                  //edge length of element in refined mesh
double meshconstcoarse;               //edge length of element in coarse mesh
int inc;                   //current odb increment
std::vector<std::vector<double> > NT;      //saved node temperature of all nodes
string NTfile;                //name of the file containing the NT data in outer part
string epsfile;               //name of the eps quartz file
std::vector<std::vector<double> > epsdata;     //temperature dependent eps values, first column temperature, second column
      real part, third column imaginary part
string avgeps;                //string of the file name containing the eps data
int elementx, elementy, elementz;  //amount of elements in the specific direction
//
//INITALIZE METHODS
//
double eps (const vec &p);        //method to create spatial distribution of the real part of epsilon
complex<double> gauss(const vec &p); //Method to define the gauss profile
void polycrystal(double a, component c);     //Method which define the polycrystal structure of the model
void readinNT();              //Method to read in the temperature data
double getTemp(double point[3]);    //Method to calculate the temperature of the sought point based on the node
     temperatures
double interpolate (double temp, int epsrealim);  //Method to linearly interpolate the eps value
void readineps ();             //Method to read in eps data
//
//Main program#
//
int main(int argc, char **argv) {
  initialize mpi(argc, argv);    //initialize the MPI variables
  MPI_Comm_size(MPI_COMM_WORLD, &totalnodes); //get the amount of MPI nodes
  MPI_Comm_rank(MPI_COMM_WORLD, &mynode);    //get the single MPI nodes
  std::stringstream str_xd;    //string of xd
  std::stringstream str_yd;    //string of yd
  std::stringstream str_yf;    //string of yf
  std::stringstream str_zd;    //string of zd
  std::stringstream str_resolution;    //string of spatial resolution
  std::stringstream str_meshconstref;    //string of mesh constant in refined part
  std::stringstream str_meshconstcoarse;     //string of mesh constant in coarse part
  std::stringstream str_inc;     //string of current inc of coupling
  std::stringstream str_avgeps; //string of the file name containing the eps data
  std::stringstream temp_inc;    //stringstream temp
  std::stringstream str_model;  //string of model name
  str_xd<<argv[1];           //get xd from python script
  str_yd<<argv[2];           //get yd from python script
  str_yf<<argv[3];           //get yf from python script
  str_zd<<argv[4];           //get zd from python script
  str_avgeps<<argv[5];        //get avgeps from python script
  str_resolution<<argv[6];     //get spatial resolution from python script
  str_inc<<argv[7];          //get current inc from python script
  str_meshconstref<<argv[8];      //get meshconstref from python script
  str_meshconstcoarse<<argv[9];  //get meshconstcoarse from python script
  str_model<<argv[10];        //get modelname from python script
  str_inc>>inc;            //save inc on double variable
  temp_inc<<inc-1;           //increment minus 1
  avgeps=str_avgeps.str();      //save avgeps on double variable
  str_xd>>xd;              //save xd on double variable
  str_yd>>yd;              //save yd on double variable
  str_yf>>yf;              //save yf on double variable
  str_zd>>zd;              //save zd on double variable
  str_resolution>>resolution;    //save spatial resolution on double variable
  str_meshconstref>>meshconstref; //save meshconstref on double variable
  str_meshconstcoarse>>meshconstcoarse;    //save meshconstcoarse on double variable
  elementx=(int)((xd-2.*pmlt)/(2.*meshconstref)+numlim);   //calculate total amount of elements in x direction
  elementz=(int)((zd-2.*pmlt)/(2.*meshconstref)+numlim);   //calculate total amount of elements in z direction
  elementy=(int)(yf/meshconstref+numlim)+((int)((yd-yf-2.*pmlt-airt)/meshconstcoarse+numlim));   //calculate total amount
      of element in y direction
  cout<<"elementx="<<elementx<<"elementy= "<<elementy<<" elementz="<<elementz<<endl;
  NTfile="NT_"+str_model.str()+"_"+temp_inc.str()+".txt"; //name of the file containing all of the node temperatures
  //START TO READ IN NT11 data
  if (inc!=1){
    readinNT();
    cout<<"NT of node "<<NT[6][0]<<" in Part microstructure is "<<NT[6][1]<<" degree"<<endl;
  }
  //SAVE EPS DATA
  readineps();     //save eps data
  //
  master_printf("begin MEEP calculation...\n");
  polycrystal(resolution, Ez);   //start MEEP calculation
  master_printf("finished.\n");
  return 0;
}
//
//definition of the different methods
//
//Method to read in NT values
void readinNT(){
//read in NT of microstructure nodes
  string sm;
  std::vector<std::string> vm;
  ifstream tm;
  tm.open(NTfile.c_str(), ios::in); //open file
  if (tm.good()==false){         //check if tess file exist
    tm.close();         //if not close file
    cout<<"NT file is missing!!!"<<endl;   //write error message
    exit(1);          //end program
  }
  while (!tm.eof()){     //loop over all lines of the file
    getline (tm,sm);   //read out line
    vm.push_back(sm); //save line
  }
  tm.close();
  for (unsigned int j=0; j<vm.size()-1; j++){ //loop over all lines
    std::vector<std::string> temp;
    istringstream iss(vm[j]);
    copy(istream_iterator<string>(iss), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp));
    std::vector<double> row;
    for (int n=0; n<2; n++){ // save node ID and temperature
      row.push_back(atof(temp[n].c_str()));    //save the components of a line
    }
    NT.push_back(row); //insert row in NT_micro
  }
}
//Method to read in the eps data
void readineps (){
  string s;
  std::vector<std::string> v;
  ifstream t;
  t.open(avgeps.c_str(), ios::in);
  if (t.good()==false){    //check if tess file exist
    t.close();          //if not close file
    cout<<"epsfile is missing!!!"<<endl;   //write error message
    exit(1);          //end program
  }
  while (!t.eof()){
    getline (t,s);
    v.push_back(s);
  }
  t.close();
  for (unsigned int j=0; j<v.size()-1; j++){
    std::vector<std::string> temp;
    istringstream iss(v[j]);
    copy(istream_iterator<string>(iss), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp));
    std::vector<double> row;
    for (int n=0; n<3; n++){ // save temperature, eps real and eps imaginary part
      row.push_back(atof(temp[n].c_str()));   //save the components of a line
    }
    epsdata.push_back(row); //insert row in epsdata
  }
}
//method to create spatial distribution of the real part of epsilon
double eps (const vec &p) {
  double realeps;      //real part of eps
  double temp;        //current temperature
  double point[3];
  if ((p.x()<pmlt)||(p.x()>(xd−pmlt))||(p.y()<(pmlt+airt))||(p.y()>(yd−pmlt))||(p.z()<pmlt)||(p.z()>(zd−pmlt))){   //check
      if material point is in pml area
    realeps=1.;
  }
  else {
    if (inc!=1){  //if the current increment is bigger than 1 calculate the current temperature
      //check in which quarter the point lies
      if (p.x()<=xd/2.&&p.z()<=zd/2.){
        point[0]=(p.x()−pmlt);
        point[1]=(p.y()−pmlt−airt);
        point[2]=(p.z()−pmlt);
      }
      else{
        if (p.x()<=xd/2.){
          point[0]=(p.x()−pmlt);
          point[1]=(p.y()−pmlt−airt);
          point[2]=(zd−2.*pmlt)−(p.z()−pmlt);
        }
        else{
          if (p.z()<=zd/2.){
            point[0]=(xd−2.*pmlt)−(p.x()−pmlt);
            point[1]=(p.y()−pmlt−airt);
            point[2]=(p.z()−pmlt);
          }
          else{
            point[0]=(xd−2.*pmlt)−(p.x()−pmlt);
            point[1]=(p.y()−pmlt−airt);
            point[2]=(zd−2.*pmlt)−(p.z()−pmlt);
          }
        }
      }
      temp=getTemp(point);  //get current temperature of the point
    }
    else{
      temp=25.;    //set current temperature to room temperature
    }
    realeps=interpolate(temp, 0);    //get real part of epsilon through interpolation
  }
  return realeps;
}
//class and methods to define spatial distribution of the imaginary part of epsilon
class my_material:public material_function {
  bool has_conducitivity(component c) {
      if(c == Dz)
            return true;
          else
            return false;
  }
  double conductivity (component c, const vec &l){
    double con;      //conductivity
    double point[3];  //sought point
    double temp;     //current temperature
    if ((l.x()<pmlt)||(l.x()>(xd−pmlt))||(l.y()<(pmlt+airt))||(l.y()>(yd−pmlt))||(l.z()<pmlt)||(l.z()>(zd−pmlt))){ //
     check if material point is in pml area
      con=0;
    }
    else {
      if (inc!=1){  //if the current increment is bigger than 1 calculate the current temperature
        //check in which quarter the point lies
        if (l.x()<=xd/2.&&l.z()<=zd/2.){
          point[0]=(l.x()−pmlt);
          point[1]=(l.y()-pmlt-airt);
          point[2]=(l.z()-pmlt);
        }
        else{
          if (l.x()<=xd/2.){
            point[0]=(l.x()-pmlt);
            point[1]=(l.y()-pmlt-airt);
            point[2]=(zd-2.*pmlt)-(l.z()-pmlt);
          }
          else{
            if (l.z()<=zd/2.){
              point[0]=(xd-2.*pmlt)-(l.x()-pmlt);
              point[1]=(l.y()-pmlt-airt);
              point[2]=(l.z()-pmlt);
            }
            else{
              point[0]=(xd-2.*pmlt)-(l.x()-pmlt);
              point[1]=(l.y()-pmlt-airt);
              point[2]=(zd-2.*pmlt)-(l.z()-pmlt);
            }
          }
        }
        temp=getTemp(point);  //get current temperature of the point
      }
      else{
        temp=25.;    //set current temperature to room temperature
      }
      con=((2*M_PI*(frequencyHz/c_m)*interpolate(temp, 1))/interpolate(temp, 0));      //calculate conductivity
    }
    return con;
  }
};
//Method to define the gauss profile
complex<double> gauss(const vec &p){
  complex<double> amplitude;
  double radius=sqrt(pow(p.x(),2)+pow(p.z(),2));
  amplitude=(e0*exp(-pow((radius/w0),2)));
  p.~vec();
  return amplitude;
}
//method to read out temperature value of odb
double getTemp(double point[3]) {
  double curtemp;    //current temperature
  int nx;        //number of elements in x direction
  int ny;        //number of elements in y direction
  int nz;        //number of elements in z direction
  int nyfine;      //number of elements in y refined
  int foundinelement; //element in which point lie
  double h1, h2, h3, h4, h5, h6, h7, h8;    //shape functions
  int n1, n2, n3, n4, n5, n6, n7, n8;   //node ID
  double r;      //x - coordinate in image space
  double s;      //y - coordinate in image space
