Tidal Dissipation and Thermal Evolution of Io: A Zero-Dimensional Model with a Basal Magma Ocean

Catherine Gower

St Edmund Hall

University of Oxford

A thesis submitted for the degree of

Master of Earth Sciences in Earth Sciences

Trinity 2025

Word count: 7990

Supervised by Prof. Richard Katz, Dr Hamish Hay, and Hannah Sanderson

Abstract

Io's extreme volcanic activity is driven by intense tidal heating, a result of its rapid, eccentric orbit around Jupiter, maintained by Laplace resonance with Europa and Ganymede. While this intense activity implies significant internal heating, Io's interior structure remains uncertain—particularly the presence of a basal magma ocean, which could strongly influence dissipation.

I present a thermal evolution model of a two-layered body, where a solid shell dissipates energy to sustain a basal magma ocean. This uses a 0-D analytical framework that captures key tidal dissipation behaviours, including rheological dependence and self-gravity. By comparing model predictions to recent *Juno* observations, I show that while Io *could* sustain a molten interior, it likely does not.

Results indicate that viscosity is the primary control on dissipation and that sustaining a magma ocean requires unrealistically low viscosities, consistent with prior research. These findings help resolve the long-standing debate over Io's interior and offer a simplified tool for further studies of tidally heated bodies, in anticipation of the upcoming *Europa Clipper* mission and future exoplanet discoveries.

Acknowledgements

First and foremost, I would like to thank my supervisors—Prof. Richard Katz, Hamish Hay, and Hannah Sanderson—for their guidance, feedback, and encouragement. They were essential for the success of this project, and I'm incredibly grateful to have learned from them.

I'd also like to thank my college tutor, Claire Nichols, for her consistent support and advice during my degree, through both academic challenges and successes.

Contents

List of Tables

List of Figures

1

Introduction

Several planetary bodies in our solar system have higher heat fluxes than expected, including two of Jupiter's moons, Io and Europa, and Enceladus, a moon of Saturn. Io is in many ways analogous to the Earth's Moon, with a similar radius $R \approx 1800\,\mathrm{km}$ and silicate bulk composition (Breuer, Hamilton and Khurana 2022), yet it is the most volcanically active body in the solar system, with surface heat flows exceeding $2\,\mathrm{Wm}^{-2}$ (Pommier and McEwen 2022). Meanwhile, the Moon has a heat flow of just $0.2 - 0.3\,\mathrm{mWm}^{-2}$ (Langseth and Keihm 1977), and the Earth, which is over 300 times larger than Io, has a heat flow of only $\approx 10\,\mathrm{mWm}^{-2}$ (Pommier and McEwen 2022). Europa is estimated to have heat flows of $30 - 45\,\mathrm{mWm}^{-2}$, and Enceladus produces heat flows of $\approx 400\,\mathrm{mWm}^{-2}$ at its poles (Bland et al. 2012). These bodies produce 3 to 200 times the heat of the Earth, raising the question: what mechanisms are driving these intense thermal outputs?

There are three key processes capable of heating on a planetary scale:

1. **Primordial heating**, energy released during accretion and early impacts. For instance, the Moon's early magma ocean likely formed from the immense energy produced in the impact between Theia and the Earth (Borg et al. 2019).

2. **Radiogenic heating**, from the decay of long- and short-lived isotopes, such as Th, U, and Al. This drives heat production on the Earth (Turcotte and Schubert 2014).

3. **Tidal heating**, the focus of this study, which arises from internal friction in response to time-varying tidal forces.

Tides are an effect of gravity, see Figure (1.1). The gravitational attraction between two bodies stretches them both along the axis connecting their centres of mass, raising tidal bulges and deforming them into elongated, rugby-ball-like shapes. The magnitude of deformation depends on the strength of the gravitational attraction—and thus the bodies' masses and radial separation—and their resistance to deformation, which is controlled by their internal structure and rheology. The Earth-Moon system is a familiar example. The Earth is 80 times more massive than the Moon, so one would expect the Moon to experience higher tides. However, the Moon's solid rock tides only reach a maximum height of $10 - 15\,\mathrm{cm}$ (Vogel 2023), while the Earth's surface oceans can exceed 10m (Archer 2013) as they are much easier to deform.

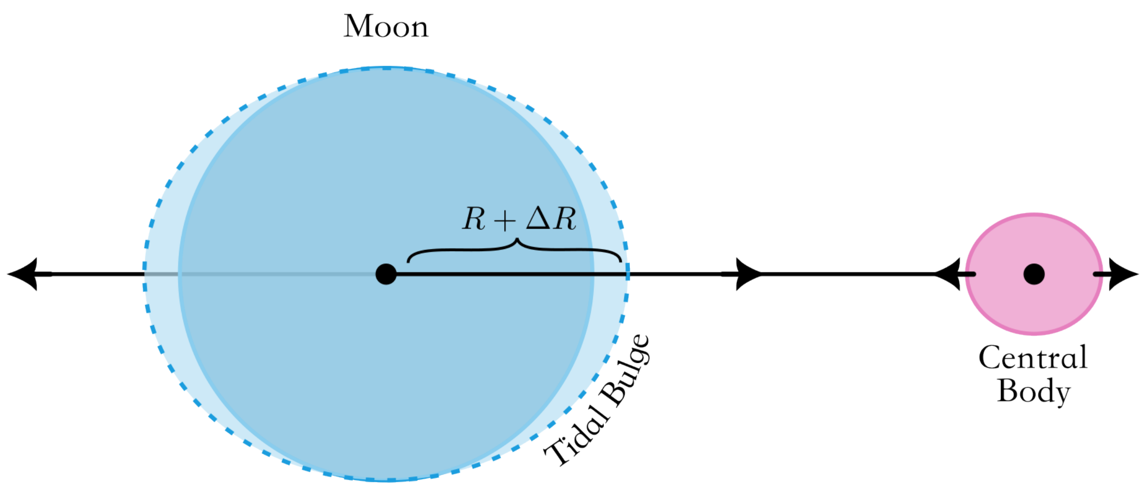

Figure 1.1: A tidal bulge raised on a moon.

1. Introduction

In this study, I consider the deformation of a moon due to the gravity of a central body. As a moon passes through its orbit, variations such as eccentricity, precession, and obliquity can periodically alter the gravitational attraction it experiences. Eccentricity in particular varies its orbital distance and causes the tidal bulge to rise and fall, continuously deforming the moon. If the moon resists deformation, it can dissipate heat according to its rheology.

If the moon behaves elastically, it opposes strain without energy dissipation. However, if the moon can flow, internal friction can convert deformational energy into thermal energy. The amount of heat generated depends on the viscosity. A rigid moon (high viscosity) resists deformation entirely, whilst a fluid moon (low viscosity) deforms easily, but with little internal friction. Significant heat generation occurs when the moon both deforms and dissipates energy.

Io is comparable to the Earth's Moon, with a similar radius, orbital distance, and bulk composition (Williams 2016; Breuer, Hamilton and Khurana 2022). However, Jupiter is around 300 times more massive than the Earth (Matsuyama, Steinke and Nimmo 2022), and Io orbits Jupiter much more rapidly, within only 42.5 hours, versus the Moon's orbit of 27.3 days (Williams 2016). Thus, Io experiences a stronger deformation over a shorter period, yielding intense tidal heating. The other critical component is Io's orbital resonance. Ordinarily, eccentricity dampens over time as tidal heating dissipates energy and the orbit gradually circularises. However, Io's eccentricity is forced by its 4:2:1 Laplace resonance with fellow Galilean moons, Europa and Ganymede (Pommier and McEwen 2022).

Io has long been a focus of scientific exploration. Discovered by Galileo in 1610, the moons of Jupiter were the first bodies found to not orbit the Earth, which contradicted the then prevailing theory of a geocentric solar system (Pommier and McEwen 2022). In 1771, their orbital resonance was described by Laplace, which explained their forced eccentricities (De Kleer, Park and McEwen 2019). However, it wasn't until the 20$^{\text{th}}$ century, 300 years later, that the significance of this resonance was fully recognised by

Peale, Cassen and Reynolds (1979). They predicted that Io's forced eccentricity, combined with Jupiter's massive gravity, would lead to intense tidal heating, proposing Io could be the hottest terrestrial body in the solar system.

Voyager flybys later that year confirmed this prediction. First, the surface shows almost no craters, implying extensive resurfacing, as Io is expected to have a similar impact rate to the inner solar system, which is heavily cratered. Instead, it is covered with volcanic features: calderas, lava flows, and a 270 km high volcanic plume (Smith et al. 1979). The surface composition also appears sulphur-rich, with density and topographic data indicating a silicate interior (Lopes and Williams 2005).

However, Io's topography contradicts Peale, Cassen and Reynolds (1979), who assumed that such high heat fluxes would generate a very thin lithosphere ($d = 18\,\mathrm{km}$) and a largely molten interior. To support the observed 10 km high mountains, the lithospheric thickness must reach $\approx 30\,\mathrm{km}$ (Segatz et al. 1988). Thus, models pivoted to focus on tidal heating in solid mantles, potentially with partially molten asthenospheres (Segatz et al. 1988; Moore 2003).

Observations from the *Galileo* mission, 1989, further constrain Io's interior and support the presence of more extensive internal melting. The heat signatures of many volcanic hotspots exceed expectations for sulphur-driven volcanism, indicating at least some ultramafic activity. This, in turn, points to a partially molten, likely undifferentiated mantle (Lopes and Williams 2005; Khurana et al. 2011). The eastward offset of surface volcanism also supports the presence of a subsurface fluid layer (Breuer, Hamilton and Khurana 2022). Io lacks an intrinsic magnetic field, which precludes core convection, but Jupiter's magnetosphere induces a magnetic response that can identify electrically conductive layers, i.e. the presence of melt. Khurana et al. (2011) demonstrated that magnetometer measurements can be reconciled with a partially or fully molten asthenosphere, reigniting interest in tidal dissipation models involving magma oceans (Roberts and Nimmo 2008; Bierson and Nimmo 2016; Aygün and Čadek 2024a).

Despite these advances, key aspects of Io's interior remain uncertain. Whether tidal dissipation occurs in the shallow or deep mantle, within solid shells, partially molten asthenospheres, or basal magma oceans, dramatically affects both the total heating and the surface expression of volcanism. A basal magma ocean, for example, can decouple the outer shell from the interior, enhancing dissipation and localising it near the surface (Matsuyama, Steinke and Nimmo 2022), a potential explanation for Io's extreme heat fluxes. Heating at shallow depths would concentrate volcanism at low latitudes, as seen on Io, whilst volcanism at the poles indicates heating at depth, if we assume volcanism is representative of heat flow patterns near the surface (Breuer, Hamilton and Khurana 2022).

As the Juno mission returns new data from Io, we will gain improved constraints on its interior structure and heat balance. To anticipate and interpret these observations, this study develops simplified models to capture the key physical processes governing tidal heating. While many existing studies rely on complex 3-D models, the focus here is to investigate fundamental behaviours using reduced models.

The central questions are:

1. What are the key factors controlling tidal dissipation, and when does it become a significant heat source?

2. Can tidal heating sustain molten interiors? How does the presence of a basal magma ocean affect heat production?

3. What can tidal heating tell us about a planet's formation and early thermal evolution?

Chapter (2) presents a 0-D analytical model to examine how key parameters, such as viscosity, influence energy dissipation. The model is verified through the analysis of Love numbers. In Chapter (3), the resulting dissipation rates are incorporated into a thermal evolution framework to assess the magnitude of key heat fluxes, identify equilibrium states, and evaluate the influence of initial conditions. Results indicate that relatively low viscosities are required to generate sufficient tidal heating, and that bodies with higher initial melt fractions are more likely to maintain partially molten interiors over geological timescales. Chapter (4) examines the broader implications of these findings and the models' relevance to Io and other tidally heated bodies.

2

Tidal Dissipation Model

2.1 Model description

I consider a moon in an elliptical orbit, see Figure (2.1). The central body exerts a gravitational force on the moon, which is strongest when they are closest together, at the pericentre passage, and raises the highest tidal bulge. Conversely, the gravitational attraction is the weakest at the apocentre passage, and the bulge is the lowest. Over an orbital period, the tidal bulge oscillates between these two extremes, and thus the moon is constantly deforming.

I simplify the system from 3-D to 0-D by focusing on the point of maximum stress, which is at the equator on the side facing the central planet—the "near-side", or the "sub-jovian side" for Io specifically. By symmetry, the "far-side" also experiences the maximum stress. I isolate the dominant, rugby-ball-like deformation, which is represented by the spherical harmonic degree $l = 2$. I model the moon as a homogenous, solid, spherical body that deforms according to a Maxwell viscoelastic rheology.

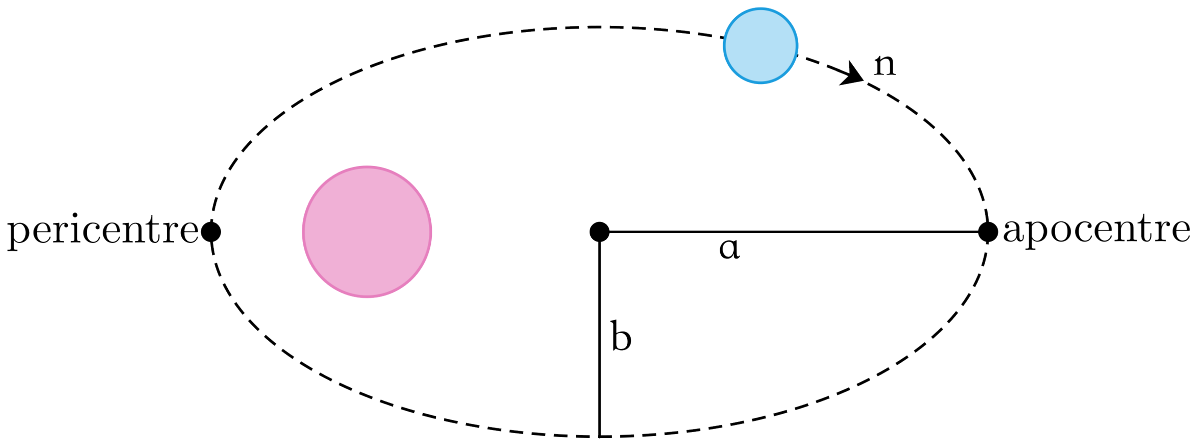

Figure 2.1: An eccentric orbit: A moon (blue) in an orbit of eccentricity, $e = \sqrt{a^2 - b^2}/a$. It orbits the central body (pink) with mean motion, n. It is closest to the central body at the pericentre, and furthest at the apocentre.

2.2 Governing equations

2.2.1 Conservation of Momentum

The conservation of momentum for the moon is given by:

$$\nabla \cdot \sigma = \rho \nabla U^T + \rho \nabla U^G + \rho \nabla U^R \tag{2.1}$$

where σ is the stress tensor, ρ is the density, and the superscripts T, G, and R, denote the tide-raising, spherical-gravity, and response potentials, respectively. Equation (2.1) states that variations in the stress field are driven by gradients in the gravitational potential field acting on the moon.

2.2.2 Gravitational Potentials

The model considers three gravitational potentials:

1. Tide-raising potential

The tide-raising potential due to an eccentric orbit, evaluated at the point closest to the central body, is:

$$\Delta U^T = 3\Omega^2 R^2 e \cos(M) \tag{2.2}$$

where Ω is the moon's spin rate (assumed to equal the orbital mean motion, n, for synchronous rotation), R is the moon's radius, and e is the eccentricity of its orbit about the central body. The mean motion anomaly, $M = n(t - t_0)$, approximates the angular position of the moon along its orbit, where t_0 is the time of the last pericentre passage. The mean motion, $n = 2\pi/T$, is the average speed over the orbital period, T.

This potential arises from the central body's gravitational pull and drives the tidal deformation. Equation (2.2) states that the tide-raising potential is strongest for large bodies, with a fast spin rate, in very eccentric orbits, and is maximum at the pericentre passage.

2. Spherical-gravity potential

The moon's *own* gravity creates a restoring potential that opposes deformation. At the surface it is:

$$\Delta U^G = -g_R R \epsilon \tag{2.3}$$

where g_R is the gravitational acceleration at the surface, and ϵ is the strain, defined as:

$$\epsilon = \frac{2\Delta R}{2R} \tag{2.4}$$

where ΔR is the radial height of the tidal bulge.

This potential represents a body's tendency towards hydrostatic equilibrium. As the surface bulges outward, the spherical-gravity potential increases proportionally to the bulge height, pulling it back down.

3. Response potential

The response potential at the surface is:

$$\Delta U^R = \frac{3}{5} g_R R \epsilon \tag{2.5}$$

The response potential also arises from the moon's self-gravity, but it *enhances* deformation rather than resisting it. As mass is redistributed into the bulge, the additional gravitational force of the bulge reinforces the deformation. Like the restoring potential, this term scales with the bulge height, but it is weaker.