  double t;      //z - coordinate in image space
  //calculate FE in which point lies
  nx=((int)(point[0]/meshconstref+numlim));
  nyfine=(int)(yf/meshconstref);
  if (point[1]<=yf){
    ny=((int)(point[1]/meshconstref+numlim));
  }
  else{
    ny=nyfine+((int)((point[1]-yf)/meshconstcoarse+numlim));
  }
  nz=((int)(point[2]/meshconstref+numlim));
  //Check rounding errors
  if (nx>elementx-1){
    nx=elementx-1;
  }
  if (ny>elementy-1){
    ny=elementy-1;
  }
  if (nz>elementz-1){
    nz=elementz-1;
  }
  r=(point[0]-(double)nx*meshconstref)*(2./meshconstref)-1.;  //calculate r coordinate
  if (point[1]>=yf){
    s=(point[1]-((double)nyfine*meshconstref+(double)(ny-nyfine)*meshconstcoarse))*(2./meshconstcoarse)-1.; //calculate s
        coordinate
  }
  else{
    s=(point[1]-(double)ny*meshconstref)*(2./meshconstref)-1.;  //calculate s coordinate
  }
  t=(point[2]-(double)nz*meshconstref)*(2./meshconstref)-1.;  //calculate t coordinate
  foundinelement=(nx+1)+(elementx)*nz+(elementx)*(elementz)*ny; //element ID
  //define shape functions
  h1=(1./8.)*(1-r)*(1-s)*(1-t);
  h2=(1./8.)*(1+r)*(1-s)*(1-t);
  h3=(1./8.)*(1+r)*(1+s)*(1-t);
  h4=(1./8.)*(1-r)*(1+s)*(1-t);
  h5=(1./8.)*(1-r)*(1-s)*(1+t);
  h6=(1./8.)*(1+r)*(1-s)*(1+t);
  h7=(1./8.)*(1+r)*(1+s)*(1+t);
  h8=(1./8.)*(1-r)*(1+s)*(1+t);
  n1=(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny;
  n2=(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny;
  n3=(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1);
  n4=(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1);
  n5=(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*ny;
  n6=(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*ny;
  n7=(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1);
  n8=(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1);
  //Check of local coordinates
  if (r<(-1.-numlim)||r>(1.+numlim)||s<(-1.-numlim)||s>(1.+numlim)||t<(-1.-numlim)||t>(1.+numlim)){
    cout<<"ERROR point "<<point[0]<<"x"<<point[1]<<"x"<<point[2]<<" is in element: "<<foundinelement<<" ("<<nx<<", "<<ny
      <<", "<<nz<<") with local coordinates out of limits:"<<r<<", "<<s<<", "<<t<<endl;
  }
  curtemp=h1*NT[n1-1][1]+h2*NT[n2-1][1]+h3*NT[n3-1][1]+h4*NT[n4-1][1]+h5*NT[n5-1][1]+h6*NT[n6-1][1]+h7*NT[n7-1][1]+h8*NT[
      n8-1][1];
  //check if node temperatures are found in file
  if (NT[(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny-1][0]!=((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)
      ){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)<<"not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny-1][0]!=((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)
      ){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)<<"not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)
      *(ny+1))){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  if (NT[(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)
      *(ny+1))){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  if (NT[(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny) -1][0]!=((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny))){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny))<<"not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny) -1][0]!=((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny))){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny))<<"not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny+1))){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  if (NT[(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny+1))){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  return curtemp;
}
//Method to linearly interpolate the eps value
double interpolate (double temp, int epsrealim){
  //search between which boundaries the temp lies
  double inteps;    //interpolated eps value
  if (temp>=epsdata[epsdata.size()-1][0]){
    if (epsrealim==0){    //save real part
      inteps=epsdata[epsdata.size()-2][1]+((epsdata[epsdata.size()-1][1]-epsdata[epsdata.size()-2][1])/(epsdata[epsdata.
      size()-1][0]-epsdata[epsdata.size()-2][0]))*(temp-epsdata[epsdata.size()-2][0]);  //linearly interpolated eps value
    }
    if (epsrealim==1){     //save imaginary part
      inteps=epsdata[epsdata.size()-2][2]+((epsdata[epsdata.size()-1][2]-epsdata[epsdata.size()-2][2])/(epsdata[epsdata.
     size()-1][0]-epsdata[epsdata.size()-2][0]))*(temp-epsdata[epsdata.size()-2][0]);   //linearly interpolated eps value
    }
  }
  else{
    if (temp<epsdata[0][0]){
      if (epsrealim==0){     //save real part
        inteps=epsdata[0][1]+((epsdata[1][1]-epsdata[0][1])/(epsdata[1][0]-epsdata[0][0]))*(temp-epsdata[0][0]);   //
     linearly interpolated eps value
      }
      if (epsrealim==1){     //save imaginary part
        inteps=epsdata[0][2]+((epsdata[1][2]-epsdata[0][2])/(epsdata[1][0]-epsdata[0][0]))*(temp-epsdata[0][0]);   //
     linearly interpolated eps value
      }
    }
    else{
      for (unsigned int i=1; i<epsdata.size(); i++){
        if (temp>=epsdata[i-1][0]&&temp<epsdata[i][0]){ // if temperature is between the two adjacent values
          if (epsrealim==0){     //save real part
            inteps=epsdata[i-1][1]+((epsdata[i][1]-epsdata[i-1][1])/(epsdata[i][0]-epsdata[i-1][0]))*(temp-epsdata[i
     -1][0]);   //linearly interpolated eps value
          }
          if (epsrealim==1){     //save imaginary part
            inteps=epsdata[i-1][2]+((epsdata[i][2]-epsdata[i-1][2])/(epsdata[i][0]-epsdata[i-1][0]))*(temp-epsdata[i
     -1][0]);   //linearly interpolated eps value
          }
        }
      }
    }
  }
  return inteps;     //return interpolated epsilon
}
//Method which define the polycrystal structure of the model
void polycrystal(double a, component c) {
  double start, end, totalstart, totalend;  //variable to calculate time
  totalstart=MPI_Wtime();             //save start time of calculation
  my_material ui;                  //create new material ui
  int timestep=0;
  grid_volume v = vol3d(xd,yd,zd,a);       //define calculation space
  start=MPI_Wtime();
  symmetry S=mirror(X,v)-mirror(Z,v);      //define symmetry planes
  structure s1(v, eps, pml(pmlt), S, 0, 0.5, false, DEFAULT_SUBPIXEL_TOL, DEFAULT_SUBPIXEL_MAXEVAL);  //define the
     structure inside the calculation spase (epsilon real distribution, ...)
  end=MPI_Wtime();
  start=MPI_Wtime();
  s1.set_conductivity(Dz,ui);          //define spatial conductivity (imaginary part of epsilon) distribution
  end=MPI_Wtime();
  fields f1(&s1);                  //initialize field
  f1.use_real_fields();             //indicate that real field should be used
  f1.output_hdf5(Dielectric, v.surroundings());    //create output of the spatial epsilon distribution
  double freq = frequencyHz/c_m;         //frequency in Meep units
  double amplitude=-1.0;              //amplitude of the source
  start=MPI_Wtime();
  continuous_src_time src(freq);         //define a continuous (in time) source
  volume src_plane(vec(0.0,ysource,0.0),vec(xd,ysource,zd));  //define the source plane
  f1.add_volume_source(Hx,src,src_plane,gauss,amplitude);    //add a Hx source
  f1.add_volume_source(Ez,src,src_plane,gauss,amplitude);    //add a Ez source
  end=MPI_Wtime();
  master_printf("volume sources added...\n");
  start=MPI_Wtime();
  //calculate electro-magnetic fields until the end of the selected period is reached
  while (f1.time()<=((period+0.5)/freq)){
    f1.step();
    if (f1.time()>=(period/freq)){       //calculate time average E field for every node of ever element
      timestep++;
      cout<<"total timesteps "<<timestep<<endl;
      if (timestep%2==0){
        f1.output_hdf5(Ex,v.surroundings());  //write the Ex field on a H5 output
        f1.output_hdf5(Ey,v.surroundings());  //write the Ey field on a H5 output
        f1.output_hdf5(Ez,v.surroundings());  //write the Ez field on a H5 output
      }
    }
  }
  end=MPI_Wtime();
  cout<<"total amount of timesteps "<<timestep<<"after "<<end-start<<"s"<<endl;
  totalend=MPI_Wtime();          //Save time when calculation has finished
  cout<<"the total caluclation took "<<totalend-totalstart<<"s "<<endl;
}
```

C++ script to calculate $\overline{E^2}/\overline{E_{0air}^2}$

Appendix4/hdf5_aut_hom_model_coupled_v7.cpp

```
//C++ script to calculate the time averaged squared electric field
//Automatic version to be started by python script
//Version 1.0
//
#include <fstream>
#include <stdio.h>       /* printf, scanf, puts, NULL */
#include <stdlib.h>      /* srand, rand */
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#define _USE_MATH_DEFINES
#include <math.h>
#include <cmath>
#include <vector>
#include "H5Cpp.h"
#ifndef H5_NO_NAMESPACE
#ifndef H5_NO_STD
using std::cout;
using std::endl;
using std::string;
using std::ifstream;
using std::ofstream;
using std::istringstream;
using std::ios;
using std::istream_iterator;
using std::vector;
using std::back_inserter;
#endif // H5_NO_STD
#endif
#ifndef H5_NO_NAMESPACE
using namespace H5;
#endif
//VARIABLES which have to be user defined
const double starttime=5875.;      //starttime of E field calculation (start of one period)
const double endtime=5995.;        //endtime of E-field calculation (End of one period)
const double e0=59650.8;         //E2 at the source position with just air
const double e0x=8.8151e-25;       //E2x at the source position with just air
const double e0y=2.98858e-24;      //E2y at the source position with just air
const double e0z=59650.8;         //E2z at the source position with just air
const double x=0.1;            //x position of coordinate system of bulk material
const double y=0.2;            //y position of coordinate system of bulk material
const double z=0.1;            //z position of coordinate system of bulk material
//
//GLOBAL VARIABLES
std::vector<std::vector<std::vector<double> > > e;     //field in which the E2 data is saved
std::vector<std::vector<std::vector<double> > > ex;    //field in which the E2x data is saved
std::vector<std::vector<std::vector<double> > > ey;    //field in which the E2y data is saved
std::vector<std::vector<std::vector<double> > > ez;    //field in which the E2z data is saved
std::vector<std::vector<std::vector<double> > >datoutput; //variable to save the node data of the output elements
std::vector<std::vector<double> > integrationpointsglobal;  //variable to save coordinates of the integration points
std::vector<int> phases;         //variable to save the phases to the corresponding polyhedra
int nx;                 //amount of points in x-direction; NOTE: nx=ny=nz!!!!
int ny;                 //amount of points in y-direction
int nz;                 //amount of points in z-direction
double h;               //grid constant xd/nx
double ttimestep;            //total amount of time steps within one period
double xd;                //dimension in x-direction of full calculation space
double yd;                   //dimension in y-direction of full calculation space
double zd;                   //dimension in z-direction of full calculation space
int resolution;                //spatial resolution in Pixel/meter
//
//MAIN PROGRAMM
//
int main (int argc, char **argv){
  //VARIABLE DEFINITION
  std::ostringstream model;      //string of model name
  std::stringstream str_xd;      //string of xd
  std::stringstream str_yd;      //string of yd
  std::stringstream str_zd;      //string of zd
  std::stringstream str_resolution; //string of spatial resolution
  std::stringstream str_inc;       //string of current increment
  model<<argv[1];              //get model name from python script
  str_xd<<argv[2];             //get xd from python script
  str_yd<<argv[3];             //get yd from python script
  str_zd<<argv[4];             //get zd from python script
  str_resolution<<argv[5];       //get spatial resolution from python script
  str_inc<<argv[6];            //get inc from python script
  str_xd>>xd;                //save xd on double variable
  str_yd>>yd;                //save yd on double variable
  str_zd>>zd;                //save zd on double variable
  str_resolution>>resolution;      //save resolution on double variable
  H5std_string out_e("output3D_"+model.str()+"_E2_"+str_inc.str()+".h5"); //output file for E2 hdf5 output
  ttimestep=(endtime-starttime)/2.+1;
  cout<<"Amount of timesteps= "<<ttimestep<<endl;
  string component;
  string filename;
  nx=xd*double(resolution);      //amount of grid points in x direction
  ny=yd*double(resolution);      //amount of grid points in y direction
  nz=zd*double(resolution);      //amount of grid points in z direction
  h=1./double(resolution);       //grid constant
  int rank;
  hsize_t dims[3];
  std::ostringstream strs;
  //BODY
  //open first ez file in order to define global array
  component="ez";
  strs<<starttime;
  if (starttime<10000){
    filename=component+"-00000"+strs.str()+".h5";
  }
  else{
    filename=component+"-0000"+strs.str()+".h5";
  }
  cout<<"filename: "<<filename<<endl;
  H5std_string FILE_NAME(filename);
  strs.str("");
  strs.clear();
        H5std_string DATASET_NAME(component);
  H5File file( FILE_NAME, H5F_ACC_RDONLY );      //open file
  DataSet dataset = file.openDataSet( DATASET_NAME ); //open dataset
  DataSpace filespace = dataset.getSpace();      //filespace for rank and dimension
  rank = filespace.getSimpleExtentNdims();       //get number of dimensions in the file dataspace
  rank = filespace.getSimpleExtentDims( dims );    //number of dimensions in the file dataspace
        DataSpace mspace1(rank, dims);
  DSetCreatPropList cparms = dataset.getCreatePlist(); //get properties list
    H5File filewrite_e2(out_e, H5F_ACC_TRUNC );       //create file
  DataSpace filespacewrite=filespace;           //space to write data
  H5std_string DATASET_NAME_WRITE("E2");          //define dataset name
  DataSet datasetwrite_e2=filewrite_e2.createDataSet(DATASET_NAME_WRITE, PredType::NATIVE_DOUBLE, mspace1,cparms);  //
      create dataset
  filespace.close();
  dataset.close();
        file.close();
  cout << "dataset rank = " << rank << ", dimensions "<< dims[0] << " x "<< dims[1] <<" x "<<dims[2]<<endl;
  //allocate e 3D array
  e.resize(nx);
  ex.resize(nx);
  ey.resize(nx);
  ez.resize(nx);
  for (int i=0;i<nx; i++){
    e[i].resize(ny);
    ex[i].resize(ny);
    ey[i].resize(ny);
    ez[i].resize(ny);
    for (int j=0; j<ny; j++){
      e[i][j].resize(nz);
      ex[i][j].resize(nz);
      ey[i][j].resize(nz);
      ez[i][j].resize(nz);
    }
  }
  //create coldata vector in which E field data is saved
  std::vector<std::vector<std::vector<std::vector<double> > > >coldata; //field in which all Efield data is saved:
     coldata[0]=Ex, coldata[1]=Ey, coldata[2]=Ez
  coldata.resize(6);
  for (int k=0; k<6; k++){
    coldata[k].resize(nx);
          for (int i=0;i<nx; i++){
                  coldata[k][i].resize(ny);
                  for (int j=0; j<ny; j++){
                        coldata[k][i][j].resize(nz);
        for (int h=0; h<nz; h++){
          coldata[k][i][j][h]=0.;
        }
                  }
          }
  }
  // sum up E2 data
  for (int i=0; i<3; i++){            //loop over all E field components
    if (i==0)
      component="ex";
    if (i==1)
      component="ey";
    if (i==2)
      component="ez";
    for (double time=starttime; time<=endtime; time=time+2.){ //loop over all times
      strs <<time;
      if (time<10000){
        filename=component+"-00000"+strs.str()+".h5";
      }
      else{
        filename=component+"-0000"+strs.str()+".h5";
      }
      strs.str("");
            strs.clear();
      cout<<"filename "<<filename<<endl;
      H5std_string FILE_NAME(filename);
            H5std_string DATASET_NAME(component);
            H5File file( FILE_NAME, H5F_ACC_RDONLY );                //open file
            DataSet dataset = file.openDataSet( DATASET_NAME );      //open dataset
        DataSpace filespace = dataset.getSpace();                  //filespace for rank and dimension
      double *data_out=new double[nx*ny*nz];       //one dim field in which the outputdata is buffered
      dataset.read( data_out, PredType::NATIVE_DOUBLE, mspace1, filespace );   //read data from hdf5 file
      //loop over all points
      for (int x=0; x<nx; x++){
        for (int y=0; y<ny; y++){
          for (int z=0; z<nz; z++){
            if (time==starttime){
              coldata[i][x][y][z]=pow(((data_out[z+y*nz+nz*ny*x])*376.73),2)/2.;          //half of boundary point, 376.73
     is sqrt(mu_0/eps_0) which accounts for the Meep units
            }
            else{
              if (time==endtime)
                coldata[i][x][y][z]=coldata[i][x][y][z]+pow(((data_out[z+y*nz+nz*ny*x])*376.73),2)/2.;   //half of
     boundary point, 376.73 is sqrt(mu_0/eps_0)
              else
                coldata[i][x][y][z]=coldata[i][x][y][z]+pow((data_out[z+y*nz+nz*ny*x]*376.73),2); //sum up of E2 values (
     components), 376.73 is sqrt(mu_0/eps_0)
            }
          }
        }
      }
      filespace.close();
            dataset.close();
            file.close();
      delete[] data_out;
    }
    }
    //loop over all points
    for (int x=0; x<nx; x++){
            for (int y=0; y<ny; y++){
                    for (int z=0; z<nz; z++){
                            e[x][y][z]=sqrt(pow((coldata[0][x][y][z]/(ttimestep -1)),2)+pow((coldata[1][x][y][z]/(ttimestep
        -1)),2)+pow((coldata[2][x][y][z]/(ttimestep -1)),2))/e0; //calculate E2 values and scale to the source value
        }
                    }
          }
    double *data_write_e2=new double[nx*ny*nz];
    for (int x=0; x<nx; x++){
                    for (int y=0; y<ny; y++){
                            for (int z=0; z<nz; z++){
            data_write_e2[z+y*nz+nz*ny*x]=e[x][y][z];
        }
      }
    }
    datasetwrite_e2.write(data_write_e2, PredType::NATIVE_DOUBLE, mspace1, filespacewrite);
    delete[] data_write_e2;
    filespacewrite.close();
    mspace1.close();
    datasetwrite_e2.close();
    filewrite_e2.close();
    return 0;
}
```

C++ script to calculate P_{abs} at the integration points

Appendix4/hdf5_int_v7_coupled_hom_model.cpp

```
//
//C++ program to calculate absorbed power density at the integration points
//
#include <fstream>
#include <stdio.h>       /* printf, scanf, puts, NULL */
#include <stdlib.h>      /* srand, rand */
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
#define _USE_MATH_DEFINES
#include <math.h>
#include <cmath>
#include <vector>
#include "H5Cpp.h"
#ifndef H5_NO_NAMESPACE
#ifndef H5_NO_STD
using std::cout;
using std::endl;
using std::string;
using std::ifstream;
using std::ofstream;
using std::istringstream;
using std::ios;
using std::istream_iterator;
using std::vector;
using std::back_inserter;
#endif  // H5_NO_STD
#endif
#ifndef H5_NO_NAMESPACE
using namespace H5;
#endif
//VARIABLES which have to be user defined
const double x=0.1;          //x position of coordinate system of bulk material
const double y=0.2;          //y position of coordinate system of bulk material
const double z=0.1;          //z position of coordinate system of bulk material
const double frequency=2.45e9;     //frequency of electromagnetic waves
const double pmlt=0.1;          //thickness of the PML layer
const double airt=0.1;          //thickness of the air layer
double numlim=1.E-8;          //numerical accuracy
//
//GLOBAL VARIABLES
std::vector<std::vector<std::vector<double> > > e;     //field in which the E2 data is saved
int nx;                 //amount of points in x-direction
int ny;                 //amount of points in y-direction
int nz;                 //amount of points in z-direction
double h;               //grid constant xd/nx
std::vector<std::vector<double> >elementsets;      //variable to save the element sets
double xd;                //dimension in x-direction of full calculation space
double yd;                //dimension in y-direction of full calculation space
double yf;                //dimension in y-direction of full calculation space
double zd;                //dimension in z-direction of full calculation space
int resolution;             //spatial resolution of FDTD grid in Pixel/meter
int inc;                //current increment
std::stringstream model;        //string of model name
string outputfile;            //string to the output file
double meshconstref;          //edge length of refined hex mesh