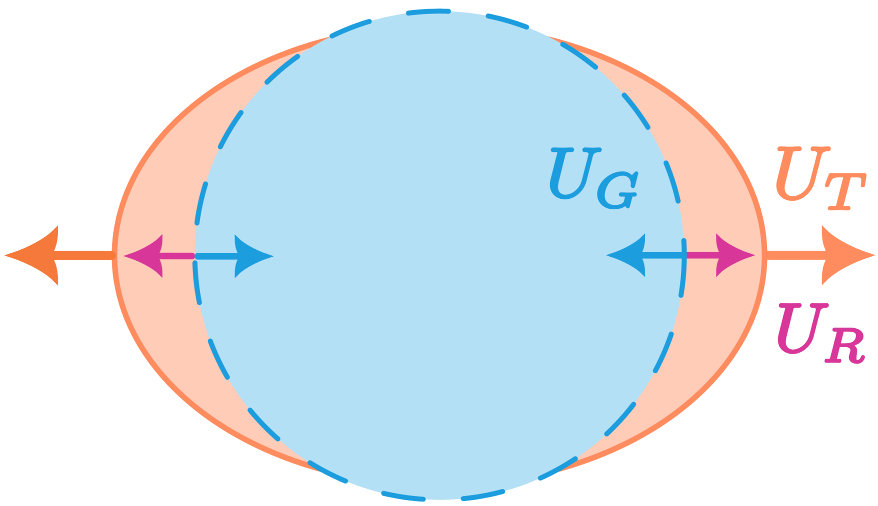

Figure 2.2: The gravitational potentials acting on the moon

Substituting the potentials from Equations (2.2)-(2.5) into the conservation of momentum, Equation (2.1), gives the total stress field. To solve the system, a constitutive law is required to relate stress to strain.

2.2.3 Constitutive Law

I model the moon as a Maxwell viscoelastic material. On short timescales, the material behaves elastically, storing mechanical energy in response to stress. On long timescales, it flows viscously, dissipating energy as heat.

The complex shear modulus captures both the elastic and viscous behaviour, and is given by:

$$\tilde{\mu} = \frac{n^2 \mu}{n^2 + \omega^2} + i \frac{n \mu \omega}{n^2 + \omega^2} \tag{2.6}$$

where μ is the shear modulus and η is the viscosity, representing the material's elastic stiffness and resistance to flow, respectively. The Maxwell frequency is defined as $\omega = \mu/\eta$, and it characterises the transition between elastic and viscous behaviour. The mean motion, n, acts as the forcing frequency.

Equation (2.6) states that:

- When $n \gg \omega$, the moon behaves elastically, and $\tilde{\mu} \approx \mu$.
- When $n \ll \omega$, it behaves viscously, and $\tilde{\mu} \approx in\eta$, dissipating potential energy as heat.

See Figure (2.3).

The complex shear modulus relates the stress and strain through:

$$\sigma = \tilde{\mu} \epsilon \tag{2.7}$$

where σ, ϵ are the stress and strain, respectively. They are assumed to vary periodically in time as:

$$\sigma = \sigma_0 e^{int}, \quad \epsilon = \epsilon_0 e^{int} \tag{2.8}$$

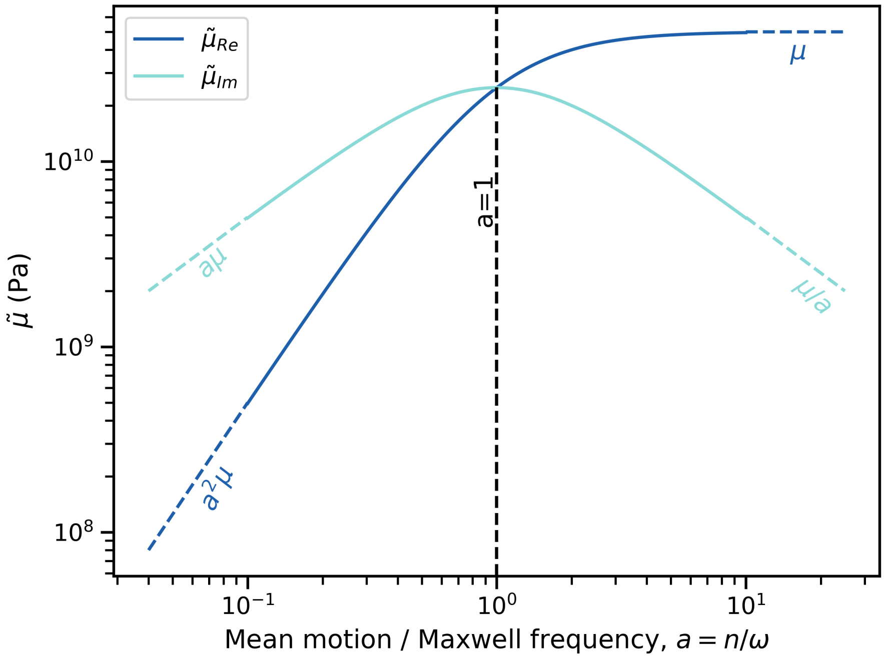

Figure 2.3: Response of $\tilde{\mu}$ to forcing frequency, n: The real component $\tilde{\mu}_{\mathrm{Re}}$ (dark blue) dominates when the forcing frequency, mean motion n, is lower than the Maxwell frequency $\omega = \mu/\eta$. The imaginary component $\tilde{\mu}_{\mathrm{Im}}$ (light blue) dominates when the forcing frequency is higher. The transition in behaviour occurs when the two frequencies are equal (dashed line). Asymptotes of both components are plotted as dashed lines (dark and light blue, respectively).

where σ_0, ϵ_0 are the amplitudes of the stress and strain, respectively. I take the real parts of these expressions. Equation (2.8) reflects that both stress and strain oscillate at the orbital frequency, n, with amplitudes determined by the material's viscoelastic response.

2.2.4 Tidal dissipation rate

The tidal dissipation rate quantifies how much orbital energy from eccentricity is converted into heat due to internal friction within the moon's viscoelastic interior. It is defined volumetrically as:

$$\dot{E}_V = \frac{1}{T} \oint_0^T \sigma \times \dot{\epsilon} \quad dt \tag{2.9}$$

where T is the orbital period, σ is the stress, and $\dot{\epsilon}$ is the time derivative of the strain.

Using the periodic expressions for stress and strain from Equation (2.8), the stress-strain relation in Equation (2.7), and the conservation of momentum, Equation (2.1), the dissipation rate becomes:

$$\dot{E}_V = \frac{9}{8}\frac{n^5 e^2 R^2}{\pi G}\frac{\tilde{\mu}_{\mathrm{Im}}\,\rho g_R R}{\left(\tilde{\mu} + \frac{2}{5}\rho g_R R\right)\left(\tilde{\mu}^* + \frac{2}{5}\rho g_R R\right)} \tag{2.10}$$

where $\tilde{\mu}_{\mathrm{Im}}$ is the imaginary part of the complex shear modulus, and $\tilde{\mu}^*$ is the complex conjugate. $\tilde{\mu}_{\mathrm{Im}}$ represents the viscous dissipation, i.e. the out-of-phase response that lags behind the applied stress and converts mechanical energy into heat.

Equation (2.10) shows that dissipation *increases* with:

- $\tilde{\mu}_{\mathrm{Im}}$, as it increases the frictional resistance,
- n, as it increases the rate of deformation, and
- e, as it increases the variation in deformation over the orbital period.

Dissipation *decreases* with:

- $\frac{2}{5}\rho g_R R$, the moon's "self-gravity", as it reduces the tidal amplitude. It is a combination of the spherical-gravity potential, Equation (2.3), and the response potential, Equation (2.5).
- $\tilde{\mu}$, the moon's "material resistance", which includes the elastic restoring force and the viscous resistance, that both oppose deformation.

2.3 Results

2.3.1 Love numbers

The 0-D model is verified by comparing analytical solutions for the tidal Love numbers, h_2 and k_2, to those of the 3-D model. These dimensionless parameters quantify a body's response to tidal forcing and depend on its interior structure, first introduced by Love (1909). k_2 can be calculated from spacecraft gravity measurements, which is particularly useful for studying bodies like Io, where surface measurements are infeasible.

The model exactly replicates the viscous limit for Love numbers in 3-D and approaches the elastic limit.

Displacement Love number

The displacement Love number, h_2, describes the height of the tidal bulge relative to the ellipsoid and can be written as:

$$h_2 = \frac{\Delta R\, g_R}{U^T} \tag{2.11}$$

I derive the analytical solution:

$$h_2 = \frac{5}{2}\frac{1}{(1 + \frac{5}{2}\frac{\tilde{\mu}}{\rho g_R R})} \tag{2.12}$$

Viscous limit

In the viscous limit, the body behaves as a hypothetical, perfect fluid. The elastic response tends to zero, $\mu \to 0$, and thus $\tilde{\mu} \to 0$. The displacement Love number becomes:

$$h_2 \to \frac{5}{2} \tag{2.13}$$

which matches the solution derived by Love (1909) for the 3-D model. This is the maximum possible value for h_2, as fluids are the easiest to deform.

Elastic limit

In the elastic limit, where viscosity tends to infinity, $\eta \to \infty$, and thus $\tilde{\mu} \to \mu$, the displacement Love number becomes:

$$h_2 \to \frac{5}{2} \frac{1}{(1 + \frac{5}{2} \frac{\mu}{\rho g_R R})} \tag{2.14}$$

Love (1909) derived a similar expression for a 3-D, homogenous, incompressible body of uniform density and rigidity:

$$h_2 = \frac{5}{2} \frac{1}{(1 + \frac{19}{2} \frac{\mu}{\rho g_R R})} \tag{2.15}$$

where the only difference lies in the coefficient, 19/2 versus 5/2.

Response Love number

The response Love number, k_2, describes the modification of the gravitational potential due to the tidal bulge:

$$k_2 = \frac{U_2^R}{U_2^T} \tag{2.16}$$

where U_2^R, U_2^T are the coefficients of the response and tide-raising potentials, respectively. I derive the analytical solution:

$$k_2 = \frac{3}{2} \frac{1}{(1 + \frac{5}{2} \frac{\tilde{\mu}}{\rho g_R R})} \tag{2.17}$$

k_2 is related to h_2:

$$k_2 = \frac{3}{5} h_2 \tag{2.18}$$

which matches the expression derived in Love (1911) .

Viscous limit

In the viscous limit, where $\mu \to 0$ and $\tilde{\mu} \to 0$:

$$k_2 \to \frac{3}{2} \tag{2.19}$$

This matches the upper bound derived by Love (1909) for the 3-D model.

Elastic limit

In the elastic limit, where $\eta \to \infty$ and $\tilde{\mu} \to \mu$:

$$k_2 \to \frac{3}{2} \frac{1}{(1 + \frac{5}{2} \frac{\mu}{\rho g_R R})} \tag{2.20}$$

Again, Love (1909) derived the analogous form for a 3-D body:

$$k_2 = \frac{3}{2} \frac{1}{(1 + \frac{19}{2} \frac{\mu}{\rho g_R R})} \tag{2.21}$$

Comparison

h_2 quantifies the height of the tidal bulge, while k_2 reflects the corresponding perturbation in the gravitational field. Both depend on the dimensionless parameter $\tilde{\mu}/\rho g_R R$, which compares the material to gravitational resistance.

A fluid body, with negligible elastic resistance, has a smaller denominator and thus larger h_2 and k_2 values, indicating high tidal bulges. Conversely, an elastic body has stronger restoring forces, a larger denominator, and correspondingly smaller Love numbers.

The difference in coefficients between the model and Love (1909)—5/2 versus 19/2—likely arises from simplifications in reducing the model to 0-D. The impact is that the model shows reduced resistance to deformation and a slightly higher tidal response. When material resistance dominates ($\tilde{\mu}/\rho g_R R \gg 1$) this difference may be significant. However, when self-gravity dominates ($\tilde{\mu}/\rho g_R R \ll 1$) it is minimal.

Nonetheless, the model matches the analytical solution exactly in the viscous limit, and closely in the elastic limit, while assuming spherical symmetry and solving in 0-D.

2.3.2 Rheological dependence

Tidal dissipation is highly sensitive to the rheological properties of a body, including the shear modulus, μ, and, more significantly, the viscosity, η.

At low viscosities, the body behaves like a fluid, resulting in a large tidal bulge. However, the material offers little resistance to the deformation and therefore dissipates little energy as heat. Conversely, at high viscosities, the body is effectively rigid and resists deformation entirely. Maximum energy dissipation occurs at intermediate viscosities, where the body both deforms and resists with sufficient friction.

For a simple Maxwell viscoelastic body, the dissipation rate peaks when the forcing frequency, n, matches the Maxwell frequency, $\omega = \mu/\eta$ (Ross and Schubert 1986). However, the model also includes the effect of self-gravity, which acts as an additional restoring force, reducing the amplitude of tidal deformation. As a result, the dissipation peak does not occur at exactly the Maxwell frequency.

Dissipation is maximised when the dimensionless ratio $a = n/\omega$ reaches 0.075. For a solid shear modulus of $\mu = 5 \times 10^{10}\,\text{Pa s}$, and Io's mean motion $n = 4.111 \times 10^{-5}\,\text{s}^{-1}$, this corresponds to a viscosity of $\eta = 9.1 \times 10^{13}\,\text{Pa s}$. This is in line with pre-existing literature, such as Hay and Matsuyama (2019), which peaks at $a = 0.02$.

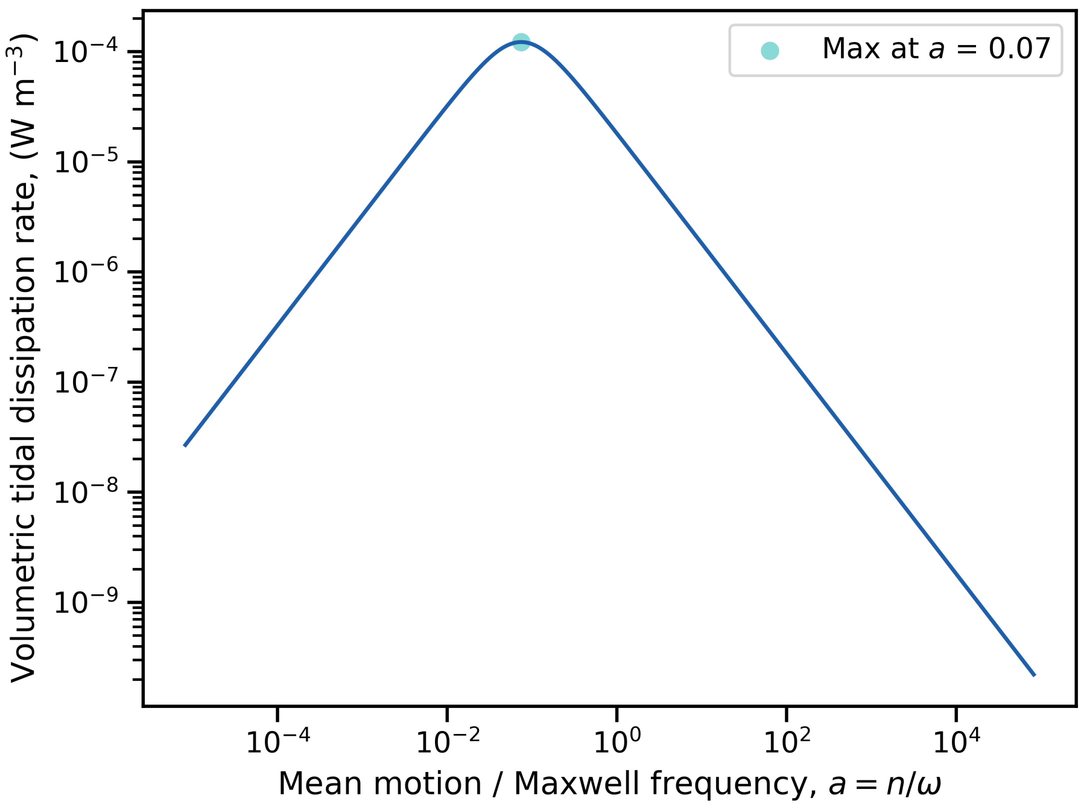

Figure 2.4: Tidal dissipation rate as a function of viscosity: Peak dissipation occurs at an intermediate viscosity, $\eta = 9.1 \times 10^{13}$Pa s, where $a = n/\omega = 0.075$.

2.4 Modification for a basal magma ocean

Chapter (3) considers tidal dissipation in a solid shell of variable thickness. To capture the increased flexibility of thinner shells, I scale the complex shear modulus by the normalised shell thickness:

$$\tilde{\mu}_{\circ} = \lambda \tilde{\mu} \tag{2.22}$$

where $\lambda = d/R$, d is thickness of the shell, and R is the planetary radius. λ is proportional to shell thickness, so the shell's rigidity is reduced for thinner shells, which increases its tidal flexure. This then increases strain and consequently heat dissipation too.

The maximum volumetric dissipation occurs at the optimal $\lambda_{\max}$, which is derived by

Symbol	Quantity	Value	Units
e	Eccentricity	0.0041	
n	Mean motion	4.11×10^{-5}	rad s^{-1}
Ω	Spin rate	n	rad s^{-1}
R	Radius	1.82×10^6	m

Table 2.1: Table of planetary parameters: Values chosen based on Io.

differentiating the tidal dissipation rate with respect to λ, see Appendix (A.3). The expression for λ_{max} is:

$$\lambda_{\text{max}} = \left| \frac{\rho g_R R}{\tilde{\mu}} \right| \tag{2.23}$$

which is also governed by the balance between material and gravitational resistance. If gravitational resistance dominates ($\tilde{\mu} \ll \rho g_R R$), peak dissipation occurs in relatively thicker shells; if material resistance dominates ($\tilde{\mu} \gg \rho g_R R$), a thinner shell can achieve the same level of strain.

As shown in Figure (2.5), the total power output is controlled by two competing effects:

1. Flexibility: Thinner shells deform more easily, increasing the tidal strain.

2. Volume: Thinner shells contain less material, reducing the total volume over which dissipation occurs.

Maximum total dissipation occurs where these two effects balance. Notably, this does not occur at λ_{max}.

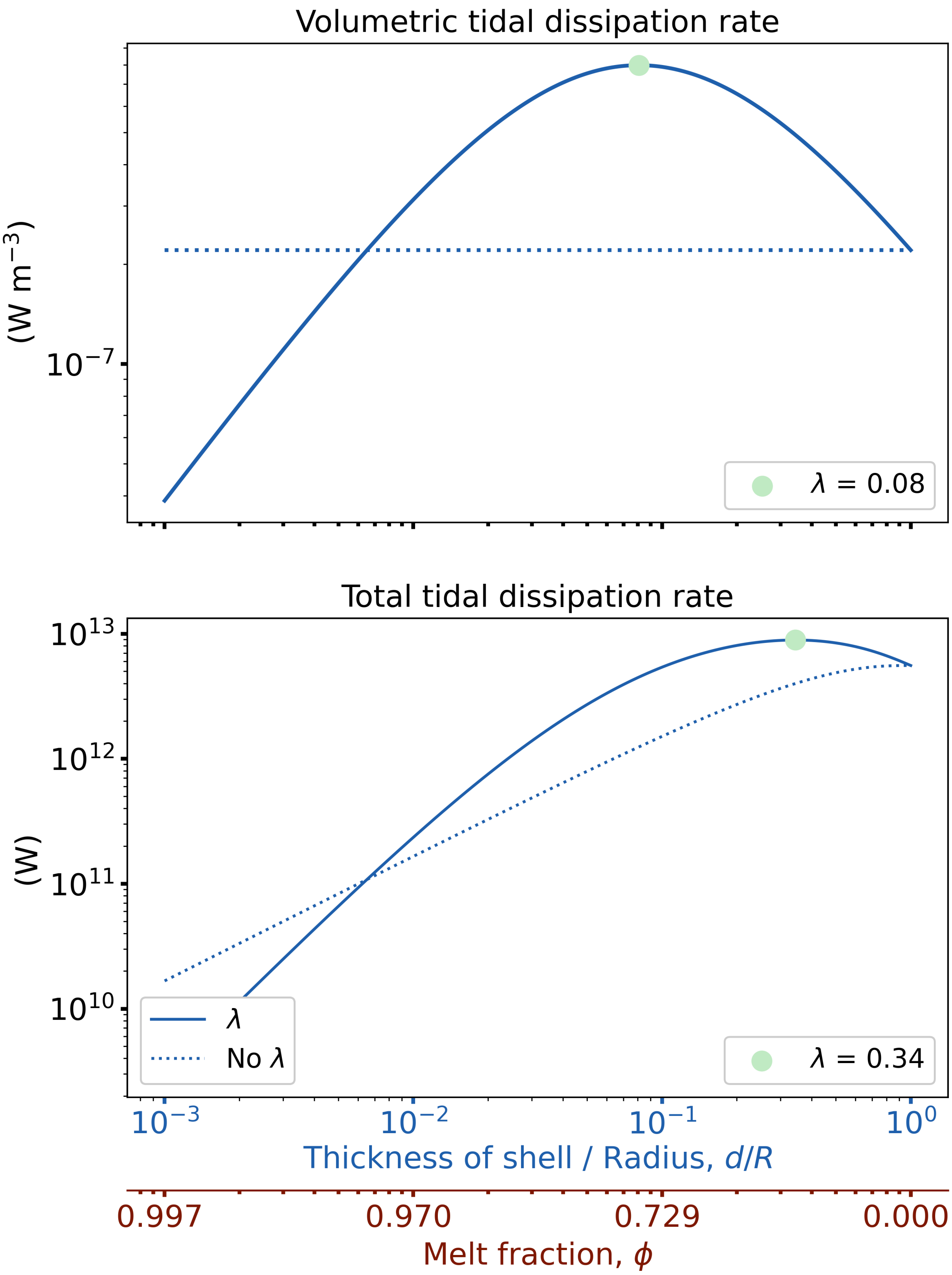

Figure 2.5: Tidal dissipation rate as a function of shell thickness: Volumetric tidal dissipation rate (top) and total tidal dissipation rate (bottom) as a function of normalised shell thickness d/R (or melt fraction ϕ). Shown with (solid) and without (dotted) the inclusion of the λ factor. For $\eta_{\text{diss}} = 10^{17}$.

3

Heat evolution

3.1 Model description

I model the heat evolution of a two-layered moon, with a solid shell overlying a magma ocean. The basal magma ocean is held at the melting temperature for silicates, $T_{\mathrm{ml}} = 1698\,\mathrm{K}$. There is no heat flux into the base of the shell. Instead, the shell is internally heated by tidal dissipation. It cools at the surface, which is held at $T_s = 200\,\mathrm{K}$, through either conduction or convection. Any melt produced in the shell instantaneously sinks to the magma ocean, i.e. I assume the timescale for melt migration is much less than the timescale of convection. To form a basal magma ocean, the melt must be denser than the solid. However, when considering the gravitational acceleration of the moon, I assume that the density difference between the melt and solid is negligible, modelling both as silicates with densities $\rho_m = \rho_s = 3300\,\mathrm{kg\ m^{-3}}$.

I choose different viscosities for tidal dissipation and convection in the shell, η_{tidal} and η_{conv} respectively. The convective viscosities of silicate mantles are well-constrained by Earth observations ($\eta \approx 10^{21}\mathrm{Pa\ s}$, Lambeck, Johnston and Nakada 1990). Thus,

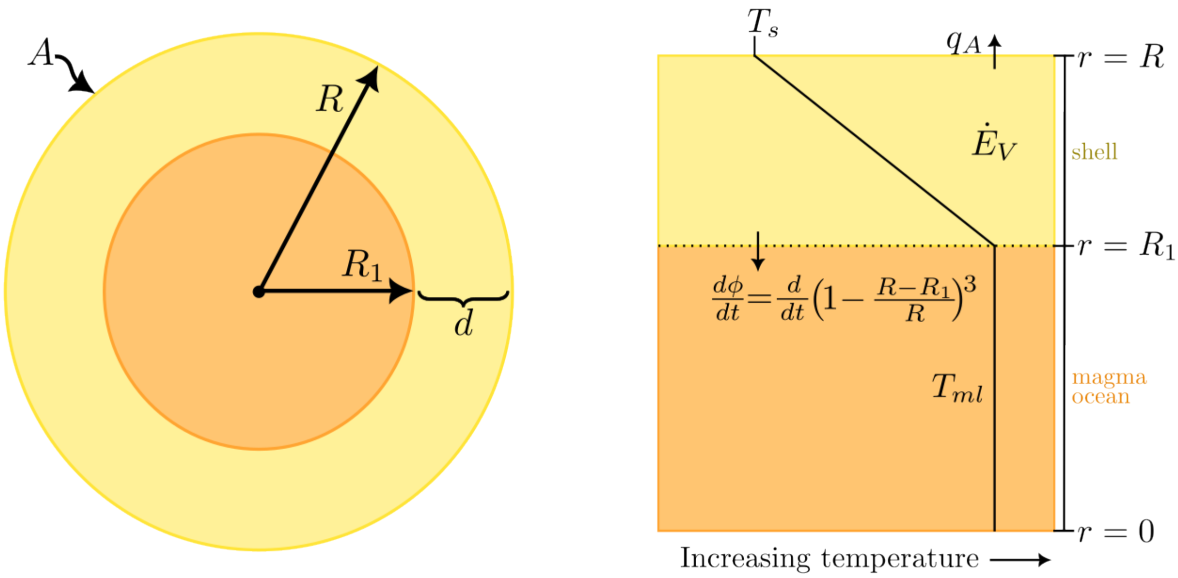

Figure 3.1: A two-layered moon: with surface area A, and a solid shell of radius R (yellow) overlying a basal magma ocean of radius R_1 (orange). The shell is heated by the volumetric tidal dissipation rate $\dot{E}_V$ and cools via the surface heat flux q_A. Any melt produced in the shell is transferred to the magma ocean through the flux $\frac{d\phi}{dt}$. The surface is held at $T_s = 200$K and the magma ocean at $T_{\text{ml}} = 1698$K.

I test convective viscosities in this range ($10^{17} - 10^{25}$Pa s). However, significant tidal dissipation can only be generated by much lower viscosities, see Section (4.1.1). The timescales for these two processes also differ, with dissipation occurring over a 42.5 hour orbit (Williams 2016), whilst convection occurs over $10^{12} - 10^{21}$ hours, for the tested parameters see Appendix (A.4).

3.2 Governing Equations

3.2.1 Conservation of energy

The conservation of energy for the moon is given by:

$$\rho L \frac{d\phi}{dt} = \frac{V_{\mathrm{sh}}}{V} \times \dot{E}_V - \frac{A}{V} \times q_A \tag{3.1}$$

where L is the latent heat of fusion, and ϕ is the melt fraction. $\dot{E}_V$ is the volumetric tidal dissipation rate, and q_A is the surface heat flux. V_{sh} is the volume of the dissipating shell, and V is the volume of the entire moon. A is the surface area of the moon. Equation (3.1) states that tidal heating within the shell is balanced by both melting and heat loss from the moon's surface.

I iterate Equation (3.1) using the forward Euler method, as it is the simplest time integration method (Vuik et al. 2023).

The melt fraction, ϕ, is defined as the ratio of the volume of the ocean, V_{oc}, to the total volume of the moon V:

$$\phi = \frac{V_{\mathrm{oc}}}{V} \tag{3.2}$$

and is thus related to the shell thickness, d:

$$\phi = \left(1 - \frac{d}{R}\right)^3 \tag{3.3}$$

where $d = R - R_1$, and R_1 is the radius of the basal magma ocean.

3.2.2 Heat transfer

Rayleigh number

Heat is transported through either conduction or convection, depending on the Rayleigh number, Ra—a dimensionless parameter that measures the efficiency of convection relative to conduction.

The shell is dominated by tidal heating, which is distributed internally, so I use the internally heated Rayleigh number, rather than that for basal heating. It is defined as follows:

$$\mathrm{Ra} = \frac{\alpha\, g_R\, \rho\, \dot{E}_V\, d^5}{k\, \kappa\, \eta_{\mathrm{conv}}} \tag{3.4}$$

where α is the thermal expansivity, k is thermal conductivity, κ is thermal diffusivity, and η_{conv} is the convective viscosity. d is the thickness of the shell.

If the Rayleigh number is high, the timescale for fluid motion is shorter than the timescale for heating, meaning convection dominates. Conversely, if the Rayleigh number is low, the system remains static, and heat is transferred primarily by conduction. Laboratory and numerical experiments suggest that convection initiates above the critical Rayleigh number, $\mathrm{Ra} > \mathrm{Ra}_{\mathrm{crit}} \approx 10^3$ (Schubert, Turcotte and Olson 2001).

Nusselt number

To quantify the heat transfer efficiency, I introduce the Nusselt number, Nu. It is a ratio of the total heat transfer to purely conductive heat transport:

$$q_A = q_{\mathrm{cond}} \times \mathrm{Nu} \tag{3.5}$$

where q_A is the total heat flux and q_{cond} is the conductive heat flux, given by:

$$q_{\mathrm{cond}} = \frac{k\Delta T}{d} \tag{3.6}$$

A Nusselt number of one, $\mathrm{Nu} = 1$, indicates that heat transfer is only occurring through conduction. If it is greater than one, $\mathrm{Nu} > 1$, convection must be occurring.

I choose Nusselt-Rayleigh scaling to the power of 1/3, which is typical of convective systems, as in Schubert, Turcotte and Olson (2001):

$$\mathrm{Nu} \propto \mathrm{Ra}^{1/3} \tag{3.7}$$

When substituted into Equation (3.5), this gives the total surface heat flux:

$$q_A = \begin{cases} q_{\text{cond}} & \text{if } \text{Ra} < \text{Ra}_{\text{crit}} \\ q_{\text{cond}} \left(\dfrac{\text{Ra}}{\text{Ra}_{\text{crit}}} \right)^{1/3} & \text{if } \text{Ra} \geq \text{Ra}_{\text{crit}} \end{cases} \tag{3.8}$$

which incorporates both the conductive and the convective behaviour. Equation (3.8) states that when the Rayleigh number is high ($\text{Ra} \geq \text{Ra}_{\text{crit}}$) convection will dominate, whereas when it is low ($\text{Ra} < \text{Ra}_{\text{crit}}$) conduction will dominate.

Symbol	Quantity	Value	Units
μ	Shear modulus	5×10^{10}	Pa s
$\rho_s = \rho_m$	Density	3.3×10^3	kg m^{-3}
η_{conv}	Convective viscosity	$10^{17} - 10^{25}$	Pa s
η_{tidal}	Tidal viscosity	$10^{10} - 10^{18}$	Pa s

Table 3.1: Table of rheological parameters: Values chosen based on Io. Convective viscosities, η_{conv}, are chosen close to observed values on Earth ($\eta \approx 10^{21}$Pa s, Lambeck, Johnston and Nakada 1990). Tidal viscosities, η_{tidal}, shown can sustain molten interiors when paired with convective viscosities within this chosen range.

Symbol	Quantity	Value	Units
L	Latent heat of fusion	5×10^5	J kg^{-1}
α	Thermal expansivity	3×10^{-5}	K^{-1}
k	Thermal conductivity	4	W $m^{-1}K^{-1}$
κ	Thermal diffusivity	10^{-6}	m^2s^{-1}
T_{ml}	Melting temperature	1698	K
T_s	Surface temperature	200	K
Ra_{crit}	Critical Rayleigh number	10^3	

Table 3.2: Table of thermal parameters: Values chosen based on Io.

Symbol	Quantity	Value	Units
G	Gravitational constant	6.674×10^{-11}	N m^2kg^2

Table 3.3: Table of universal parameters

Symbol	Quantity	Units
R_1	Ocean radius	m
d	Shell thickness	m
V	Total volume of the moon	m^3
V_{oc}	Volume of the magma ocean	m^3
V_{sh}	Volume of the solid shell	m^3
A	Surface area of the moon	m^2
$\dot{E}_V$	Volumetric tidal dissipation rate	Wm^{-3}
q_A	Outward surface heat flux	Wm^{-2}
q_{cond}	Conductive outward surface heat flux	Wm^{-2}
ΔT	Change in temperature over shell	K
ϕ	Melt fraction	
Ra	Rayleigh number	
Nu	Nusselt number	

Table 3.4: Table of miscellaneous symbols

3.3 Results

3.3.1 Heating regimes

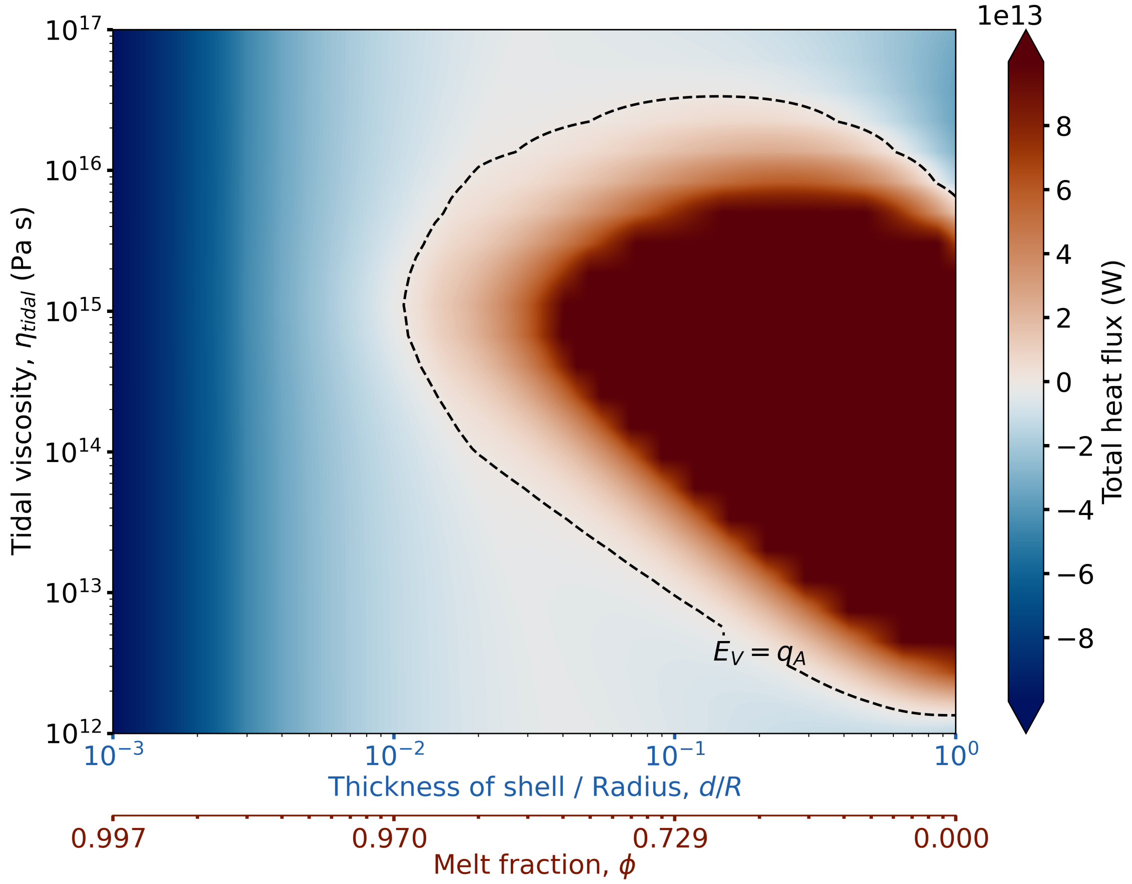

Figure 3.2: Heating versus cooling regimes: as a function of tidal viscosity, η_{tidal}, and normalised shell thickness d/R (or melt fraction, ϕ). For chosen convective viscosity $\eta_{\text{conv}} = 10^{21}$. Equilibrium, where $\dot{E}_V = q_A$, is plotted as a dashed line.