double meshconstcoarse;         //edge length of coarse hex mesh
string avgeps;              //string of the file name containing the eps data
string epsfile;             //name of the eps quartz file
std::vector<std::vector<double> > NT;         //saved node temperature of all nodes
string NTfile;              //name of the file containing the NT data in outer part
std::vector<std::vector<double> > epsdata;       //temperature dependent eps values, first column temperature, second
      column real part, third column imaginary part
int elementx, elementy, elementz, elementyref, elementycoarse;  //amount of elements in the specific direction
//
//INIT MEHTHODS
//
std::vector<std::vector<double> > getintpoint ();
double getE2 (double P [3]);
double interpolate (double temp, int epsrealim);     //Method to linear interpolate the eps value
void readinNT();              //Method to read in the temperature data
double getTemp(double point[3]);     //Method to calculate the temperature of the sought point based on the node
      temperatures
void readineps ();              //Method to read in eps data
//
//MAIN PROGRAMM
//
int main (int argc, char **argv){
  //VARIABLE DEFINITION
  std::stringstream str_xd;      //string of xd
  std::stringstream str_yd;      //string of yd
  std::stringstream str_yf;      //string of yd
  std::stringstream str_zd;      //string of zd
  std::stringstream str_resolution; //string of spatial resolution
  std::stringstream str_avgeps;    //string of file containing the averaged relative permittivity
  std::stringstream str_meshconstref; //string of edge length of the refined mesh
  std::stringstream str_meshconstcoarse;     //string of edge length of the coarse mesh
  std::stringstream temp_inc;      //stringstream temp
  std::stringstream str_inc;       //string of current increment
  model<<argv[1];              //get model name from python script
  str_xd<<argv[2];             //get xd from python script
  str_yd<<argv[3];             //get yd from python script
  str_yf<<argv[4];             //get yf from python script
  str_zd<<argv[5];             //get zd from python script
  str_avgeps<<argv[6];           //get avgeps from python script
  str_resolution<<argv[7];        //get spatial resolution from python script
  str_meshconstref<<argv[8];        //get edge length of the refined mesh
  str_meshconstcoarse<<argv[9];    //get edge length of the coarse mesh
  str_inc<<argv[10];            //get inc from python script
  str_xd>>xd;                //save xd on double variable
  str_yd>>yd;                //save yd on double variable
  str_yf>>yf;                //save yf on double variable
  str_zd>>zd;                //save zd on double variable
  str_meshconstref>>meshconstref;    //save meshconstref on double variable
  str_meshconstcoarse>>meshconstcoarse;    //save meshconstcoarse on double variable
  str_resolution>>resolution;      //save resolution on double variable
  str_inc>>inc;              //save inc on double variable
  temp_inc<<inc-1;             //increment minus 1
  avgeps=str_avgeps.str();       //save avgeps on double variable
  NTfile="NT_"+model.str()+"_"+temp_inc.str()+".txt"; //name of the file containing all of the node temperatures
  elementx=(int)((xd-2.*pmlt)/(2.*meshconstref)+numlim);   //calculate total amount of elements in x direction
```

```
elementz=(int)((zd-2.*pmlt)/(2.*meshconstref)+numlim);   //calculate total amount of elements in z direction
elementy=(int)(yf/meshconstref+numlim)+((int)((yd-yf-2.*pmlt-airt)/meshconstcoarse+numlim));   //calculate total amount
    of element in y direction
elementyref=((int)(yf/meshconstref+numlim));   //calculate amount of refined elements in y direction
elementycoarse=((int)((yd-2.*pmlt-airt-yf)/meshconstcoarse+numlim)); //calculate amount of coarse elements in y
    direction
if (elementyref+elementycoarse!=elementy){
  cout<<"ERROR wrong amount of elements calculation!!!"<<endl;
}
cout<<"elementx="<<elementx<<"elementy= "<<elementy<<" elementz="<<elementz<<" elementyref= "<<elementyref<<"
    elementycoarse= "<<elementycoarse<<endl;
//START TO READ IN NT11 data
if (inc!=1){
  readinNT();
  cout<<"NT of node "<<NT[6][0]<<" in Part microstructure is "<<NT[6][1]<<" degree"<<endl;
}
//SAVE EPS DATA
readineps();     //save eps data
//
outputfile="absorption_101_"+model.str()+"_hom_model_"+str_inc.str()+".txt";
nx=xd*double(resolution);      //amount of grid points in x direction
ny=yd*double(resolution);      //amount of grid points in y direction
nz=zd*double(resolution);      //amount of grid points in z direction
h=1./double(resolution);       //grid constant
int rank;
hsize_t dims[3];
std::ostringstream strs;
double absorption;            //absorbed power density
ofstream abs;             //output string
//BODY
//open first ez file in order to define global array
      H5std_string DATASET_NAME("E2");
H5File e2_data("output3D_"+model.str()+"_E2_"+str_inc.str()+".h5", H5F_ACC_RDONLY );   //open file
DataSet dataset = e2_data.openDataSet( DATASET_NAME );   //open dataset
DataSpace filespace = dataset.getSpace();   //filespace for rank and dimension
rank = filespace.getSimpleExtentNdims();    //get number of dimensions in the file dataspace
rank = filespace.getSimpleExtentDims( dims ); //number of dimensions in the file dataspace
      DataSpace mspace1(rank, dims);
DSetCreatPropList cparms = dataset.getCreatePlist();   //get properties list
cout << "dataset rank = " << rank << ", dimensions "<< dims[0] << " x "<< dims[1] <<" x "<<dims[2]<<endl;
//allocate e 3D array
e.resize(nx);
for (int i=0;i<nx; i++){
  e[i].resize(ny);
  for (int j=0; j<ny; j++){
    e[i][j].resize(nz);
  }
}
// save E2 homogeneous material data
double *data_out=new double[nx*ny*nz];      //one dim field in which the E2 data is buffered
dataset.read(data_out, PredType::NATIVE_DOUBLE, mspace1, filespace);    //read data from E2 hdf5 file
//loop over all points
for (int x=0; x<nx; x++){
  for (int y=0; y<ny; y++){
    for (int z=0; z<nz; z++){
      e[x][y][z]=data_out[z+y*nz+nz*ny*x];     //save e2 value
    }
  }
}
filespace.close();
dataset.close();
e2_data.close();
delete[] data_out;
cout<<"dimensions e: "<<e.size()<<" x "<<e[0].size()<<" x "<<e[0][0].size()<<endl;
cout<<"End h5 saving"<<endl;
double point [3];         //array which contain the coordinates of the integration point
abs.open(outputfile.c_str());
cout<<"begin absorption calculation"<<endl;
cout<<"xd= "<<xd<<" yf= "<<yf<<" zd ="<<zd<<endl;
double avgepsima;            //imaginary part of the effective relative dielectric constant
//loop over all refined elements
for (int iy=0; iy<elementyref; iy++){       //loop over y
  for (int iz=0; iz<elementz; iz++){        //loop over z
    for (int ix=0; ix<elementx; ix++){      //loop over x
      for (int iint=1; iint<=8; iint++){    //loop over integration points
```

```
          if (iint%2==0){
            point[0]=((double)(ix)+1.)*meshconstref;
          }
          else{
            point[0]=((double)(ix))*meshconstref;
          }
          if (iint<=4){
            point[2]=((double)(iz))*meshconstref;
          }
          else{
            point[2]=((double)(iz)+1.)*meshconstref;
          }
          if (iint==1||iint==2||iint==5||iint==6){
            point[1]=((double)(iy))*meshconstref;
          }
          else{
            point[1]=((double)(iy)+1.)*meshconstref;
          }
          if (inc!=1){
            avgepsima=interpolate(getTemp(point), 1);    //calculate the imaginary part of the dielectric constant of the
    current point
          }
          else{
            avgepsima=interpolate(25., 1);             //calculate the imaginary part of the dielectric constant of the
    current point
          }
          absorption=getE2(point)*2*M_PI*frequency*avgepsima*8.85418781762e-12;    //calculate absorbed power density
          abs<<absorption<<endl;
        }
      }
    }
  }
  for (int iy=elementyref; iy<(elementy); iy++){     //loop over y
    for (int iz=0; iz<elementz; iz++){          //loop over z
      for (int ix=0; ix<elementx; ix++){        //loop over x
        for (int iint=1; iint<=8; iint++){      //loop over integration points
          if (iint%2==0){
            point[0]=((double)(ix)+1.)*meshconstref;
          }
          else{
            point[0]=((double)(ix))*meshconstref;
          }
          if (iint<=4){
            point[2]=((double)(iz))*meshconstref;
          }
          else{
            point[2]=((double)(iz)+1.)*meshconstref;
          }
          if (iint==1||iint==2||iint==5||iint==6){
            point[1]=((double)(iy))*meshconstref;
          }
          else{
            point[1]=((double)(iy)+1.)*meshconstref;
          }
//          cout<<"point "<<point[0]<<"x"<<point[1]<<"x"<<point[2]<<" has an E2 value of "<<getE2(point)<<endl;
          if (inc!=1){
            avgepsima=interpolate(getTemp(point), 1);    //calculate the imaginary part of the dielectric constant of the
    current point
          }
          else{
            avgepsima=interpolate(25., 1);             //calculate the imaginary part of the dielectric constant of the
    current point
          }
          absorption=getE2(point)*2*M_PI*frequency*avgepsima*8.85418781762e-12;    //calculate absorbed power density
          abs<<absorption<<endl;
        }
      }
    }
  }
  abs.close();
  return 0;
}
//Method to get e2 value
double getE2 (double P [3]){
  //VARIABLE declaration
  int nx;         //position of the quadrant
  int ny;
  int nz;
  //
  //MAIN Body
  nx=((P[0]+x)/h);     //calculate position in grid, (p[x]+x0-h/2) since different coordinate system is used
  ny=((P[1]+y)/h);
  nz=((P[2]+z)/h);
  return e[nx][ny][nz];
}
//Method to read in NT values
void readinNT(){