Whether the moon is heating or cooling depends primarily on the tidal viscosity and the shell thickness, see Figure (3.2). The tidal viscosity controls both the tidal dissipation rate and the convective heat flux, as the Rayleigh number is a function of the tidal dissipation rate, $q_{\text{conv}}\left(\text{Ra}(\dot{E}_V)\right)$. Net heating only occurs within a narrow range of tidal viscosities. The total heat flux, $F = \dot{E}_V - q_A$, reaches its maximum at the same tidal viscosity as the peak in tidal dissipation rate, $\dot{E}_V$, which is $\eta_{\text{tidal}} = 9.05 \times 10^{13}$ Pa s.

The heating regime also depends on shell thickness. Thick shells are heating and producing melt, which thins the shell, until it reaches thermal equilibrium where the fluxes balance ($\dot{E}_V = q_A$). Conversely, thin shells are cooling, and the magma ocean crystallises until equilibrium is reached.

3.3.2 Thermal equilibria

The total tidal dissipation rate generally increases with shell thickness, as there is a larger volume to dissipate in. However, this relationship is not linear, due to the trade-off with decreasing shell flexure, as discussed in Section (2.4). At very large shell thicknesses, the total tidal dissipation will decrease despite dissipation occurring over the largest volume, as flexure has reduced greatly.

The outward heat flux is also a function of shell thickness, as shown in Figure (3.3). Thin shells cool conductively, and the outward heat flux decreases with shell thickness. However, as the shell thickens, its Rayleigh number increases. Once it reaches the critical Rayleigh number ($\mathrm{Ra}_{\mathrm{crit}} \approx 10^3$), it will begin to convect, and the outward heat flux increases with shell thickness.

Thus, the inward and outward heat fluxes can intersect at two equilibrium points, as in Figure (3.3), which control the system's evolution:

1. A **stable equilibrium** point, where any change in shell thickness triggers heating or cooling that returns the system to stable equilibrium. This occurs at thinner shell thicknesses.

2. An **unstable equilibrium** point. Any change in shell thickness also triggers heating or cooling, but the system can evolve in two divergent paths. It will either tend towards the stable equilibrium point through heating, or experience runaway cooling. This occurs at higher shell thicknesses.

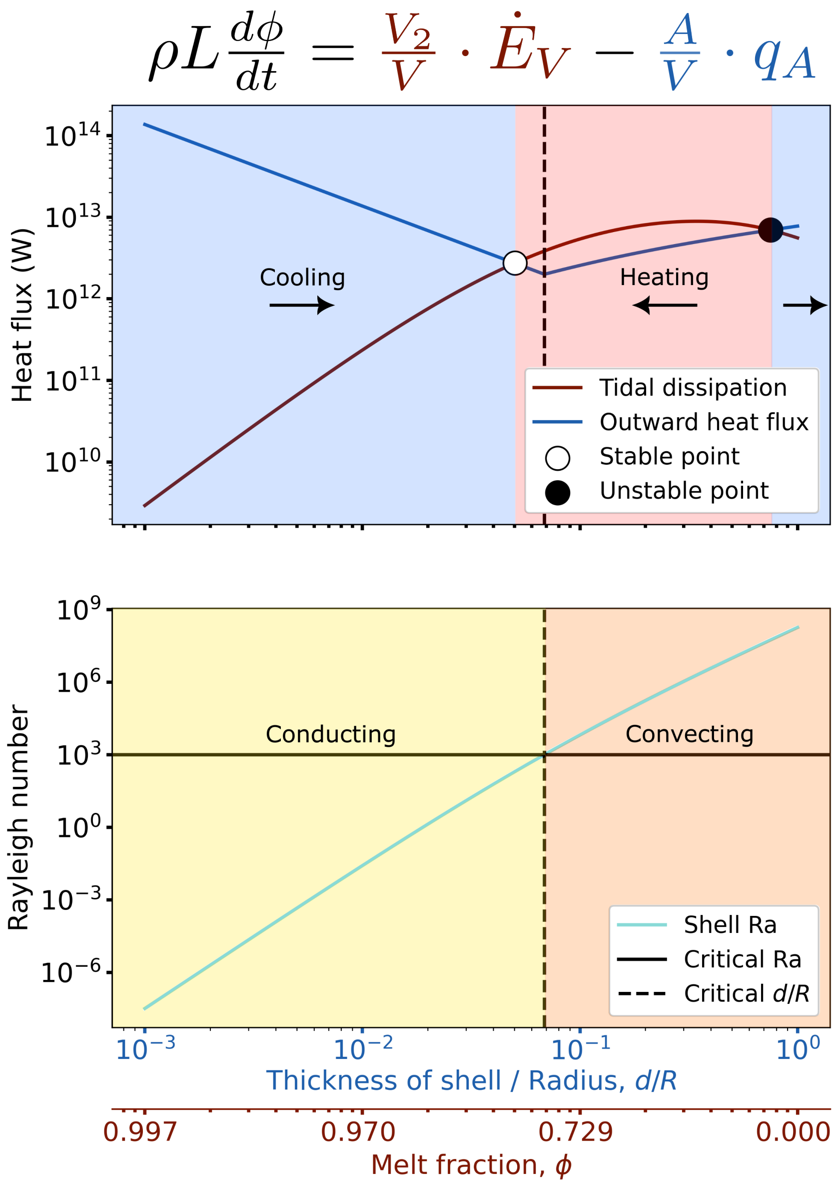

Figure 3.3: Balance of heat fluxes: Heat balance (above) and Rayleigh number (below) as a function of normalised shell thickness d/R (or melt fraction, ϕ). For chosen tidal viscosity $\eta_{\text{tidal}} = 10^{17}$ and convective viscosity $\eta_{\text{conv}} = 10^{21}$.

The tidal dissipation rate over the entire shell, W, (red) intersects the outward heat flux over the shell surface, W, (blue) at the stable (white circle) and unstable (black circle) normalised shell thicknesses, which are $d/R = 0.05$ and $d/R = 0.751$, respectively. The Rayleigh number of the shell, Ra, (teal) reaches the critical Rayleigh number, $\text{Ra}_{\text{crit}} = 10^3$, (black) at a normalised shell thickness (dashed black) of $d/R = 0.069$. The shell is conductive (yellow) below the critical Rayleigh number and convective (orange) above it.

Thus, if the initial shell thickness exceeds the unstable equilibrium point, the moon will cool off and fail to sustain a basal magma ocean. Conversely, if the initial shell thickness is lower than this point, the moon can evolve to maintain a magma ocean, with its volume defined by the stable equilibrium point.

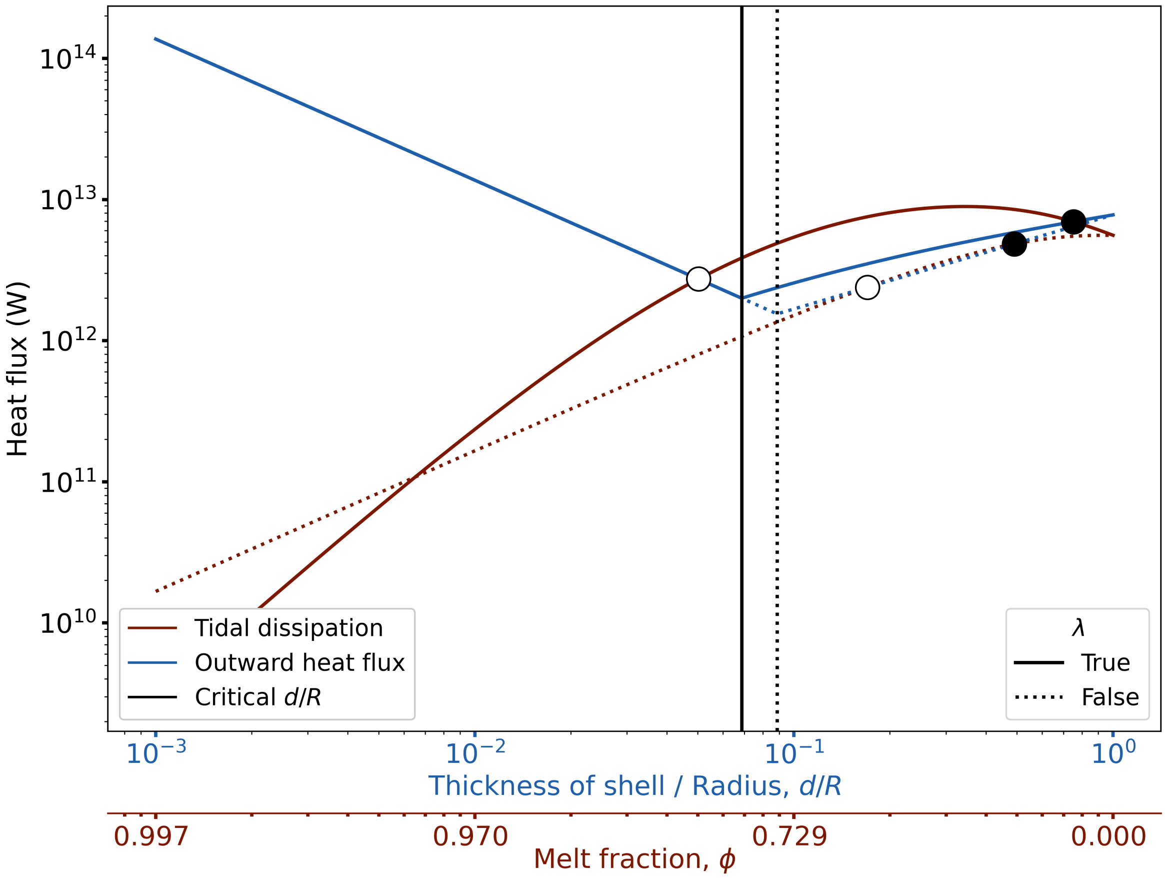

Figure 3.4: Effect of λ on heat fluxes: Heat balance with (solid lines) and without (dotted lines) scaling of the complex shear modulus, $\tilde{\mu}$, by λ. Plotted as a function of normalised shell thickness d/R (or melt fraction, ϕ). For chosen tidal viscosity $\eta_{\text{tidal}} = 10^{17}$ and convective viscosity $\eta_{\text{conv}} = 10^{21}$.

The tidal dissipation rate over the entire shell, W, (red) intersects the outward heat flux over the shell surface, W, (blue) at the stable (white) and unstable (black) points, $d/R = 0.05, 0.751$ and $d/R = 0.17, 0.49$, with and without λ respectively. The normalised shell thickness when the critical Ra is reached (black), is $d/R = 0.069, 0.089$, with and without λ respectively.

Scaling the complex shear modulus by λ impacts the value of both the stable and unstable equilibrium points, which in turn controls the influence of initial conditions, see

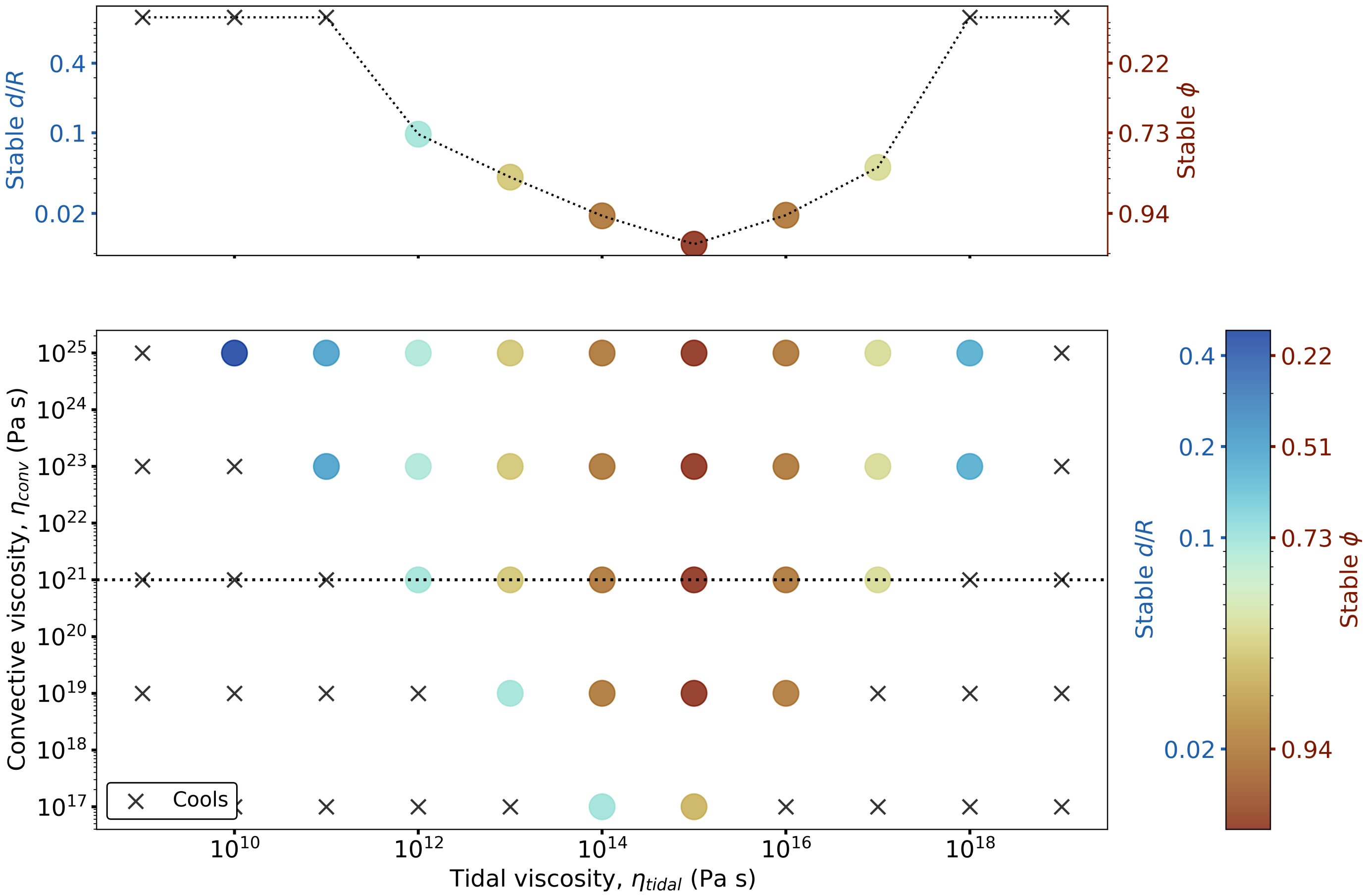

Figure 3.5: Effect of viscosities on stable melt fraction: Stable normalised shell thicknesses d/R (or melt fractions, ϕ), as a function of tidal viscosity η_{tidal} and convective viscosity η_{conv}. A cross-section through convective viscosity $\eta_{\text{conv}} = 10^{21}$ is shown in the top panel.

Figure (3.4). When including λ, the unstable point occurs at higher shell thicknesses, which means that the system will reach stability for a wider range of initial shell thicknesses. In addition, the stable point occurs at lower shell thicknesses, so the moon will sustain larger magma oceans.

3.3.3 Control of viscosity

Tidal viscosity exerts the greatest control on equilibrium conditions, as shown in Figure (3.5). It determines the stable melt fraction, with the highest values achieved when tidal viscosity $\eta_{\text{tidal}} = 10^{15}$ Pa s, for all convective viscosities. If tidal viscosities are too high or too low, the system dissipates less heat, and the stable melt fractions are lower.

The convective viscosity is also significant, as it defines the window of tidal viscosities that can reach equilibrium. Higher convective viscosities decrease the Rayleigh number, and convection is reduced. Thus, the inward and outward heat fluxes can intersect for lower tidal dissipation rates, and a wider range of tidal viscosities can reach equilibrium. If the convective viscosity is too low, fewer tidal viscosities reach equilibrium.

Lower stable melt fractions ($\phi \lessapprox 0.5$) are only reached for very high ($\eta_{\text{tidal}} \gtrapprox 10^{18}\,\text{Pa s}$) or very low ($\eta_{\text{tidal}} \lessapprox 10^{11}\,\text{Pa s}$) tidal viscosities, in combination with high convective viscosities ($\eta_{\text{conv}} \geq 10^{23}\,\text{Pa s}$), where both heating and convection are reduced.

3.3.4 Control of initial conditions

The model exhibits some sensitivity to initial conditions, as shown in Figure (3.6). For a small viscosity range, very low initial melt fractions can cause the moon to cool off and become completely solid.

ϕ_h is the threshold melt fraction. It is the minimum initial melt fraction required to reach a "hot" equilibrium state, i.e. one that will sustain a molten interior. The lowest ϕ_h is 0, for most combinations of viscosities. The highest ϕ_h achieved is 0.017, for a low convective viscosity $\eta_{\text{conv}} = 10^{15}\,\text{Pa s}$ and a tidal viscosity $\eta_{\text{tidal}} = 10^{17}\,\text{Pa s}$. Thus, ϕ_h spans a wide range, depending on the chosen parameters, and most initial conditions lead to sustained molten interiors.

ϕ_s is the stable melt fraction, which a system will tend to if initial melt fractions exceed ϕ_h. For a realistic convective viscosity ($\eta_{\text{conv}} = 10^{21}\,\text{Pa s}$), ϕ_s spans 0.736 to 0.968 which correlates to normalised shell thicknesses of $d/R = 0.011$ to 0.097. Equilibrium states exist for tidal viscosities from $\eta_{\text{tidal}} = 10^{12}$Pa s to 10^{17}Pa s; beyond this, the fluxes do not intersect as tidal heating is too low.

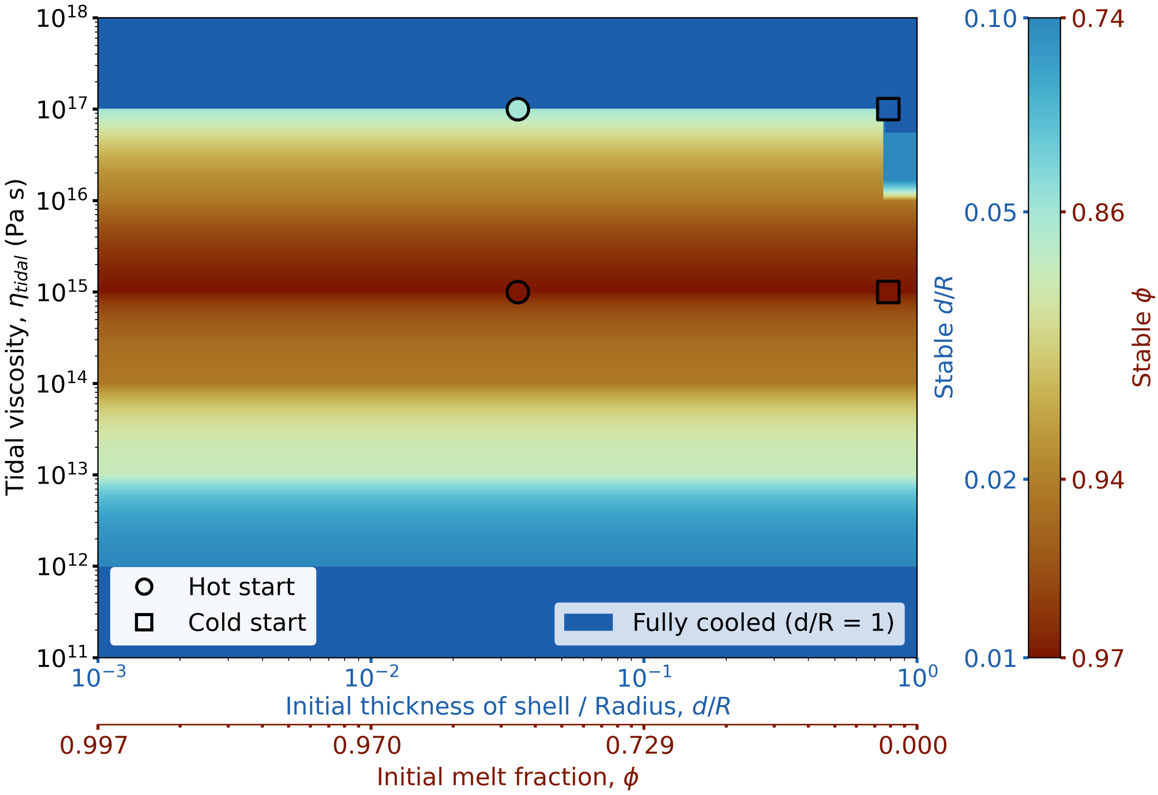

Figure 3.6: Final melt fractions: as a function of tidal viscosity, η_{tidal}, and normalised shell thickness d/R (or melt fraction, ϕ). For chosen convective viscosity $\eta_{\text{conv}} = 10^{21}$. Heat evolutions ran from hot starts (circles) and cold starts (squares) are shown on top.

Iterating Equation (3.1) successfully reaches the steady states hypothesised from evaluating the intersecting fluxes, for both hot and cold starts, as shown in Figure (3.6).

4 Discussion

The model's key findings are as follows: tidal viscosity is the primary control on both tidal dissipation rates and the equilibrium melt fraction. Convective viscosity is also important as it constrains the range of tidal viscosities that can sustain a molten interior. For convective viscosities in the range $\eta_{\mathrm{conv}} = 10^{17} - 10^{25}$ Pa s, magma oceans can exist at equilibrium for noticeably low tidal viscosities within the range $\eta_{\mathrm{tidal}} = 10^{10} - 10^{18}$ Pa s, at melt fractions of $\phi_s = 0.14 - 0.97$. In most cases, initial conditions have a minimal impact on the final state, and tidal heating can sustain molten interiors.

4.1 Model limitations

A primary limitation of the model is the assumption of spherical symmetry. In reality, tidal potential varies with radius, latitude, and longitude. It reaches a maximum where the near-side (or far-side) meridian intersects the equator, i.e. at points that align with the axis connecting the centres of mass of the moon and the central planet (Murray and Dermott 2000). I evaluate the maximum stress experienced at the same points, and integrate this over the shell to estimate the total tidal heat flux. This method likely overestimates the actual global dissipation, as it assumes uniform maximum stress across the entire shell. Alternatively, I could consider volumetric fluxes at these points, but the result would still represent an upper bound on the true tidal heating.

The model does not account for tidal dissipation within the basal magma ocean. While one might expect dissipation in a fluid to be minimal due to reduced resistance, studies suggest that it could contribute a significant amount of heat. Tyler, Henning and Hamilton (2015) find that not only can Io's heat flux be replicated by fluid-generated tides, but solutions exist for a wider parameter space than purely solid-generated tides.

Another simplification is the exclusion of higher-order deformations beyond the primary, rugby-ball-like deformation. In reality, for an eccentric, synchronous orbit, the moon constantly faces the empty focus of the ellipse. Meanwhile, the tidal bulge will point towards the central planet. This misalignment causes the bulge to oscillate across the surface, generating an additional, but lesser, dissipation through librational tides (Murray and Dermott 2000).

Other orbital variations, such as obliquity and precession, can also modulate the tidal potential over time, introducing further temporal complexity not captured in the model. Additionally, I have neglected solar tides, which can amplify deformation. However, these are all weaker than the impact of an eccentric orbit on Io's tides.

In the heat evolution model, the assumption of spherical symmetry also limits the ability to capture lateral variations in heating. I consider viscosity to be spatially uniform. However, the viscosity of silicates is highly sensitive to temperature and melt fraction. I assume that any melt is immediately transferred to the underlying magma ocean, so the solid shell has a melt fraction of zero everywhere. While this simplifies the treatment, assuming a purely solid, zero-porosity shell removes the temperature-viscosity feedback loops in tidal dissipation (Moore 2003). As discussed throughout, viscosity is a key control. Tidal heating raises temperatures, reducing viscosity, which in turn alters the efficiency of tidal dissipation. Depending on whether the system lies above or below the peak viscosity, a reduction in viscosity may either suppress or enhance dissipation. This non-linear feedback is important for understanding the thermal and dynamic evolution of the system, but is not captured in the model. Spatial heterogeneity in heating can also lead to varied convective behaviour, including localised melting, plume formation and volcanism, and slab dripping, which is not represented.

The model also assumes that melt and solid phases have the same composition. However, during melting, incompatible elements tend to partition into the melt phases over time. For example, in systems dominated by iron silicates, iron preferentially enters the melt, increasing its density. This chemical differentiation affects buoyancy, and could lead to stratification (Boukaré, Badro and Samuel 2025).

Finally, the model assumes that melt is denser than the solid shell and sinks to form a basal magma ocean. Although this does not limit the model in general, it may not fully capture the behaviour of Io. On Io, some melt must rise to the surface during eruptions, suggesting that at least part of the melt is buoyant or sourced from a separate reservoir. To better represent Io, the model could be adapted to include this complexity.

4.1.1 Viscosities

I assume a Maxwell viscoelastic rheology, as it is the simplest rheology that captures both elastic and viscous deformation. The model finds that viscosity is the primary control on tidal dissipation. To generate sufficient tidal heating, viscosities must fall below 10^{18} Pa s, which is consistent with findings from previous studies (Moore 2003; Tyler, Henning and Hamilton 2015).

However, these viscosities are somewhat unrealistic. First, they are at least three orders of magnitude lower than typical silicate mantle values inferred from Earth observations (Lambeck, Johnston and Nakada 1990). Second, such low viscosities generally imply melt fractions that exceed 50%. In partially molten silicate systems, compaction processes would likely cause melt to rapidly segregate from the solid matrix, making it difficult to sustain such high melt fractions, thus leading to higher viscosities (Bierson and Nimmo 2016).

In response to this issue, some studies have adopted more complex rheologies, such as the Burgers or Andrade models, which modify how elasticity and viscosity interact. The Andrade rheology, for example, is based on laboratory experiments (Renaud and Henning 2018) and includes the contribution of transient creep. Initially, the material responds elastically, followed by both transient creep and viscous relaxation (Walterová et al. 2023). As a consequence, it allows for greater tidal dissipation at higher, more realistic viscosities (Bierson and Nimmo 2016).

Nonetheless, the viscosity range I adopt ($10^{10} - 10^{18}$ Pa s) remains consistent with earlier thermal evolution models for tidally heated silicate bodies near the solidus (Moore 2003; Fischer and Spohn 1990; Tyler, Henning and Hamilton 2015). Furthermore, the lack of detailed constraints on the interior structure of Io justifies the use of simple, first-order models. These help identify broad parameter space trends and key feedbacks, even if they cannot capture the full complexity of real planetary interiors.

4.2 Equilibrium conditions

The model predicts a range of stable shell thicknesses, depending on the assumed viscosities for both tidal dissipation and convection. Convective viscosities comparable to the Earth's mantle ($\eta \approx 10^{21}$Pa s, Lambeck, Johnston and Nakada 1990) yield stable normalised shell thicknesses in the range $d/R = 0.011$ to 0.097, corresponding to physical shell thicknesses on the order of $10 - 100$ km. These correlate to melt fractions of $\phi = 0.736$ to 0.968.

Observations of Io indicate a crustal thickness of 30 to 50 km; an asthenosphere of 50 to 100 km, with a melt fraction $\geq 20\%$; and a metallic core of 900 to 1000 km, which leaves a mantle thickness of 800 to 900 km (Khurana et al. 2011). The model's shell thickness, for a realistic convective viscosity, is much lower than the observed value, as it represents the entire solid mantle. However, it is important to note that Io also possesses a metallic core, which would affect the planet's radial structure and heat budget, and is not accounted for in this simplified model. The model also overestimates tidal heating across the body, so these melt fractions represent an upper bound.

Across the full parameter space I explored, the range of stable normalised shell thicknesses extends from $d/R = 0.011$, for low tidal and convective viscosities ($\eta_{\text{diss}} = 10^{15}$, $\eta_{\text{conv}} = 10^{19}$Pa s) to as high as $d/R = 0.484$ for extremely low tidal viscosity ($\eta_{\text{diss}} = 10^{10}$ Pa s) and high convective viscosity ($\eta_{\text{conv}} = 10^{25}$ Pa s). The latter represents a physical shell thickness of ≈ 870 km and is more in line with observed values. These results illustrate the sensitivity of equilibrium conditions to rheological parameters and show that tidal heating in the presence of a basal magma ocean yields high melt fractions.

4.3 Initial conditions

The formation of the Galilean moons, including Io, is intrinsically linked to the formation of Jupiter itself. Current models suggest that the moons formed after Jupiter's hydrodynamic collapse phase, in a circumplanetary disk fed by material from the solar nebula. The disk likely exhibited a velocity gradient, with gas closer to Jupiter orbiting more rapidly than gas further out. This introduces shear and, if the disk is viscous, can lead to dissipation and turbulence (McKinnon 2023).

As a result, Io's initial thermal state would have been set by the thermal conditions within this circumjovian disk. These conditions are governed by a balance between radiative cooling and various heating mechanisms - Jupiter's luminosity, the kinetic energy of infalling material, viscous dissipation within the disk, and external heating from the background protosolar nebula. However, the relative importance of these heat sources is difficult to constrain. For example, viscous dissipation is highly sensitive to assumptions about disk viscosity and structure (McKinnon 2023).

Accretional heating of Io adds further uncertainty. The total heat gained during accretion depends on both the timescale of accretion and the size distribution of accreted material. For instance, assuming a background temperature of 200 K and accretion from small planetesimals ($10 - 100$ m diameter), the resulting temperature of Io could be less than 300 K if accretion occurred over 10^6 yrs, but could reach up to 1400 K if accreted in just 10^3 yrs, (McKinnon 2007).

The model results indicate a slight sensitivity to initial conditions. The long-term stability of a basal magma ocean requires a pre-existing melt fraction, dependent on viscosities η_{diss}, η_{conv}. For more realistic viscosities ($\eta_{\text{conv}} = 10^{21}$, $\eta_{\text{tidal}} = 10^{17}$ Pa s) this melt fraction is $\phi_h = 0.016$ which suggests Io must have formed under slightly warm conditions, at least hot enough to achieve the minimum melt fraction required for sustaining a partially molten shell. This requirement may help place a lower bound on Io's initial thermal state. However, since $\phi_h = 0$ in most scenarios, this inference is made hesitantly.

4.4 Observations, Io's interior, and magma oceans

Spacecraft missions have provided remote observations of Io, allowing broad inferences to be made about its interior. Doppler measurements from *Galileo* confirmed that Io is differentiated, likely possessing an iron or iron-sulphide metallic core (Anderson, Sjogren and Schubert 1996). Later, magnetometer data suggested the presence of a global conductive layer, interpreted as a subsurface magma ocean or partially molten asthenosphere (Khurana et al. 2011). Indeed, Io generates roughly 100 TW of heat, enough to plausibly sustain such a layer (Park et al. 2025). The global distribution of volcanism, as mapped by *Juno*, also supports the idea of a largely molten interior (Davies et al. 2023). A global magma ocean would mechanically decouple the lithosphere from the deeper interior, enhancing strain and dissipation in the overlying solid shell (Miyazaki and Stevenson 2022), and may also be significantly tidal itself (Tyler, Henning and Hamilton 2015).

Stronger constraints come from measurements of tidal Love numbers, particularly k_2 (Bierson and Nimmo 2016). The real part, $\mathrm{Re}(k_2)$, reflects the elastic (in-phase) deformation, the ratio of the "self-gravity" response to tidal forcing. The imaginary part, $\mathrm{Im}(k_2)$, reflects the viscous (out-of-phase) dissipation. The dissipation factor, Q, is inversely proportional to the energy dissipated per tidal cycle and can be inferred from k_2 through the relation $\mathrm{Im}(k_2) = -|k_2|/Q$ (Park et al. 2025; Auclair-Desrotour, Poncin-Lafitte and Mathis 2014).

The *Juno* mission has recently measured Io's gravitational field with sufficient precision to determine its tidal response. Combining the data of *Juno* and *Galileo*, Park et al. (2025) estimated $\mathrm{Re}(k_2) = 0.125 \pm 0.047(1\sigma)$ and $-\mathrm{Im}(k_2) = 0.0109 \pm 0.0054(1\sigma)$. These values suggest that a global magma ocean is unlikely, as it would produce a much larger tidal response.

The model supports this conclusion. I computed k_2 analytically across a range of tidal viscosities ($\eta_{\mathrm{diss}} = 10^{13}, 10^{15}, 10^{17}$ Pa s) at their respective equilibrium shell thicknesses, assuming a convective viscosity $\eta_{\mathrm{conv}} = 10^{21}$ Pa s, see Figure (4.1). In all cases, the model $\mathrm{Re}(k_2)$ exceeds the observed value, meaning the modelled body deforms more under tidal forcing than Io does. This implies that Io is less responsive, consistent with a thicker, more rigid shell and no basal magma ocean. Conversely, in most scenarios, the model $\mathrm{Im}(k_2)$ is

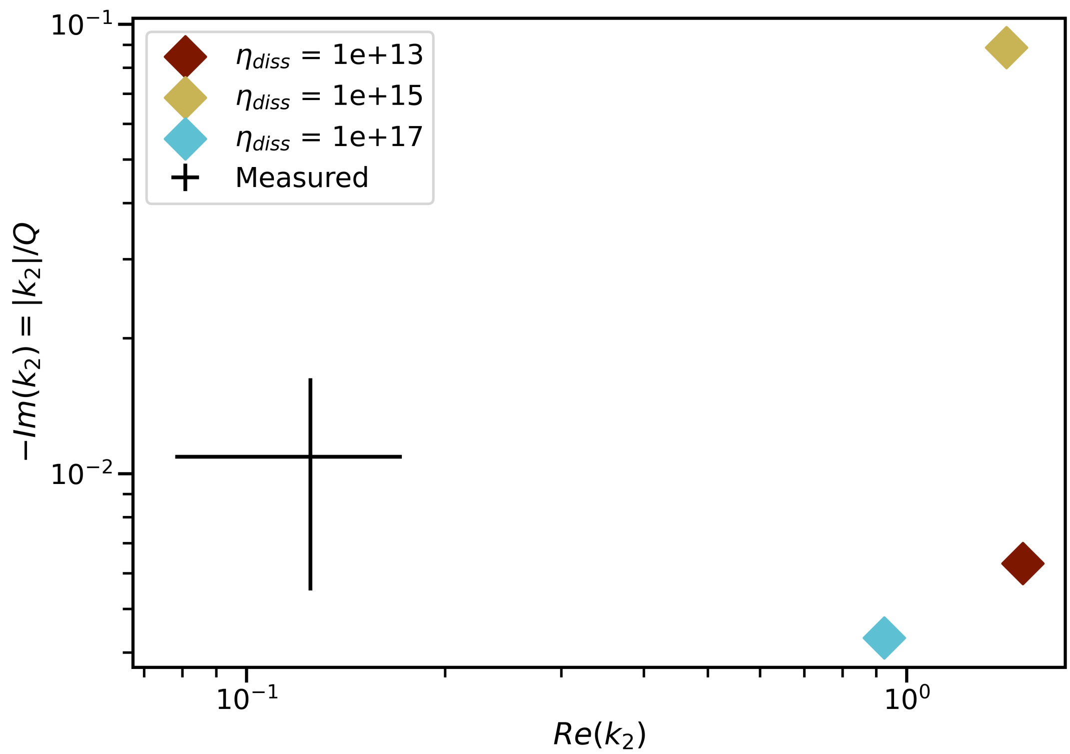

Figure 4.1: $\mathbf{k_2}$: Plot of model $-\mathrm{Im}(k_2), \mathrm{Re}(k_2)$ for chosen tidal viscosities $\eta_{\mathrm{diss}} = 10^{13}, 10^{15}, 10^{17}$ Pa s, and convective viscosity $\eta_{\mathrm{conv}} = 10^{21}$ Pa s. The measured value for Io (Park et al. 2025) is plotted in black, with error bars.

lower than the observed values, indicating that Io dissipates more heat than predicted, except for $\eta_{\mathrm{diss}} = 10^{15}$ Pa s where the model dissipation peaks and exceeds observations.