//read in NT of microstructure nodes
  string sm;
  std::vector<std::string> vm;
  ifstream tm;
  tm.open(NTfile.c_str(), ios::in); //open file
  if (tm.good()==false){          //check if tess file exist
    tm.close();         //if not close file
    cout<<"NT file is missing!!!"<<endl;  //write error message
    exit(1);          //end program
  }
  while (!tm.eof()){     //loop over all lines of the file
    getline (tm,sm);  //read out line
    vm.push_back(sm); //save line
  }
  tm.close();
  for (unsigned int j=0; j<vm.size()-1; j++){ //loop over all lines
    std::vector<std::string> temp;
    istringstream iss(vm[j]);
    copy(istream_iterator<string>(iss), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp));
    std::vector<double> row;
    for (int n=0; n<2; n++){ // save node ID and temperature
      row.push_back(atof(temp[n].c_str()));    //save the components of a line
    }
    NT.push_back(row); //insert row in NT_micro
  }
}
//Method to read in the eps data
void readineps (){
  string s;
  std::vector<std::string> v;
  ifstream t;
  t.open(avgeps.c_str(), ios::in);
  if (t.good()==false){    //check if tess file exist
    t.close();          //if not close file
    cout<<"epsfile is missing!!!"<<endl;  //write error message
    exit(1);          //end program
  }
  while (!t.eof()){
    getline (t,s);
    v.push_back(s);
  }
  t.close();
  for (unsigned int j=0; j<v.size()-1; j++){
    std::vector<std::string> temp;
    istringstream iss(v[j]);
    copy(istream_iterator<string>(iss), //split of the single lines
      istream_iterator<string>(),
      back_inserter<vector<string> >(temp));
    std::vector<double> row;
    for (int n=0; n<3; n++){ // save temperature, eps real and eps imaginary part
      row.push_back(atof(temp[n].c_str()));    //save the components of a line
    }
    epsdata.push_back(row); //insert row in epsdata
  }
}
//Method to read out temperature value of odb
double getTemp(double point[3]) {
  double curtemp;    //current temperature
  int nx;      //number of elements in x direction
  int ny;      //number of elements in y direction
  int nz;      //number of elements in z direction
  int nyfine;    //number of elements in y refined
  int foundinelement; //element in which point lie
  double h1, h2, h3, h4, h5, h6, h7, h8;    //shape functions
  int n1, n2, n3, n4, n5, n6, n7, n8;   //node ID
  double r;     //x - coordinate in image space
  double s;     //y - coordinate in image space
  double t;     //z - coordinate in image space
  //calculate FE in which point lies
  nx=((int)(point[0]/meshconstref));
  nyfine=(int)(yf/meshconstref);
  if (point[1]<=yf){
    ny=((int)(point[1]/meshconstref));
  }
  else{
    ny=nyfine+((int)((point[1]-yf)/meshconstcoarse));
  }
  nz=((int)(point[2]/meshconstref));
  //Check rounding errors
  if (nx>elementx-1){
    nx=elementx-1;
  }
  if (ny>elementy-1){
    ny=elementy-1;
  }
  if (nz>elementz-1){
    nz=elementz-1;
  }
  r=(point[0]-(double)nx*meshconstref)*(2./meshconstref)-1.;  //calculate r coordinate
  if (point[1]>=yf){
    s=(point[1]-((double)nyfine*meshconstref+(double)(ny-nyfine)*meshconstcoarse))*(2./meshconstcoarse)-1.; //calculate s
       coordinate
  }
  else{
    s=(point[1]-(double)ny*meshconstref)*(2./meshconstref)-1.;  //calculate s coordinate
  }
  t=(point[2]-(double)nz*meshconstref)*(2./meshconstref)-1.;  //calculate t coordinate
  foundinelement=(nx+1)+(elementx)*nz+(elementx)*(elementz)*ny; //element ID
  //define shape functions
  h1=(1./8.)*(1-r)*(1-s)*(1-t);
  h2=(1./8.)*(1+r)*(1-s)*(1-t);
  h3=(1./8.)*(1+r)*(1+s)*(1-t);
  h4=(1./8.)*(1-r)*(1+s)*(1-t);
  h5=(1./8.)*(1-r)*(1-s)*(1+t);
  h6=(1./8.)*(1+r)*(1-s)*(1+t);
  h7=(1./8.)*(1+r)*(1+s)*(1+t);
  h8=(1./8.)*(1-r)*(1+s)*(1+t);
  n1=(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny;
  n2=(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny;
  n3=(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1);
  n4=(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1);
  n5=(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*ny;
  n6=(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*ny;
  n7=(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1);
  n8=(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1);
  //Check of local coordinates
  if (r<(-1.-numlim)||r>(1.+numlim)||s<(-1.-numlim)||s>(1.+numlim)||t<(-1.-numlim)||t>(1.+numlim)){
    cout<<"ERROR point "<<point[0]<<"x"<<point[1]<<"x"<<point[2]<<" is in element: "<<foundinelement<<" ("<<nx<<", "<<ny
      <<", "<<nz<<") with local coordinates out of limits:"<<r<<", "<<s<<", "<<t<<endl;
  }
  curtemp=h1*NT[n1-1][1]+h2*NT[n2-1][1]+h3*NT[n3-1][1]+h4*NT[n4-1][1]+h5*NT[n5-1][1]+h6*NT[n6-1][1]+h7*NT[n7-1][1]+h8*NT[
      n8-1][1];
  //check if node temperatures are found in file
  if (NT[(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny-1][0]!=((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)
      ){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)<<" not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny-1][0]!=((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)
      ){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*ny)<<" not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1)-1][0]!=((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)
      *(ny+1))){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1))<<" not found!!!"<<endl;
  }
```

```
  if (NT[(nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)
      *(ny+1))){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*nz+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  if (NT[(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny) -1][0]!=((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny))){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny))<<"not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny) -1][0]!=((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny))){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny))<<"not found!!!"<<endl;
  }
  if (NT[(nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny+1))){
    cout<<"ERROR node ID "<<((nx+2)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  if (NT[(nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1) -1][0]!=((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(
      elementz+1)*(ny+1))){
    cout<<"ERROR node ID "<<((nx+1)+(elementx+1)*(nz+1)+(elementx+1)*(elementz+1)*(ny+1))<<"not found!!!"<<endl;
  }
  if (nx==5&&ny==10&&nz==20){
    cout<<"point: "<<point[0]<<"x"<<point[1]<<"x"<<point[2]<<" is in element: "<<foundinelement<<" with nodes= "<<n1<<",
      "<<n2<<", "<<n3<<", "<<n4<<", "<<n5<<", "<<n6<<", "<<n7<<", "<<n8<<"with a temperature of "<<curtemp<<endl;
  }
  return curtemp;
}
//Method to linear interpolate the eps value
double interpolate (double temp, int epsrealim){
  //search between which boundaries the temperature lies
  double inteps;    //interpolated eps value
  if (temp>=epsdata[epsdata.size() -1][0]){
    if (epsrealim==0){    //save real part
      inteps=epsdata[epsdata.size() -2][1]+((epsdata[epsdata.size() -1][1]-epsdata[epsdata.size() -2][1])/(epsdata[epsdata.
     size() -1][0]-epsdata[epsdata.size() -2][0]))*(temp-epsdata[epsdata.size() -2][0]);  //linear interpolated eps value
    }
    if (epsrealim==1){    //save imaginary part
      inteps=epsdata[epsdata.size() -2][2]+((epsdata[epsdata.size() -1][2]-epsdata[epsdata.size() -2][2])/(epsdata[epsdata.
     size() -1][0]-epsdata[epsdata.size() -2][0]))*(temp-epsdata[epsdata.size() -2][0]);  //linear interpolated eps value
    }
  }
  else{
    if (temp<epsdata[0][0]){
      if (epsrealim==0){    //save real part
        inteps=epsdata[0][1]+((epsdata[1][1]-epsdata[0][1])/(epsdata[1][0]-epsdata[0][0]))*(temp-epsdata[0][0]);  //
     linear interpolated eps value
      }
      if (epsrealim==1){    //save imaginary part
        inteps=epsdata[0][2]+((epsdata[1][2]-epsdata[0][2])/(epsdata[1][0]-epsdata[0][0]))*(temp-epsdata[0][0]);  //
     linear interpolated eps value
      }
    }
    else{
      for (unsigned int i=1; i<epsdata.size(); i++){
        if (temp>=epsdata[i-1][0]&&temp<epsdata[i][0]){ // if temperature is between the two adjacent values
          if (epsrealim==0){    //save real part
            inteps=epsdata[i-1][1]+((epsdata[i][1]-epsdata[i-1][1])/(epsdata[i][0]-epsdata[i-1][0]))*(temp-epsdata[i
     -1][0]);  //linear interpolated eps value
          }
          if (epsrealim==1){    //save imaginary part
            inteps=epsdata[i-1][2]+((epsdata[i][2]-epsdata[i-1][2])/(epsdata[i][0]-epsdata[i-1][0]))*(temp-epsdata[i
     -1][0]);  //linear interpolated eps value
          }
        }
      }
    }
  }
  return inteps;    //return interpolated epsilon
}
```

Python script to create the *FORTRAN* subroutine

Appendix4/create_BF_subroutine_hom_model_coupled_v7.py

```
########################################################################
# Script which automatically creates the DFLUX and UEXTERNALDB subroutine
########################################################################
#
import sys                          # better than:     import os.sys
import os                           # os = operating system
import argparse                     # to parse arguments
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()            # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created Dflux file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',     type=str, required=True, help='Name of the model.')
parser.add_argument('--initialBF', type=float, required=True, help='Initial constant body heat multiplier.')
parser.add_argument('--nameBF',       type=str, required=True, help='Name of the DFLUX subroutine.')
parser.add_argument('--work',       type=str, required=True, help='Path of the working directory.')