In summary, Io likely has a subdued deformation compared to the model predictions, but dissipates more heat, indicative of a more viscous response. However, the model k_2 represents the local maximum displacement, whereas the observed value is a global average. Thus, higher elastic deformation in the model is expected. Additionally, the model assumes Io is in thermal equilibrium. Increasing the shell thickness would reduce the amplitude of tidal displacement and could also alter the viscous dissipation behaviour.

Aygün and Čadek (2024b) further confirms this through the alternative approach of calculating harmonic-dependent k_2 values. They calculate k_2 for the spherical harmonics degree $l = 2$ and orders $m = 0, 2$ - $k_{2,0}$, $k^c_{2,2}$, and $k^s_{2,2}$, respectively - that represent the dominant, rugby-ball-like deformation. Park et al. (2025) equated these values, which in reality only holds for solid-body tides. Aygün and Čadek (2024b) found that although shallow magma ocean models can satisfy the observed k_2 values, they overestimate the time lags, ultimately reaching the same conclusion that Io is unlikely to have a shallow magma ocean. However, they do not rule out a basal magma ocean and reiterate the need for more precise k_2 measurements.

But, if Io generates so much internal heat, why doesn't it have a basal magma ocean?

One key factor is density. To form a stable basal magma ocean, the melt must be denser than the surrounding solid. On large bodies like Earth, high internal pressures can cause silicate melt to become denser than solid phases at depth. But Io, being much smaller, $< 30\%$ Earth's size, does not reach such pressures (Pommier and McEwen 2022; Mosenfelder, Asimow and Ahrens 2007).

Basal magma oceans can also form during the solidification of a planet through chemical stratification, regardless of the depth of this density crossover (Boukaré, Badro and Samuel 2025). Iron-rich melts can be denser than their solid counterparts even at lower pressures, and remain stable at depth. However, this creates another issue as this iron-rich melt would not easily rise and erupt. Io's surface is covered with active volcanoes, suggesting that at least some melts are buoyant enough to reach the surface. One possibility is the presence of multiple melt reservoirs: an iron-rich layer at depth, and a more buoyant, silicate-rich magma that drives surface volcanism. Yet, given the lack of direct constraints, the simplest explanation is that melt generally rises, and that Io lacks a stable basal magma ocean (Park et al. 2025).

4.5 Other planetary bodies

Although Io inspired the basal magma ocean model, current evidence suggests it is unlikely to host one. However, the model still demonstrates that tidal heating can sustain molten interiors, and it may apply to other planetary bodies.

For example, basal magma oceans may be more viable on icy moons, where the liquid phase (water) is usually denser than the solid (ice), allowing melt to remain stable at depth. Europa and Enceladus are key examples in our solar system.

Such oceans may also be possible on larger rocky planets, where deep silicate melts can become denser than surrounding solids, like on Earth. While no known rocky bodies in our solar system meet the conditions for our model - Io is too small, while Earth and Venus are not tidally heated in their rocky layers - a growing number of Earth-mass (and larger) exoplanets are being discovered. If they experience sufficient tidal heating, these planets could potentially host long-lived magma oceans.

4.5.1 Icy Moons

Europa

Europa is another Galilean moon under the strong tidal influence of Jupiter. It is the second closest to Jupiter and is in a 2:1 orbital resonance with Io. Unlike Io, Europa is an icy moon, with its silicate interior hidden beneath an outer shell of ice and liquid water. A global subsurface ocean was first proposed to explain Europa's surface features and was later supported by gravity and magnetometer measurements from *Galileo* (Khurana et al. 1998).

The thickness of Europa's ice shell and ocean remains poorly constrained. Interior models typically predict an ice shell with a $\approx 7\,\mathrm{km}$ upper conductive layer and a $\approx 13\,\mathrm{km}$ lower convective layer, overlying a global, liquid water ocean (Wakita et al. 2024). However, recent unpublished results from the *Juno* mission suggest the conductive ice shell could be as thick as 35 km, up to five times greater than previous estimates (Levin et al. 2024; Voosen 2024). This would imply a significantly thinner liquid ocean than previously thought.

The model of equilibrium shell thickness under tidal heating is relevant here. Like in the model, Europa's shell thickness likely depends on the balance between internal heating and heat transport through conduction or convection. Tidal dissipation within the icy shell produces meltwater that sinks into the liquid ocean (Roberts et al. 2023). The upcoming *Europa Clipper* mission is expected to provide constraints on both the shell and ocean thicknesses through magnetometry and gravity measurements. Applying our model to Europa-like parameters could help predict shell thicknesses and may reveal whether Europa is currently in thermal equilibrium.

Understanding Europa's internal dynamics, in particular, is also interesting due to its impact on habitability. Subsurface oceans may host environments favourable to prebiotic chemistry, especially where water and rock interact via conduits of hydrothermal activity. Maintaining a liquid ocean requires sustained internal heat, which could be generated through tidal dissipation. Investigating the behaviour of tides on icy moons, such as Europa, may help determine whether they could feasibly sustain extraterrestrial life.

Enceladus

Enceladus is another icy moon, which orbits Saturn. In 2006, the *Cassini* spacecraft discovered active plumes venting water vapour and fine ice particles from its South Pole (Porco et al. 2006), which was the first indicator of significant tidal heating. More recent data have revealed the presence of a $\lessapprox$ 30km thick conductive ice shell, a $\lessapprox$ 26km thick global subsurface ocean, and a low-density rocky core (Park et al. 2024).

The tidal processes on Enceladus are more unclear than on Europa, due to the limited observational constraints. Enceladus is in a 2:1 orbital resonance with Dione, which is also thought to fix its orbital eccentricity, but this is not certain. Additionally, whilst heating may occur in the ice shell or ocean, as on Europa, some models suggest it may instead concentrate in Enceladus' low-density core (Nimmo, Neveu and Howett 2023). My model could also be applied to Enceladus to estimate potential shell thicknesses in the absence of detailed observations.

4.5.2 Exoplanets

The first exoplanet was discovered in 1992, through changes in pulsar timing (Wolszczan and Frail 1992). As of 21$^{\mathrm{st}}$ April 2025, we have identified $5,876$ exoplanets (*NASA Exoplanet Archive* 2025). Many of these planets are candidates for significant tidal heating, as they possess both sufficient orbital eccentricity and are located close enough to massive bodies to experience large tidal forces (Henning and Hurford 2014).

An example is the TRAPPIST-1 system, which contains seven closely packed terrestrial planets. This system is particularly interesting because the planets' tides are raised not only by their host star, but also by each other due to their proximity and relative sizes (Hay and Matsuyama 2019). The inner six planets form a resonant chain, which forces eccentricity, and thus the $4^{\mathrm{th}} - 7^{\mathrm{th}}$ planets (from the star) may have conditions suitable to host liquid water. (Papaloizou, Szuszkiewicz and Terquem 2018).

Tidal heating is not only a mechanism for sustaining subsurface oceans and volcanic activity, and thus potentially life; it also plays a major role in shaping orbital evolution (Henning and Hurford 2014). Many of these candidate exoplanets do not have fixed eccentricities, and their thermal states may evolve with their orbits. In such cases, my model could be integrated into orbital evolution frameworks to explore how molten interiors evolve through time. Given the limited observational constraints and the many trade-offs involved in characterising exoplanets (Henning and Hurford 2014), simple models with a few parameters are especially valuable.

5 Conclusion

I aimed to develop a 0-D model that simulates tidal dissipation due to orbital eccentricity, to assess how a basal magma ocean affects the thermal evolution of tidally heated bodies. The model produces analytical solutions for the tidal Love numbers, h_2 and k_2, which closely match those of the 3-D model (Love 1909), demonstrating that it captures the main features of tidal deformation. Its parameter dependence, including mean motion and Maxwell frequency, is consistent with previous studies (Hay and Matsuyama 2019), providing further validation. Overall, the model effectively reduces the 3-D problem to 0-D and provides a simplified framework for future research.

The key finding is that tidal viscosity is the main control on both tidal dissipation rate and the overall thermal balance, including the resulting equilibrium states. Maximum heating and the most molten equilibrium states occur at intermediate viscosities—high enough to generate friction through deformation, but not so high as to prevent deformation altogether. Convective viscosity plays a secondary role, limiting the range of tidal viscosities that can sustain a non-zero equilibrium melt fraction. The initial melt fraction can also be important, as in some cases it must exceed a threshold to maintain a molten interior over long timescales.

A 0-D model was chosen to isolate the key controls on tidal dissipation and thermal evolution, given the limited observational constraints for Io. The model includes three gravitational potentials—tide-raising, spherical, and response—and assumes a Maxwell viscoelastic rheology. The results suggest these assumptions are sufficient to capture the essential behaviour. While a further simplified model using only the tide-raising potential was considered, self-gravity was found to significantly influence the deformational response and was therefore necessary.

A compelling next step would be to incorporate a more complex rheology, such as the Andrade model which may better capture the response of a body to tides (Bierson and Nimmo 2016). The inclusion of self-gravity already introduces deviations from the simple Maxwell behaviour, so the model may be better suited to an Andrade rheology, although this would add more complexity. Future work could also explore alternative internal structures, such as a surficial magma ocean or the inclusion of a core, to investigate their influence on tidal heating.

The results align with recent studies that suggest Io does not host a global magma ocean, based on interpretations of measured k_2 values (Park et al. 2025; Aygün and Čadek 2024a), supporting a growing consensus surrounding Io's interior structure. The model has some implications for Io's initial conditions, potentially indicating that it formed in a sufficiently hot region of Jupiter's circumplanetary disc, or retained enough primordial or early radiogenic heat, to generate an initial melt fraction.

While developed with Io in mind, the model is likely more applicable to icy bodies with subsurface oceans, such as Europa or Enceladus, and will be useful in light of incoming observations from the *Europa Clipper*. Applying the model to tidally active exoplanets, as they continue to be discovered, could provide critical insight into their thermal states and potential habitability.

In summary, as more precise measurements and constraints on planetary internal structures become available, for both our solar system and exoplanets, this model provides a foundation for quickly assessing the potential for tidal heating and guiding more complex layered models.

Appendices

A

Derivations

A.1 3-D to 0-D

I begin with the conservation of momentum:

$$\nabla \cdot \sigma = \rho \nabla U^T + \rho \nabla U^G + \rho \nabla U^R \tag{A.1}$$

To simplify the problem in 0-D, I first define a direction, $\hat{X}$, that runs between the centres of mass for the central body and the satellite, and thus between the nearest and furthest points.

To find the potential at these points, I consider the variation in $\hat{X}$.

$$\begin{gathered}\hat{X} \cdot [\nabla \cdot \sigma] = \hat{X} \cdot [\rho \nabla U^T + \rho \nabla U^G + \rho \nabla U^R] \\ \frac{\partial}{\partial X} \sigma_{XX} = \rho \left(\frac{\partial U^T}{\partial X} + \frac{\partial U^G}{\partial X} + \frac{\partial U^R}{\partial X} \right)\end{gathered} \tag{A.2}$$

This is parallel to the radial direction on the far-side of the moon, and equal but opposite on the near-side.

For the far-side, $\hat{r} = \hat{X}$

$$\frac{\partial}{\partial r}\sigma_{rr} = \rho\left(\frac{\partial U^T}{\partial r} + \frac{\partial U^G}{\partial r} + \frac{\partial U^R}{\partial r}\right)$$

$$\frac{\Delta}{\Delta r}\sigma_{rr} = \rho\left(\frac{\Delta U^T}{\Delta r} + \frac{\Delta U^G}{\Delta r} + \frac{\Delta U^R}{\Delta r}\right) \tag{A.3}$$

$$\Delta\sigma_{rr} = \rho\left(\Delta U^T + \Delta U^G + \Delta U^R\right)$$

And for the nearside, $\hat{r} = -\hat{X}$

$$-\frac{\partial}{\partial r}\sigma_{rr} = -\rho\left(\frac{\partial U^T}{\partial r} + \frac{\partial U^G}{\partial r} + \frac{\partial U^R}{\partial r}\right)$$

$$\frac{\Delta}{\Delta r}\sigma_{rr} = \rho\left(\frac{\Delta U^T}{\Delta r} + \frac{\Delta U^G}{\Delta r} + \frac{\Delta U^R}{\Delta r}\right) \tag{A.4}$$

$$\Delta\sigma_{rr} = \rho\left(\Delta U^T + \Delta U^G + \Delta U^R\right)$$

which shows the change in stress is the same at both the nearest and furthest points.

A.2 Potentials

A.2.1 Tidal potential

The full equation for tidal potential due to an eccentric orbit is:

$$U^T_{\mathrm{ecc}} = \Omega^2 r^2 e\left[-\frac{3}{2}P_{20}(\cos\theta)\cos M + \frac{1}{8}P_{22}(\cos\theta)\left(7\cos(2\phi - M) - \cos(2\phi + M)\right)\right] \tag{A.5}$$

where θ is colatitude, measured from the spin axis of the satellite, and ϕ is longitude. I evaluate the potential at $\theta = \phi/2, \phi = 0$ to find the maximum stress.

A.2.2 Response potential

The response potential is derived as in (Norsen, Dreese and West 2017).

First, Gauss' Law of gravitation:

$$\nabla^2 U^R = 4\pi G \xi \tag{A.6}$$

where ξ is the surface mass density, $\xi = \rho \Delta R$, and$\nabla^2 U^R = -g_r$.

The azimuthally symmetric solutions, in spherical coordinates, for order $l = 2$:

$$\begin{gathered} U^R = \begin{bmatrix} Ar^2 & r < R \\ Br^{-3} & r > R \end{bmatrix} \\ \frac{\partial U^R}{\partial r} = \begin{bmatrix} 2Ar & r < R \\ -3Br^{-4} & r > R \end{bmatrix} \end{gathered} \tag{A.7}$$

Equating Equations (A.6)-(A.7), at $r = R$, yields:

$$U^R = -\frac{3}{5} g_R R \epsilon \tag{A.8}$$

A.3 Optimal shell thickness

The tidal dissipation rate, modified for a basal magma ocean:

$$\dot{E}_V = \frac{d}{R} n \tilde{\mu}_{\text{Im}} \frac{(3\Omega^2 e \rho R^2)^2}{2\left(\frac{d}{R}\tilde{\mu} + \frac{2}{5}\rho g_R R\right)\left(\frac{d}{R}\tilde{\mu}^* + \frac{2}{5}\rho g_R R\right)} \tag{A.9}$$

To find the peak dissipation, as a function of shell thickness, first differentiate with respect to shell thickness:

$$\frac{\mathbf{d}\dot{E}_V}{\mathbf{d}d} = A\tilde{\mu}_{\text{Im}}/R \frac{(\rho g_R R)^2 - [(\frac{\tilde{\mu}_{\text{Re}}}{R})^2 + \frac{\tilde{\mu}_{\text{Im}}}{R})^2]d^2}{\left((\rho g_R R)^2 + 2\rho g_R R(\frac{\tilde{\mu}_{\text{Re}}}{R})d + [(\frac{\tilde{\mu}_{\text{Re}}}{R})^2 + (\frac{\tilde{\mu}_{\text{Im}}}{R})^2]d^2\right)^2} \tag{A.10}$$

where A is a group of constants $\frac{n}{2}(3\Omega^2 e \rho R^2)^2$.