parser.add_argument('--xd', type=float, required=True, help='Dimension in x direction of numerical domain')
parser.add_argument('--yd', type=float, required=True, help='Dimension in y direction of numerical domain')
parser.add_argument('--yf', type=float, required=True, help='Dimension in y direction of refined mesh')
parser.add_argument('--zd', type=float, required=True, help='Dimension in z direction of numerical domain')
parser.add_argument('--meshconstref', type=float, required=True, help='Edge length of the refined mesh')
parser.add_argument('--meshconstcoarse', type=float, required=True, help='Edge length of the coarse mesh')
parser.add_argument('--inc',     type=int, required=True, help='Current increment')
args = parser.parse_args(namespace=parameter)
########################################################################
#Define Variables
#
pmlt=0.1       #thickness of the pml layer
airt=0.1       #thickness of the air layer
numlim=1.E-8     #numerical limit
########################################################################
# Open and write in DFLUX file
#
df=open(str(parameter.nameBF), 'w')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('CCCCC\n')
df.write('CCCCC        Modul to use data in various subroutines\n')
df.write('CCCCC\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n\n')
df.write('      MODULE Information\n\n')
df.write('      DOUBLE PRECISION, DIMENSION(:), allocatable ::\n')
df.write('     *  absorption\n\n')
df.write('C    enthaltene Information:\n')
df.write('C ipoint     Position in absorbed power denisty file (starts with 0)\n')
df.write('C absorption  Absorbed power density on integration point\n\n')
df.write('      save\n')
df.write('      END MODULE Information\n')
df.write('C\n')
df.write('C\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C Subroutine UEXTERNALDB to read in external files\n')
df.write('C\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('      SUBROUTINE UEXTERNALDB(LOP,LRESTART,TIME,DTIME,KSTEP,KINC)\n')
df.write('C\n')
df.write('      USE Information\n')
df.write('C\n')
df.write('      INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('      DIMENSION TIME(2)\n')
df.write('C\n')
df.write('      integer :: inpoint, stat\n')
df.write('C\n')
df.write('        inpoint='+str(int((parameter.xd-2.*pmlt)/(parameter.meshconstref*2.)+numlim)*int((parameter.zd-2.*pmlt)/(
       parameter.meshconstref*2.)+numlim)*(int(parameter.yf/parameter.meshconstref+numlim)+int((parameter.yd-parameter.yf
       -2.*pmlt-airt)/parameter.meshconstcoarse+numlim))*8)+'\n')
df.write('        stat=1\n')
df.write('C\n')
if parameter.inc==1:
  df.write('        IF (LOP==0) THEN\n')
else:
          df.write('  IF (LOP.EQ.4.AND.KINC.EQ.0) THEN\n')
df.write('          ALLOCATE(absorption(inpoint))\n')
df.write('          WRITE(*,*) \'======= EINLESEN   =========\'\n')
path=str(parameter.work)+'/absorption_101_'+str(parameter.model)+'_hom_model_'+str(parameter.inc)
firstline=path[:42]
secondline=path[42:]+'.txt\', iostat=stat, status=\'old\')'
df.write('          OPEN (unit=140, file=\''+firstline+'\n')
if len(secondline)>66:
  secondlinenew=secondline[:66]
  thirdline=secondline[66:]
  df.write('     *'+secondlinenew+'\n')
  df.write('     *'+thirdline+'\n')
else:
  df.write('     *'+secondline+'\n')
df.write('          if (stat==0) then\n')
df.write('            write(*,*) \'Das Oeffnen der Datei hat geklappt\'\n')
df.write('          END IF\n')
df.write('          READ(140,*)(absorption(I), I=1,inpoint)\n')
df.write('          CLOSE(unit=140)\n')
df.write('        END IF\n')
df.write('        RETURN\n')
df.write('        END\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C Subroutine Dflux to calculate body heat flux\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('        SUBROUTINE DFLUX(FLUX,SOL,KSTEP,KINC,TIME,NOEL,NPT,COORDS,\n')
df.write('      1 JLTYP,TEMP,PRESS,SNAME)\n')
df.write('C\n')
df.write('        USE Information\n')
df.write('C\n')
df.write('        INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('        DIMENSION FLUX(2), TIME(2), COORDS(3)\n')
df.write('        CHARACTER*80 SNAME\n')
df.write('        integer :: I\n')
df.write('        double precision BF\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
f.write('C Definition of variable BF (body flux scale factor)\n')
df.write('        BF='+str(parameter.initialBF)+'\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C SOL estimated results variable\n')
df.write('C FLUX body heat flux which has to be defined\n')
df.write('C TIME step \n')
df.write('C COORDS Coordinate of integration point\n')
df.write('C\n')
df.write('        FLUX(1)=BF*absorption((NOEL-1)*8+NPT)\n')
df.write('        RETURN\n')
df.write('        END\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C        Subroutine UVARM to set user defined variables\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('        SUBROUTINE UVARM(UVAR,DIRECT,T,TIME,DTIME,CMNAME,ORNAME,\n')
df.write('      1 NUVARM,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,NDI,NSHR,COORD,\n')
df.write('      2 JMAC,JMATYP,MATLAYO,LACCFLA)\n')
df.write('C\n')
df.write('        USE Information\n')
df.write('C\n')
df.write('        INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('        CHARACTER*80 CMNAME,ORNAME\n')
df.write('        CHARACTER*3 FLGRAY(15)\n')
df.write('        DIMENSION UVAR(NUVARM),DIRECT(3,3),T(3,3),TIME(2)\n')
df.write('       DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)\n')
df.write('       double precision BF\n')
df.write('C      The dimensions of the variables FLGRAY, ARRAY and JARRAY\n')
df.write('C      must be set equal to or greater than 15.\n')
df.write('       BF='+str(parameter.initialBF)+'\n')
df.write('       IF (TIME(1).gt.1.e-2) THEN\n')
df.write('         UVAR(1)=absorption((NOEL-1)*8+NPT)\n')
df.write('       END IF\n')
df.write('       RETURN\n')
df.write('       END\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('C\n')
df.write('C      Subroutine URDFIL to abort calculation after one increment\n')
df.write('CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\n')
df.write('       SUBROUTINE URDFIL(LSTOP,LOVRWRT,KSTEP,KINC,DTIME,TIME)\n')
df.write('C\n')
df.write('       INCLUDE \'ABA_PARAM.INC\'\n')
df.write('C\n')
df.write('       DIMENSION ARRAY(513),JRRAY(NPRECD,513),TIME(2)\n')
df.write('       EQUIVALENCE (ARRAY(1),JRRAY(1,1))\n')
df.write('C\n')
df.write('       LSTOP=1\n')
df.write('C\n')
df.write('       RETURN\n')
df.write('       END\n')
df.close()
exit()
```

Python script to create *Abaqus* heat input files

Appendix4/create_abaqus_input_file_coupled_v2.py

```
########################################################################
# This Python script which creates the Abaqus input file
########################################################################
import sys                              # better than:    import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()                # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created Dflux file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model', type=str, required=True, help='Name of the model.')
parser.add_argument('--horc', type=int, required=True, help='0...create heat model, 1...create stress model.')
parser.add_argument('--inc',      type=int, required=True, help='Current increment')
parser.add_argument('--maxdeltatemp', type=float, required=True, help='Maximum amount of temperature change during one
      increment.')
parser.add_argument('--material', type=str, required=True, help='Name of the material.')
parser.add_argument('--materialfile', type=str, required=True, help='Name of the material file.')
parser.add_argument('--timeheating',     type=float, required=True, help='Time to heat sample.')
args = parser.parse_args(namespace=parameter)
########################################################################
# Open and write in DFLUX file
if (parameter.horc==0 and parameter.inc==1):
  df=open('j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(parameter.inc)+'.inp', 'w')
  df.write('** Automatic created abaqus model for heat transfer calculation\n')
  df.write('*INCLUDE, Input=model_information_'+str(parameter.model)+'_heating.inp\n')
else:
  if (parameter.horc==1):
    df=open('j101_03_'+str(parameter.nameBF)+'_hom_model.inp', 'w')
    df.write('** Automatic created abaqus model for stress calculation\n')
    df.write('*INCLUDE, Input=model_information_'+str(parameter.model)+'_static.inp\n')
  else:
    if (parameter.horc>1):
      print ("ERROR wrong horc integer value!!!")
if (parameter.inc==1):
  df.write('*Elset, elset=hom_material\n')
  df.write('coarse_elements, refined_elements\n')
  df.write('*Solid Section, elset=hom_material, material='+str(parameter.material)+'\n')
  df.write('1.,\n')
  df.write('*Nset, nset=ALL\n')
  df.write(' nodes_ref, nodes_coarse\n')
if (parameter.horc==0 and parameter.inc==1):
  df.write('*INCLUDE, input='+str(parameter.materialfile)+'.inp\n')
else:
  if (parameter.horc==1):
    df.write('*INCLUDE, input='+str(parameter.materialfile)+'_elastic.inp\n')
if (parameter.inc==1):
  df.write('*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23\n')
  df.write('*Initial Conditions, type=TEMPERATURE\n')
  df.write('ALL, 25.\n')
if (parameter.horc==0 and parameter.inc==1):
  df.write('*Step, name=Heating\n')
  df.write('*Heat Transfer, end=PERIOD, deltmx='+str(parameter.maxdeltatemp)+'\n')
  df.write('3., '+str(parameter.timeheating)+', 0.002, '+str(parameter.timeheating)+'\n')
  df.write('*Dflux, OP=MOD\n')
  df.write('hom_material, BFNU\n')
  df.write('*Sfilm\n')
  df.write('negy, F , 25., 20.\n')
  df.write('*Sradiate\n')
  df.write('negy, R, 25., 0.8\n')
  df.write('posy, R, 25., 0.8\n')
  df.write('negx, R, 25., 0.8\n')
  df.write('negz, R, 25., 0.8\n')
  df.write('*Restart, write, frequency=1, overlay\n')
  df.write('*Output, field\n')
  df.write('*Element output\n')
  df.write('TEMP, IVOL, UVARM1\n')
  df.write('*Element Output, POSITION=NODES, directions=YES\n')
  df.write('HFL\n')
  df.write('*Node output\n')
  df.write('NT\n')
  df.write('*Output, history, variable=PRESELECT\n')
  df.write('*ENERGY FILE, frequency=1\n')
  df.write('*End Step\n')
if (parameter.horc==1):
  df.write('*Boundary\n')
  df.write('z1, ZSYMM\n')
  df.write('x1, XSYMM\n')
  df.write('fix, 2, 2\n')
  df.write('*Step, name=Heating\n')
  df.write('*Static\n')
  df.write('0.5, '+str(parameter.timeheating)+', 0.000001,'+str(parameter.timeheating)+'\n')
  df.write('*TEMPERATURE, FILE=j101_01_'+str(parameter.model)+'_hom_model_heating.odb, BSTEP=1\n')
  df.write('ALL_poly\n')
  df.write('*Restart, write, frequency=0\n')
  df.write('*Output, field\n')
  df.write('*Element output\n')
  df.write('S, IVOL, E\n')
  df.write('*Node output\n')
  df.write('NT, U, CF, RF\n')
  df.write('*Output, history, variable=PRESELECT\n')
  df.write('*End Step\n')
  df.write('*Step, name=Cooling\n')
  df.write('*Static\n')
  df.write('10., 3600., 0.00001, 3600.\n')
  df.write('*TEMPERATURE, FILE=j101_01_'+str(parameter.model)+'_hom_model_cooling.odb, BSTEP=1\n')
  df.write('ALL_poly\n')
  df.write('*Restart, write, frequency=0\n')
  df.write('*Output,frequency=5, field\n')
  df.write('*Element output\n')
  df.write('S, IVOL, E\n')
  df.write('*Node output\n')
  df.write('NT, U, CF, RF\n')
  df.write('*Output, history, variable=PRESELECT\n')
  df.write('*End Step\n')
if (parameter.horc==0 and parameter.inc!=1):
  df=open('j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(parameter.inc)+'.inp', 'w')
  df.write('** Automatic created abaqus model for heat transfer calculation\n')
  df.write('*RESTART, READ\n')
df.close()
exit()
```

Python script to read out *Abaqus* data

Appendix4/read_out_odb_hom_model_coupled_v1.py

```
########################################################################
# Python script to read out node temperatures and save it to text files
########################################################################
#
import sys                              # better than:     import os.sys
import os                               # os = operating system
import argparse                         # to parse arguments
import math
import numpy
from odbAccess import *
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()                # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Automatic created stress input file.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model',    type=str, required=True, help='Name of the model.')