The maximum point is where the gradient is zero:

$$d_{max} = \sqrt{\frac{(\rho g_R R)^2}{(\frac{\tilde{\mu}_{\mathrm{Re}}}{R})^2 + (\frac{\tilde{\mu}_{\mathrm{Im}}}{R})^2}} \tag{A.11}$$

which rearranges to a normalised shell thickness:

$$\frac{d_{\mathrm{max}}}{R} = \frac{\rho g_R R}{|\mu|} \tag{A.12}$$

A.4 Timescales of dissipation and convection

The velocity of a sinking parcel is:

$$U \propto \frac{g\rho\alpha\Delta T r^2}{\eta} \tag{A.13}$$

where g is gravitational acceleration, ρ is density, ΔT is the temperature difference across the layer, r is the radius of the parcel, and η is the viscosity.

The timescale for the parcel to sink:

$$\tau = \frac{h}{W} \tag{A.14}$$

where h is the thickness of the layer. Evaluating with our chosen parameters:

$$\tau \approx \frac{\eta}{10^6} h \tag{A.15}$$

For viscosities $\eta_{\mathrm{conv}} = 10^{17} - 10^{25}$Pa s, and shell thicknesses on the order of $10 - 100\,\mathrm{km}$, is $\tau = 10^{15} - 10^{24}$s.

B

Heat Evolution Model

B.1 Heat evolution

```
#Import necessary libraries
import numpy as np
import math
import os
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
now = datetime.now()

#Import functions
from fn_delta_phi import delta_phi
from heat_evolution.fn_fluxes_phi import shell_diss, shell_out
from fn_properties import melt_layer

#Import parameters
from param.parameters_io import R
from param.parameters_rheo import Ts, Tml, k
from param.parameters_uni import pi

#Conversion parameters
yrs_to_s = 365 * 24 * 3600 # for time


```

```
def heat_evolution(phii, eta_tidal, eta_conv, scale, t_end=4.5e9
    , dt=100):
    """
    Function to run the heat evolution model.

    Parameters
    ----------
    phii : float
        Initial melt fraction
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scaling of the complex shear modulus ('True', 'False')
    t_end : float
        End time of the simulation [yrs]
    dt : float
        Time step of the simulation [yrs]

    Returns
    -------
    t : list
        Time [yrs]
    phi : list
        Melt fraction
    q_tidal : list
        Tidal dissipation [W]
    q_out : list
        Outward flux [W]
    date : str
        Date of the simulation
    d : list
        Thickness of the shell [m]
    Ra : list
        Rayleigh number of the shell
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : float
        Scaling of the complex shear modulus ('True', 'False')
    """
    ###########Starting time###########
    date = now.strftime("%d/%m/%Y %H:%M:%S")
```

```python

    ###########Unpack Time###########
    #Convert the times from years to seconds
    t_end = t_end * yrs_to_s #s, End time
    dt = dt * yrs_to_s #s, Time step

    #Check the time step is stable
    conductive_time = R**2 / (k * (Tml-Ts)) #s, Conductive time scale, need > 1/10th
    if dt <= conductive_time / 10:
        print('Error: Chosen time step may be unstable')

    #Construct the time list to run over
    iterations = int(t_end / dt) #Number of iterations
    t_run = np.linspace(0, t_end, iterations) #s, Time to run over

    #Set the checking & saving frequencies
    save_freq = max(1, iterations // 1e5) #When to save values
    print_freq = max(1, iterations // 1e2) #When to print values

    ###########Initial values###########
    #Initial fluxes
    q_tidal_i , *_ = shell_diss(phii, eta_tidal, scale)
    q_out_i , _, Ra_i = shell_out(phii, eta_tidal, eta_conv, scale)
    _, di = melt_layer(phii)

    #Store initial values
    t = [0] #yrs, Time
    phi = [phii] #Melt fraction
    q_tidal = [q_tidal_i] #W, Tidal dissipation
    q_out = [q_out_i] #W, Outward flux
    d = [di] #m, Thickness of shell
    Ra = [Ra_i]# Rayleigh number


    ###########Time loop###########
    for i, ti in enumerate(t_run[1:]): # For every point but the first
        #Update values
        dphi, q_tidal_i, _, q_out_i, _, Ra, d, _ = delta_phi(phii, eta_tidal, eta_conv, scale, dt)
        phii += dphi

```

```python
        #Check shell volume
        if phii > 1:
            phii = 1 #Cannot melt more than 100%

        if phii < 0:
            phii = 0 #Cannot solidify more than 0%

        #Save values
        if i % save_freq == 0:
            t.append(ti / yrs_to_s) #yrs, Time
            phi.append(phii) #Melt fraction
            q_tidal.append(q_tidal_i) #W, Tidal dissipation
            q_out.append(q_out_i) #W, Outward flux
            d.append(di) #m, Thinness of shell
            Ra.append(Ra_i) #Rayleigh number

        #Print values
        if i % print_freq == 0:
            print(f'Time: {ti / yrs_to_s:.2e} yrs, Melt fraction: {phii:.2e}, Tidal dissipation: {q_tidal_i:.2e} W, Outward flux: {q_out_i:.2e} W, Thickness: {di:.2e}, Rayleigh number: {Ra_i:.2e}')

    return t, phi, q_tidal, q_out, date, d, Ra, eta_tidal, eta_conv, scale

def save_evolution(t, phi, q_tidal, q_out, date, d, Ra, eta_tidal, eta_conv, scale, folder):
    #Create/find directories
    os.makedirs('runs', exist_ok=True)
    os.makedirs(f'runs/{folder}', exist_ok=True)
    os.makedirs(f'runs/{folder}/data', exist_ok=True) # For lists

    #Find run number
    try:
        prev_data = pd.read_csv(f'runs/{folder}/Heat_Evolution.csv')
        run = prev_data['Run'].iloc[-1] + 1
    except FileNotFoundError:
        run = 1

    #Save data
    run_data = pd.DataFrame({
        'Run': [run],
```

```
            'Date': [date],
            'Initial Melt Fraction': [float(phi[0])],
            'Final Melt Fraction': [float(phi[-1])],
            'Time': [float(t[-1])],
            'eta_tidal': [float(eta_tidal)],
            'eta_conv': [float(eta_conv)],
            'scale': [scale]
    })
    try:
        prev_data = pd.read_csv(f'runs/{folder}/Heat_Evolution.csv')
        run_data = pd.concat([prev_data, run_data], ignore_index=True)
    except FileNotFoundError:
        pass

    run_data.to_csv(f'runs/{folder}/Heat_Evolution.csv', index=False)

    #Save arrays in compressed npz
    np.savez_compressed(f'runs/{folder}/data/Heat_Evolution_{run}.npz', t=t, phi=phi, q_tidal=q_tidal, q_out=q_out, d=d, Ra = Ra)

    return run

def plot_evol(folder, run):
    #Load data
    df = pd.read_csv(f'runs/{folder}/Heat_Evolution.csv')
    initial_melt_fraction = df.loc[df['Run'] == run, 'Initial Melt Fraction'].values[0]
    eta_tidal = df.loc[df['Run'] == run, 'eta_tidal'].values[0]
    eta_conv = df.loc[df['Run'] == run, 'eta_conv'].values[0]
    scale = df.loc[df['Run'] == run, 'scale'].values[0]

    #Load arrays
    data = np.load(f'runs/{folder}/data/Heat_Evolution_{run}.npz')
    t, phi, q_tidal, q_out, d, Ra = data['t'], data['phi'], data['q_tidal'], data['q_out'], data['d'], data['Ra']

    #Plot
    fig, axs = plt.subplots(2, 3, sharex = True, figsize=(10, 10))
```

```
    fig.suptitle(f'Heat Evolution for Run {run}, Initial Melt Fraction: {initial_melt_fraction}, scale = {scale} \n eta_tidal: 1e{int(math.log10(eta_tidal))}, eta_conv: 1e{int(math.log10(eta_conv))}')

    #Top row
    axs[0,0].plot(t, phi, label='Melt Fraction')
    axs[0,0].set_title('Melt Fraction, $\phi$')

    axs[0,1].plot(t, d/R, label='Shell Thickness')
    axs[0,1].set_title('Normalised shell Thickness, $d/R')

    axs[0,2].plot(t, Ra, label='Rayleigh Number')
    axs[0,2].set_title('Rayleigh Number')

    #Bottom row
    power  = [q_tidal_i - q_out_i for (q_tidal_i, q_out_i) in zip(q_tidal, q_out)]
    axs[1,0].plot(t, power, label='Total Power')
    axs[1,0].set_title('Total Power (W)')

    axs[1,1].plot(t, q_tidal, label='Tidal Dissipation')
    axs[1,1].set_title('Tidal Dissipation (W)')

    axs[1,2].plot(t, q_out, label='Outward Flux')
    axs[1,2].set_title('Outward Flux (W)')

    #Axes
    axs[0,2].yaxis.tick_right()
    axs[0,2].yaxis.set_label_position("right")

    axs[1,2].yaxis.tick_right()
    axs[1,2].yaxis.set_label_position("right")

    axs[0,0].set_xlabel('Time (yrs)')
    axs[0,1].set_xlabel('Time (yrs)')
    axs[0,0].set_xscale('log')
    axs[0,1].set_xscale('log')

    #Save
    os.makedirs(f'runs/{folder}/plots', exist_ok=True)
    plt.savefig(f'runs/{folder}/plots/Heat_Evolution_{run}.png')

    return
```

B.2 Delta phi

```
#Calculate change in phi for heat evolution.

#Import functions
from heat_evolution.fn_fluxes_phi import shell_diss, shell_out
from fn_properties import melt_layer

#Import parameters
from param.parameters_rheo import rhoms, L
from param.parameters_io import R
from param.parameters_uni import pi

def delta_phi(phi, eta_tidal, eta_conv, scale, dt):
    '''Calculate change in phi for heat evolution.

    Parameters
    ----------
    phi : float
        Melt fraction
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : float
        Scale factor for the complex shear modulus ('True' or '
           False')
    dt : float
        Time step [s]

    Returns
    -------
    dphi : float
        Change in melt fraction
    q_tidal : float
        Tidal dissipation [W]
    q_v_tidal : float
        Volumetric tidal dissipation [W/m^3]
    q_out : float
        Outward flux [W]
    q_a_out : float
        Outward surface heat flux [W/m^2]
    Ra : float
        Rayleigh number of the shell
    d : float
```

```
        Thickness of the shell [m]
    Rprime : float
        Radius of the melt layer [m]
    '''
    #Calculate tidal diss
    q_tidal , q_v_tidal, _ = shell_diss(phi, eta_tidal, scale)

    #Calculate outward flux
    q_out , q_a_out, Ra = shell_out(phi, eta_tidal, eta_conv,
       scale)

    #Calculate shell volume and area
    Rprime, d = melt_layer(phi)
    V = 4/3 * pi * R**3 #Of whole body
    V2 = 4/3 * pi * (R**3 - Rprime**3) #Of shell
    A = 4 * pi * R**2 #Of surface

    #Check fluxes calculated right
    q_tidal_2 = q_v_tidal * V2 #W
    q_out_2 = q_a_out * A #W
    if abs(q_tidal - q_tidal_2) > 1e-10:
        print('Tidal dissipation not calculated right')
    if abs(q_out - q_out_2) > 1e-10:
        print('Outward flux not calculated right')

    #Calculate change in melt fraction
    dphi = (V2/V * q_v_tidal - A/V * q_a_out) / (rhoms * L) * dt

    return dphi, q_tidal, q_v_tidal, q_out, q_a_out, Ra, d,
       Rprime
```

B.3 Fluxes

```
#Functions to calculate fluxes in a mantle shell.

#Import parameters
from param.parameters_io import R, Om, e
from param.parameters_rheo import Tml, rhoms, Ts, k, alpha,
   kappa
from param.parameters_uni import pi

#Import functions
from fn_rheology import complex_mod
```

```
from fn_properties import grav, melt_layer

#Functions
def shell_diss(phi, eta_tidal, scale):
    """Calculate tidal dissipation in the shell.

    Parameters
    ----------
    phi : float
        Melt fraction
    eta_tidal : float
        Tidal viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    q_tidal : float
        Tidal dissipation rate [W]
    q_v_tidal : float
        Volumetric tidal dissipation rate [W/m^3]
    gR : float
        Gravity at the surface [m/s^2]
    """
    #Calculate gravity
    gR, Rprime = grav(phi)

    #Calculate complex shear modulus
    muc, mucc, _, mu_im, _ = complex_mod(phi, eta_tidal, scale)

    q_v_tidal = abs(mu_im/2 * Om * (3 * Om**2 * rhoms * R**2 * e
        )**2 / ((muc + 2/5*rhoms*gR*R) * (mucc + 2/5*rhoms*gR*R))
        )
    q_tidal = q_v_tidal * 4/3 * pi * (R**3 - Rprime**3) # W


    return q_tidal, q_v_tidal,  gR

def shell_conv(phi, eta_tidal, eta_conv, scale):
    """Calculate convective flux in the shell.

    Parameters
    ----------
    phi : float
        Melt fraction
```

```
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    q_conv : float
        Convective flux [W]
    q_a_conv : float
        Convective surface heat flux [W/m^2]
    """
    #Calculate internal heating rate
    _, H_v, gR = shell_diss(phi, eta_tidal, scale)
    H_m = H_v / rhoms

    #Calculate temperature grad
    deltaT = Tml - Ts

    #Calculate thickness of the shell
    d = melt_layer(phi)[1]

    #Calculate convective surface heat flux
    q_a_conv = deltaT / 10 * (k**2 * alpha * gR * rhoms**2 * H_m * d**2 / (kappa * eta_conv) )**(1/3)  # W/m^2
    q_conv = q_a_conv * 4 * pi * R**2 # W

    return q_conv, q_a_conv

def shell_cond(phi):
    """Calculate conductive flux in the shell.

    Parameters
    ----------
    phi : float
        Melt fraction

    Returns
    -------
    q_cond : float
        Conductive heat flux [W]
    q_a_cond : float
        Conductive surface heat flux [W/m^2]
```

```
    """
    #Calculate temperature grad
    deltaT = Tml - Ts

    #Calculate thickness of the shell
    d = melt_layer(phi)[1]

    #Calculate conductive heat flux
    q_a_cond = k * deltaT / d
    q_cond = q_a_cond * 4 * pi * R**2 # W

    return q_cond, q_a_cond

def shell_out(phi, eta_tidal, eta_conv, scale):
    """Calculate outward flux in the shell.

    Parameters
    ----------
    phi : float
        Melt fraction
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    q_out : float
        Outward flux [W]
    q_a_out : float
        Outward surface heat flux [W/m^2]
    Ra : float
        Rayleigh number of the shell
    """
    #Calculate Rayleigh number
    Ra = Rayleigh(phi, eta_tidal, eta_conv, scale)

    #Determine regime and outward flux
    if Ra < 1e3:
        q_out, q_a_out = shell_cond(phi)[0]
    else:
        q_out, q_a_out = shell_conv(phi, eta_tidal, eta_conv,
            scale)[0]
```

```python

    return q_out, q_a_out, Ra

def Rayleigh(phi, eta_tidal, eta_conv, scale):
    """Calculate Rayleigh number in the shell.

    Parameters
    ----------
    phi : float
        Melt fraction.
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    Ra : float
        Rayleigh number of the shell
    """
    #Calculate shell thickness
    d = melt_layer(phi)[1]

    #Calculate internal heating rate
    _, H_v, gR = shell_diss(phi, eta_tidal, scale)
    H_m = H_v / rhoms

    #Calculate Rayleigh number
    Ra = alpha * gR * rhoms**2 * H_m * d**5 / (k * kappa * eta_conv)

    return Ra
```

B.4 Rheology

```python
#Calculate complex shear modulus and phase transitions.

#Import parameters
from param.parameters_io import n, R
from param.parameters_rheo import mus, Tms, Tc, Tml, Tmu

```

```
#Import functions
from fn_properties import melt_layer


##########Call complex shear modulus##########
def complex_mod(phi, eta_tidal, scale):
    """ Calculate complex shear modulus for given melt fraction
        and viscosity.

    Parameters:
    -----------
    phi : float
        Melt fraction.
    eta_tidal : float
        Tidal viscosity [Pa.s].
    scale: str
        Scale for shear modulus ('True' or 'False').

    Returns:
    --------
    muc : float
        Complex shear modulus.
    mucc : float
        Conjugate of complex shear modulus.
    mu_re : float
        Real part of complex shear modulus.
    mu_im : float
        Imaginary part of complex shear modulus.
    mu : float
        Shear modulus.
    """
    #Calculate shear modulus
    mu = mus
    omega = mu / eta_tidal

    #Calculate shell thickness
    d = melt_layer(phi)[1]

    if scale == 'True':
        mu_re = d/R*(n**2 * mu / (n**2 + omega**2))
        mu_im = d/R*(n * mu * omega / (n**2 + omega**2))
    elif scale == 'False':
        mu_re = n**2 * mu / (n**2 + omega**2)
        mu_im = n * mu * omega / (n**2 + omega**2)
    else:
```

```
            print('Error with d/R')

    muc = mu_re + 1j * mu_im
    mucc = mu_re - 1j * mu_im

    return muc, mucc, mu_re, mu_im, mu

##########Call phase transitions##########
def phase_transitions():
    """ Returns phase transition temperatures.

    Returns:
    ----------
    Tms : float
        Solidus temperature.
    Tc : float
        Critical temperature (melt dominance).
    Tml : float
        Liquidus temperature.
    Tmu : float
        Critical temperature for shear modulus.
    """
    return Tms, Tc, Tml, Tmu
```

B.5 Properties

```

#Functions that vary with the layer distribution of the body.

#Import parameters
from param.parameters_uni import pi, G
from param.parameters_io import R
from param.parameters_rheo import rhoms, rhoml

def melt_layer(phi):
    '''
    Calculate radius of melt layer, for given melt fraction.

    Parameters
    ----------
    phi : float
        Melt fraction.