parser.add_argument('--inc',     type=int, required=True, help='Current increment.')
args = parser.parse_args(namespace=parameter)
########################################################################
#variable declaration
a=[]
b=[]
c=[]
d=[]
e=[]
f=[]
g=[]
i=0
j=0
k=0
########################################################################
# Main
odb=openOdb(path='j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(parameter.inc)+'.odb')  #open odb
lastFrame=odb.steps['Heating'].frames[-1]   #last frame of step Heating
hom_material=odb.rootAssembly.instances['PART-1-1'] #instance containing the whole model
#START NT saving
NT=lastFrame.fieldOutputs['NT11']      #get all node temperatures
NT_model=NT.getSubset(region=hom_material)  #Node temperatures of the  whole model
fieldValues_NT=NT_model.values         #load field values
for a in fieldValues_NT:
  b.append(a.nodeLabel)          #save ID of the nodes
  c.append(a.data)             #save NT11 value of the node
#write current time to file
t=open('currenttime_'+str(parameter.model)+'_'+str(parameter.inc)+'.txt', 'w')  #open current time file
t.write(str(lastFrame.frameValue))
t.close()
#write NT to file
f=open('NT_'+str(parameter.model)+'_'+str(parameter.inc)+'.txt', 'w') #open NT file
while i<len(b):
  f.write(str(b[i])+' '+str(c[i])+'\n')
  i=i+1
f.close()
exit()
```

Python script to calculate the thermal energy

Appendix4/script_calc_heat_coupled_hom_model_v2.py

```
########################################################################
# Script to calculate thermal energy in SCM
########################################################################
#import  block
import math
import numpy
import argparse
from odbAccess import *
########################################################################
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()             # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Script to calculate the thermal energy.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model', type=str, required=True, help='Name of the model.')
parser.add_argument('--heatout', type=str, required=True, help='Name of the output file in which the thermal energy is
      written.')
parser.add_argument('--inc',     type=int, required=True, help='Current increment')
parser.add_argument('--timeold',     type=float, required=True, help='End time of preceding increment')
parser.add_argument('--aquad',     type=float, required=True, help='Area of the refined mesh of one cuboid side')
parser.add_argument('--material', type=str, required=True, help='Name of the material.')
args = parser.parse_args(namespace=parameter)
########################################################################
#Initialization
aa=[]
ba=[]
ca=[]
da=[]
aao=[]
bao=[]
dao=[]
gb=[]
hb=[]
i_b=[]
tin=25.             #initial temperature in celsius
timeold=parameter.timeold   #initialize oldtime by former increment
Aquad=parameter.aquad     #Area of rectangle of outer elements
########################################################################
#Variables which have to be defined
denhom_material=2667.36     #density of hom_material
########################################################################
#input
inodb="j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(parameter.inc) #new odbfile
inodbold="j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(parameter.inc-1)  #old odbfile
inout=parameter.heatout
########################################################################
#method to interpolate the cp value
def getcp (temp, cptable):
  z=1
  cp=0
  if temp<=cptable[0][1]:
    cp=cptable[0][0]
  else:
    if temp>=cptable[len(cptable)-1][1]:
      cp=cptable[len(cptable)-1][0]
    else:
      while z<len(cptable):
        if cptable[z-1][1]<=temp<=cptable[z][1]:
          cp=cptable[z-1][0]+(temp-cptable[z-1][1])*((cptable[z][0]-cptable[z-1][0])/(cptable[z][1]-cptable[z-1][1]))
          z=len(cptable)
        z=z+1
  if cp==0:
    print "ERROR cp is zero !!! temp= "+str(temp)
  return cp
#calc Energy
def calcE (x, cp, d, dold, den, inc):      #Q=roh*cp*V*deltaT
  if (inc==1):                   #if first increment calculate temperature difference based on initial temperature
```

```
    en=((d[1][x]-tin)*cp*d[2][x]*den)   #d[1]=end Temperature of increment, d[2]=IVOL
  else:                    #if increment is greater than 1 use oldtemperature from old odb file
    if d[0][x]!=dold[0][x]:
      print "ERROR element label from new to old odb are not equal!!!"
    else:
      en=((d[1][x]-dold[1][x])*cp*d[2][x]*den)
  return en
#Method to calculate the average HFL on the boundary elements
def avgHFL (t, time):
  u=[[],[]]
  x=[[],[],[]]
  zz=0
  zx=0
  elementz=0
  #loop over all nodes which are on the front surface
  while zz<len(t[0]):
    elementz=t[0][zz]        #current element node ID
    zzz=0
    inx=0
    while zzz<len(x[0]):       #loop over already saved nodes
      if (x[0][zzz]==elementz): #check if element ID is equal to saved element ID
        x[1][zzz]=x[1][zzz]+t[1][zz][1] #if yes add flux in y-direction
        x[2][zzz]=x[2][zzz]+1 #and update counter
        zzz=len(x[0])       #and end while
        inx=1
      zzz=zzz+1
    if inx==0:               #check if element ID is already saved
      x[0].append(t[0][zz])   #if not save the element ID
      x[1].append(t[1][zz][1])  #and the HFL in y-direction
      x[2].append(1)
    zz=zz+1
  while zx<len(x[0]):
    if (x[1][zx]<0):
      u[0].append(x[0][zx])
      u[1].append(x[1][zx]*(Aquad/4)*time)  #and calculate Q=HFL2*deltatime*A
    zx=zx+1
  return u
#main
odb = openOdb(path=inodb+'.odb')           #open odb file
hom_material=odb.rootAssembly.instances['PART-1-1'].elementSets['HOM_MATERIAL']   #open element set hom_material
boundsurface=odb.rootAssembly.instances['PART-1-1'].nodeSets['Y0']          #set containing front surface nodes
material_hom_material=odb.materials[str(parameter.material)].specificHeat     #read in Cp values of plagioclase
cptable_hom_material=material_hom_material.table
print "cp of "+str(cptable_hom_material[0][1])+" is "+str(cptable_hom_material[0][0])
lastFrame=odb.steps['Heating'].frames[-1]      #save last frame of the step
frame=-1                         #set current frame to last frame of current increment
if (parameter.inc!=1):
  odbold = openOdb(path=inodbold+'.odb')      #open old odb file
  hom_material_old=odbold.rootAssembly.instances['PART-1-1'].elementSets['HOM_MATERIAL']  #open element set hom_material
print "frame ID: "+str(lastFrame.frameId)
f=open(inout+".txt",'w')               #open output file
f.write("Increment  Heating [J]  Flux over boundaries [J] Energy [J]  \n" )
#variable declaration
yza=0
yxa=0
yxb=0
yha=0
ykb=0
einc=0          #accumulated thermal energy
minusinc=0         #accumulated energy of heat flux over the boundaries
########################################################################
#main
curFrame = odb.steps['Heating'].frames[frame]   #just load results from step "Heating"
temp=curFrame.fieldOutputs['TEMP']           #get temperature values of all elements
if (parameter.inc!=1):
  curFrame_old = odbold.steps['Heating'].frames[frame]       #just load results from step "Heating" of old odb
  temp_old=curFrame_old.fieldOutputs['TEMP']     #get temperature values of all elements
vol=curFrame.fieldOutputs['IVOL']           #get IVOL values of all elements
heat=curFrame.fieldOutputs['HFL']           #get HFL values of all elements
timec=curFrame.frameValue              #end time of current instance
time=timec-timeold                  #increment of current instance
fieldHFL_boundsurf=heat.getSubset (region=boundsurface, position=ELEMENT_NODAL, elementType='DC3D8')     #HFL values of
      nodes in set bound surface
fieldHFLVal_boundsurf=fieldHFL_boundsurf.values
```

```
field_hom_material=temp.getSubset (region=hom_material, position=INTEGRATION_POINT, elementType='DC3D8') #TEMP values of
       elements in set hom_material
fieldvol_hom_material=vol.getSubset (region=hom_material, position=INTEGRATION_POINT, elementType='DC3D8') #IVOL values
      of elements in set hom_material
fieldValues_hom_material=field_hom_material.values                #load field values
fieldvolVal_hom_material=fieldvol_hom_material.values
if (parameter.inc!=1):
  field_hom_material_old=temp_old.getSubset (region=hom_material, position=INTEGRATION_POINT, elementType='DC3D8') #TEMP
       values of elements in set hom_material of old odb
  fieldValues_hom_material_old=field_hom_material_old.values                #load field values
#loop over all boundary nodes on poly surface
for yb in fieldHFLVal_boundsurf:
  gb.append(yb.nodeLabel)     #index b stands for boundary
  hb.append(yb.data)
i_b.append(gb)
i_b.append(hb)
#loop over hom_material
for va in fieldValues_hom_material: #index a stands for hom_material
  aa.append(va.elementLabel)    #label of the FE
  ba.append(va.data)        #Temperature of the current integration point
for wa in fieldvolVal_hom_material:
  ca.append(wa.data)        #IVOL value of the current integration point
da.append(aa)
da.append(ba)
da.append(ca)
if (parameter.inc!=1):
  #loop over old odb
  for vao in fieldValues_hom_material_old:  #index a stands for hom_material
    aao.append(vao.elementLabel)  #label of the FE
    bao.append(vao.data)      #Temperature of the current integration point
  dao.append(aao)
  dao.append(bao)
#calculate energy in hom_material per element
while yza<len(da[0]):
  if (parameter.inc==1):
    cpa=getcp((da[1][yza]+tin)/2,cptable_hom_material)
  else:
    cpa=getcp((da[1][yza]+dao[1][yza])/2,cptable_hom_material)
  einc=einc+calcE(yza, cpa, da, dao, denhom_material, parameter.inc)
  yza=yza+1
#calculate HFL out of the grid
kb=avgHFL(i_b, time)
while yxb<len(kb[0]):
  minusinc=minusinc+kb[1][yxb]
  yxb=yxb+1
del i_b[:]
del aa[:]
del ba[:]
del ca[:]
del da[:]
#write results to output file
f.write(str(curFrame.frameId)+"   "+str(einc)+"   "+str(minusinc)+"   "+str(einc+minusinc)+"\n")
f.write("Heating: "+str(einc)+"\n")
f.write("Flux over boundaries: "+str(minusinc)+"\n")
f.write("Total energy input: "+str(einc-minusinc))
f.close()
```

Python script to create the *Abaqus* stress model

Appendix4/create_stress_file_coupled_v5_1.py

```
######################################################################
# Script to create Abaqus stress file for coupled calculation
########################################################################
#import  block
import math
import numpy
import argparse
########################################################################
# Class containing all Parameters which are parsed
class Parameters:
    pass
parameter = Parameters()            # instance of the class Parameters
# Parsing arguments
parser = argparse.ArgumentParser(description='Script to calculate the thermal energy.')
# Adding all necessary and possible arguments to the parser.
parser.add_argument('--model', type=str, required=True, help='Name of the model.')
parser.add_argument('--inc',     type=int, required=True, help='Current increment')
parser.add_argument('--material', type=str, required=True, help='Name of the material.')
parser.add_argument('--materialfile', type=str, required=True, help='Name of the material file.')
args = parser.parse_args(namespace=parameter)
########################################################################
#Initialization
y=0
floatsta=[]
endtime=0 #endtime of current increment
starttime=0 #starttime of current increment
########################################################################
#
#
#output
outinp="j101_03_"+str(parameter.model)+"_hom_model.inp" #stress input file
########################################################################
#main
#open df file
df=open(outinp, 'w')
df.write('** Automatic created abaqus model for stress calculation\n')
df.write('*INCLUDE, Input=model_information_'+str(parameter.model)+'_static.inp\n')
df.write('*Elset, elset=hom_material\n')
df.write('coarse_elements, refined_elements\n')
df.write('*Solid Section, elset=hom_material, material='+str(parameter.material)+'\n')
df.write('1.,\n')
df.write('*Nset, nset=ALL\n')
df.write(' nodes_ref, nodes_coarse\n')
df.write('*INCLUDE, input='+str(parameter.materialfile)+'_elastic.inp\n')
df.write('*Physical Constants, absolute zero=-273.15, stefan boltzmann=1.38065e-23\n')
df.write('*Initial Conditions, type=TEMPERATURE\n')
df.write('ALL, 25.\n')
df.write('*Boundary\n')
df.write('z1, ZSYMM\n')
df.write('x1, XSYMM\n')
df.write('fix, 2, 2\n')
#loop over all increments
for i in range (1, (parameter.inc)):
  # open current sta file
  a=[]
  l=5 #line to search increment
  insta="j101_01_"+str(parameter.model)+"_hom_model_heating_inc"+str(i)+".sta"  #new odbfile
  staread=open(insta, 'r')       #q stands for quartz
  for lineq in staread:        #read in every line
    a.append(lineq.rstrip())     #save each line on a
  staread.close()
  if (a[l][14] is 'U'):
    l=l+1
    if (a[l][14] is 'U'):
      l=l+1
      if (a[l][14] is 'U'):
        l=l+1
  print" Line"+str(l+1)+": "+str(a[l])
  floatsta=[float(x) for x in a[l].split()] #split only line number 6
  endtime=floatsta[6]        #save endtime of current increment
  print "End time of increment "+str(i)+" is: "+str(endtime)
  df.write('*Step, name=Stress'+str(i)+'\n')
  df.write('*Static\n')
  df.write('0., '+str(endtime-starttime)+', 0.000001,'+str(endtime-starttime)+'\n')
  df.write('*TEMPERATURE, FILE=j101_01_'+str(parameter.model)+'_hom_model_heating_inc'+str(i)+'.odb, BSTEP=1\n')
  df.write('ALL\n')
  df.write('*Restart, write, frequency=0\n')
  df.write('*Output, field\n')
  df.write('*Element output\n')
  df.write('S, IVOL, E\n')
  df.write('*Node output\n')
  df.write('NT, U, CF, RF\n')
  df.write('*Output, history, variable=PRESELECT\n')
  df.write('*End Step\n')
  starttime=endtime
df.close()
exit()
```

Appendix E

Thermo-mechanical material parameters of basalt

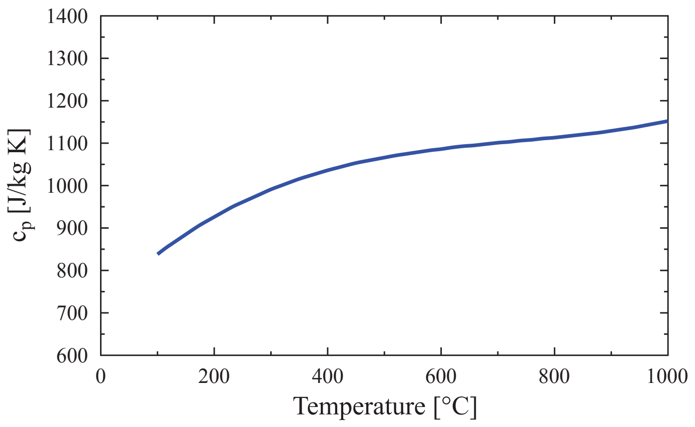


Fig. E.1 Specific heat capacity c_p [$^{J}/_{kg\ K}$] of basalt as a function of temperature [°C] (Hartlieb et al., 2016).

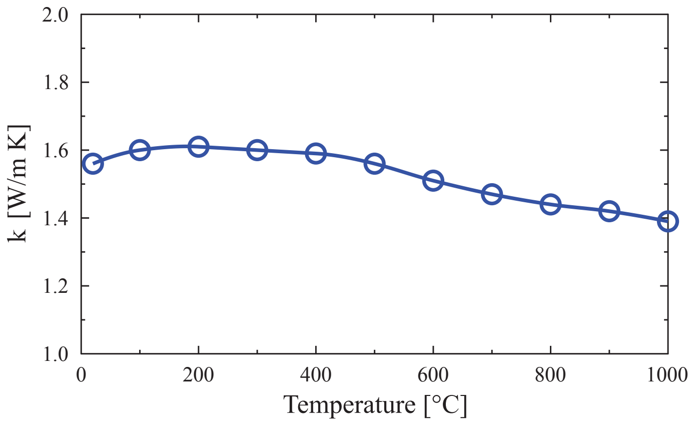


Fig. E.2 Thermal conductance k [$^{W}/_{m\,K}$] of basalt as a function of temperature [°C] (Hartlieb et al., 2016).

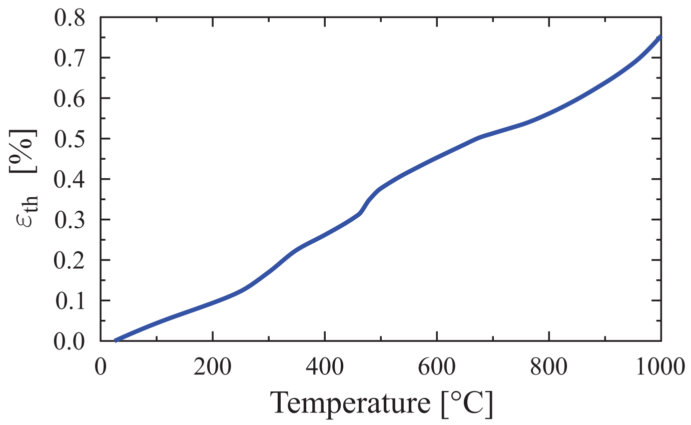


Fig. E.3 Mean uniaxial thermal strain ε_{th} [%] of basalt as a function of temperature [°C] (Hartlieb et al., 2016).

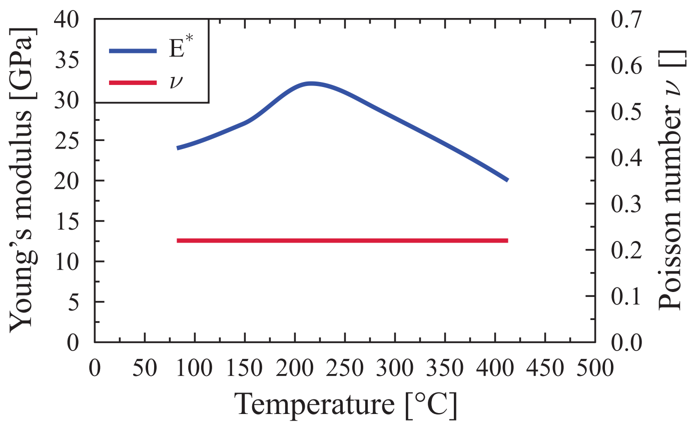


Fig. E.4 Elastic parameters E^* and ν of basalt as a function of temperature [°C] (Peinsitt, 2009).