```

```
    Returns
    -------
    Rprime : float
        Radius of the melt layer [m].
    d : float
        Thickness of the shell [m].
    '''
    Rprime = R * (phi*rhoms / ((1 - phi)*rhoml + phi*rhoms))
       **(1/3)
    d = R - Rprime

    return Rprime, d

def grav(phi):
    """
    Calculate gravity at the surface of a body with a given melt
        fraction.

    Parameters
    ----------
    phi : float
        Melt fraction.

    Returns
    -------
    gR : float
        Gravity at the surface of the body [m/s^2].
    Rprime : float
        Radius of the melt layer [m].
    """
    Rprime = melt_layer(phi)[0]
    gR = 4/3 * pi * G / R**2 * (rhoml*Rprime**3 + rhoms*(R**3 -
       Rprime**3))

    return gR, Rprime
```

B.6 Parameters

```
#Universal parameters
import numpy as np

#Universal constants
G = 6.6743015e-11 #Nm^2/kg^2, universal gravitational constant
```

```
pi = np.pi
Rgas = 8.321 #J/(K mol), universal gas constant
```

```
#Orbital and physical parameters for Io
import numpy as np

#Planetary properties
R = 1.82e6 #m, radius

#Orbital properties
n = 2 * np.pi / (1.769137786 * 24 * 60**2) #rad/s, mean motion
Om = n #rad/s, spin on axis
e = 0.0041 #eccentricity of orbit
```

```
#Thermal and rheological parameters for silicate

##########Densities##########
rhoms = 3300 #kg/(m^3), solid mantle density
rhoml = rhoms #kg/(m^3), liquid mantle density

##########Temperature transitions##########
Tml = 1698 #K, liquidus, Moore 2003
Ts = 200 #K, surface temperature

##########Shear modulus##########
mus = 5e10 #Pa, solid shear modulus, Fischer and Spohn 1990

##########Thermal parameters##########
cp = 1.23e3 #J/(kg K), specific heat capacity, Fischer and Spohn
    1990
k = 4 #W/(mK), thermal conductivity, Moore 2003
kappa = 1e-6 #m^2/s, thermal diffusivity, Moore 2003
alpha = 3e-5 #1/K, thermal expansivity, Moore 2003
L = 5 * 1e5 #J/kg, latent heat, Moore 2001
```

B.7 Run model

```
#Run heat evolution
from heat_evolution import heat_evol, save_evol, plot_evol

dR = 'True'
eta_tidal = 1e17
eta_conv = 1e21
```

```
folder = ''

for phii in [0.05, 0.8]:
    t, phi, q_tidal, q_out, date, d, Ra, eta_tidal, eta_conv, dR
        = heat_evol(phii, eta_tidal, dR, eta_conv, t_end=4.5e9,
       dt=10)
    run = save_evol(t, phi, q_tidal, q_out, date, d, Ra,
       eta_tidal, eta_conv, dR, folder)
    plot_evol(folder, run)
    print(f'Run {run} complete')
```

C

Theory

C.0.1 Fluxes

```
#Fluxes in the shell

#Import parameters
from heat_evolution.param.parameters_rheo import rhoml, rhoms,
    mus, Tml, Ts, k, alpha, kappa
from heat_evolution.param.parameters_io import R, n, Om, e
from heat_evolution.param.parameters_uni import G, pi

#####Fluxes#####
def shell_diss(d, eta_tidal, scale):
    """Calculate tidal dissipation in the shell.

    Parameters
    ----------
    d : float
        Shell thickness [m]
    eta_tidal : float
        Tidal viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    q_tidal : float
```

```
        Tidal dissipation rate [W]
    q_v_tidal : float
        Volumetric tidal dissipation rate [W/m^3]
    gR : float
        Gravity at the surface [m/s^2]
    """
    #Calculate gravity
    Rprime = R - d
    gR = 4/3 * pi * G / R**2 * (rhoml*Rprime**3 + rhoms*(R**3 - Rprime**3)) #rhoms=rhoml

    #Calculate complex shear modulus
    mu = mus #5e10
    omega = mu / eta_tidal

    #Calculate shell thickness
    if scale == 'True':
        mu_re = d/R*(n**2 * mu / (n**2 + omega**2))
        mu_im = d/R*(n * mu * omega / (n**2 + omega**2))
    elif scale == 'False':
        mu_re = n**2 * mu / (n**2 + omega**2)
        mu_im = n * mu * omega / (n**2 + omega**2)
    else:
        print('Error with d/R')

    muc = mu_re + 1j * mu_im
    mucc = mu_re -  1j * mu_im

    #Calculate tidal dissipation
    q_v_tidal = abs(mu_im/2 * Om * (3 * Om**2 * rhoms * R**2 * e)**2 / ((muc + 2/5*rhoms*gR*R)
                                                                                          *
```

```
    q_tidal = q_v_tidal * 4/3 * pi * (R**3 - Rprime**3) #W

    return q_tidal, q_v_tidal,  gR

def shell_conv(d, eta_tidal, eta_conv, scale):
    """Calculate convective heat flux in the shell.

    Parameters
    ----------
    d : float
        Shell thickness [m]
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    q_conv: float
        Convective heat flux [W]
    q_a_conv : float
        Convective surface heat flux [W/m^2]
    """

    #Calculate tidal dissipation
    _, H_v, gR = shell_diss(d, eta_tidal, scale)
    H_m = H_v / rhoms

    #Calculate temperature grad
    deltaT = Tml - Ts

    #Calculate convective surface heat flux
    q_a_conv = deltaT / 10 * (k**2 * alpha * gR * rhoms**2 * H_m
        * d**2 / (kappa * eta_conv) )**(1/3)  #W/m^2
```

```

    #Calculate convective heat flux
    q_conv = q_a_conv * 4 * pi * R**2 #W

    return q_conv, q_a_conv

def shell_cond(d):
    """Calculate conductive heat flux in the shell.

    Parameters
    ----------
    d : float
        Shell thickness [m]

    Returns
    -------
    q_cond : float
        Conductive heat flux [W]
    q_a_cond : float
        Conductive surface heat flux [W/m^2]
    """
    #Calculate temperature grad
    deltaT = Tml - Ts

    #Calculate conductive surface heat flux
    q_a_cond = k * deltaT / d #W/m^2

    #Calculate conductive heat flux
    q_cond = q_a_cond * 4 * pi * R**2 #W

    return q_cond, q_a_cond

def shell_out(d, eta_tidal, eta_conv, scale):
    """Calculate outward heat flux in the shell.

    Parameters
    ----------
    d : float
        Shell thickness [m]
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')
```

```python

    Returns
    -------
    q_out : float
        Outward heat flux [W]
    """
    #Calculate Rayleigh number
    Ra = Rayleigh(d, eta_tidal, eta_conv, scale)

    #Determine regime
    if Ra < 1e3:
        q_a_out = shell_cond(d)[1] #W/m^2
    else:
        q_a_out = shell_conv(d, eta_tidal, eta_conv, scale)[1] #W/m^2

    #Calculate outward heat flux
    q_out = q_a_out * 4 * pi * R**2 #W

    return q_out, q_a_out

def Rayleigh(d, eta_tidal, eta_conv, scale):
    """Calculate Rayleigh number in the shell.

    Parameters
    ----------
    d : float
        Shell thickness [m]
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')

    Returns
    -------
    Ra : float
        Rayleigh number
    """
    #Calculate tidal dissipation rate
    _, H_v, gR = shell_diss(d, eta_tidal, scale)
    H_m = H_v / rhoms #W/kg

    #Calculate Rayleigh number
```

```
    Ra = alpha * gR * rhoms**2 * H_m * d**5 / (k * kappa * eta_conv)

    return Ra
```

C.1 Converters

```
#Functions to convert between d_norm and phi

def d_norm_to_phi(d_norm):
        return (1 - d_norm)**3
def phi_to_d_norm(phi):
        return 1 - phi**(1/3)
```

C.2 Thermal equilibria

```
#Function to find the steady states and unstable points of the shell, and save them
#Import libraries
import numpy as np
import pandas as pd

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from fluxes import shell_diss, shell_out


##### Steady state finder #####
def steady_state_finder(name, scale, eta_tidal = np.logspace(9,19,4000), d= np.linspace(0.01, R, 100000), eta_conv = np.logspace(23, 17, 4)):
    """
    Function to find and save the stable and unstable points

    Parameters
    ----------
    name : str
        Name of the run. Used to save the results.
```

```
    scale : bool
        Scale factor for the shear modulus ('True' or 'False')
    eta_tidal : array
        Tidal viscosity [Pa.s]
    d : array
        Shell thickness [m]
    eta_conv : array
        Convective viscosity [Pa.s]
    """
    #For chosen range of convective viscosities
    for eta_conv_i in eta_conv:
            #For chosen range of tidal viscosities
            for eta_tidal_i in eta_tidal:
                #Calculate fluxes
                h = []; q = []
                for di in d:
                    h.append(shell_diss(di, eta_tidal_i, scale)[0])
                    q.append(shell_out(di, eta_tidal_i, eta_conv_i, scale)[0])
                h = np.array(h)
                q = np.array(q)

                #Find crossover points
                flux = h - q
                type = 'None'

                for i in range(1, len(flux)):
                    if flux[i-1] * flux[i] <= 0:  #Look for sign changes (+ to - or - to +)
                        if flux[i-1] - flux [i] < 0: #If the sign change is from - to + is stable
                            type = 'Stable'
                        else: #If the sign change is from + to - is unstable
                            type = 'Unstable'

                        #Save point
                        run_data = pd.DataFrame({
                                        'Convective viscosity': [eta_conv_i],
                                        'Tidal viscosity': [eta_tidal_i],
                                        'Type': [type],
                                        'Thickness': [d[i]],
```

```
                                        'scale': [scale]
                                    })
                        #If the file already exists, append the data to it
                        try:
                            prev_data = pd.read_csv(f'Crossover_points_scale_{scale}{name}.csv')
                            run_data = pd.concat([prev_data, run_data], ignore_index=True)
                        except FileNotFoundError:
                            pass
                        run_data.to_csv(f'Crossover_points_scale_{scale}{name}.csv', index=False)

                #If no crossover point is found, save the no result
                if type == 'None':
                    #Save the no result
                    run_data = pd.DataFrame({
                            'Convective viscosity': [eta_conv_i],
                            'Tidal viscosity': [eta_tidal_i],
                            'Type': [type],
                            'Thickness': [np.nan],
                            'scale': [scale]
                        })
                    #If the file already exists, append the data to it
                    try:
                        prev_data = pd.read_csv(f'Crossover_points_scale_{scale}{name}.csv')
                        run_data = pd.concat([prev_data, run_data], ignore_index=True)
                    except FileNotFoundError:
                        pass
                    run_data.to_csv(f'Crossover_points_scale_{scale}{name}.csv', index=False)
            #Check it is running
            print(f'eta_conv_i = {eta_conv_i} done')
```

```
#Find ranges of phi for stable and unstable points

```

```python
#Import modules
import pandas as pd

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from converter import d_norm_to_phi

#Functions
def range_unstable(name, scale, eta_conv=None):
    """
    Function to find the range of unstable phi for a given file name and scale. Can
    select convective viscosity.

    Parameters
    ----------
    name : str
        Name of the run. Used to save the results.
    scale : bool
        Scale factor for the shear modulus ('True' or 'False')
    eta_conv : float, optional
        Convective viscosity [Pa.s]. The default is None.
    """
    #Read file
    data = pd.read_csv(f'Crossover_points_scale_{scale}{name}.csv')
    #Filter convective viscosity
    if eta_conv is not None:
        data = data[data['Convective viscosity'] == eta_conv]

    #Find unstable points
    data_unstable = data[data['Type'] == 'Unstable']
    #Find unstable thicknesses
    unstable_thickness = data_unstable['Thickness'].values
    #Convert to melt fraction
    unstable_phi = [d_norm_to_phi(thickness/R) for thickness in unstable_thickness]

    #####Min phi#####
    #Find minimum phi
    min_phi = min(unstable_phi)
    #Find correlating dissipitative viscosity
    min_diss = data_unstable[data_unstable['Thickness'] ==
```

```
        unstable_thickness[unstable_phi.index(min_phi)]]['Tidal viscosity'].values[0]
    #Find correlating convective viscosity
    min_conv = data_unstable[data_unstable['Thickness'] == unstable_thickness[unstable_phi.index(min_phi)]]['Convective viscosity'].values[0]

    #####Max phi#####
    #Find maximum phi
    max_phi = max(unstable_phi)
    #Find correlating dissipitative viscosity
    max_diss = data_unstable[data_unstable['Thickness'] == unstable_thickness[unstable_phi.index(max_phi)]]['Tidal viscosity'].values[0]
    #Find correlating convective viscosity
    max_conv = data_unstable[data_unstable['Thickness'] == unstable_thickness[unstable_phi.index(max_phi)]]['Convective viscosity'].values[0]

    #Print results
    print(f"min threshold phi: {min_phi:.5f}",
          f"max normalised thickness: {unstable_thickness[unstable_phi.index(min_phi)]/R:.2f}",
            f"Tidal viscosity: {min_diss:.2e}",
            f"convective viscosity: {min_conv:.2e}",
            )
    print(f"max threshold phi: {max_phi:.5f}",
          f"min normalised thickness: {unstable_thickness[unstable_phi.index(max_phi)]/R:.2f}",
            f"Tidal viscosity: {max_diss:.2e}",
            f"convective viscosity: {max_conv:.2e}",
            )


def range_stable(name, scale, eta_conv=None):
    """
    Function to find the range of stable phi for a given file name and scale. Can
    select convective viscosity.

    Parameters
    ----------
    name : str
        Name of the run. Used to save the results.
    scale : bool
```

```
        Scale factor for the shear modulus ('True' or 'False')
    eta_conv : float, optional
        Convective viscosity [Pa.s]. The default is None.
    """
    #Read file
    data = pd.read_csv(f'Crossover_points_scale_{scale}{name}.csv')
    #Filter convective viscosity
    if eta_conv is not None:
        data = data[data['Convective viscosity'] == eta_conv]

    #Find stable points
    data_stable = data[data['Type'] == 'Stable']
    #Find stable thicknesses
    stable_thickness = data_stable['Thickness'].values
    #Convert to melt fraction
    stable_phi = [d_norm_to_phi(thickness/R) for thickness in stable_thickness]

    #####Min phi#####
    #Find minimum phi
    min_phi = min(stable_phi)
    #Find correlating dissipitative viscosity
    min_diss = data_stable[data_stable['Thickness'] == stable_thickness[stable_phi.index(min_phi)]]['Tidal viscosity'].values[0]
    #Find correlating convective viscosity
    min_conv = data_stable[data_stable['Thickness'] == stable_thickness[stable_phi.index(min_phi)]]['Convective viscosity'].values[0]

    #####Max phi#####
    #Find maximum phi
    max_phi = max(stable_phi)
    #Find correlating dissipitative viscosity
    max_diss = data_stable[data_stable['Thickness'] == stable_thickness[stable_phi.index(max_phi)]]['Tidal viscosity'].values[0]
    #Find correlating convective viscosity
    max_conv = data_stable[data_stable['Thickness'] == stable_thickness[stable_phi.index(max_phi)]]['Convective viscosity'].values[0]

    print(f"min stable phi: {min_phi:.5f}",
          f"max normalised thickness: {stable_thickness[
```

```
            stable_phi.index(min_phi)]/R:.2f}",
            f"Tidal viscosity: {min_diss:.2e}",
            f"convective viscosity: {min_conv:.2e}",
            )
    print(f"max stable phi: {max_phi:.5f}",
          f"min normalised thickness: {stable_thickness[
            stable_phi.index(max_phi)]/R:.2f}",
            f"Tidal viscosity: {max_diss:.2e}",
            f"convective viscosity: {max_conv:.2e}",
            )

def all_unstable(name, scale, eta_conv=None):
    """
    Function to find the all unstable phi for a given file name
        and scale. Can
    select convective viscosity.

    Parameters
    ----------
    name : str
        Name of the run. Used to save the results.
    scale : bool
        Scale factor for the shear modulus ('True' or 'False')
    eta_conv : float, optional
        Convective viscosity [Pa.s]. The default is None.
    """
    #Read file
    data = pd.read_csv(f'Crossover_points_scale_{scale}{name}.
        csv')
    #Filter convective viscosity
    if eta_conv is not None:
        data = data[data['Convective viscosity'] == eta_conv]

    #Find unstable points
    data_unstable = data[data['Type'] == 'Unstable']
    #Find unstable thicknesses
    unstable_thickness = data_unstable['Thickness'].values
    #Convert to melt fraction
    unstable_phi = [d_norm_to_phi(thickness/R) for thickness in
        unstable_thickness]
    #Correlating dissipitative viscosity
    unstable_diss = data_unstable['Tidal viscosity'].values
    #Correlating convective viscosity
    unstable_conv = data_unstable['Convective viscosity'].values

```

```
    for phi, dR, diss, conv in zip(unstable_phi,
       unstable_thickness, unstable_diss, unstable_conv):
        print(f"unstable phi: {phi:.5f}",
              f"normalised thickness: {dR/R:.2f}",
                f"Tidal viscosity: {diss:.2e}",
                f"convective viscosity: {conv:.2e}",
                )



def all_stable(name, scale, eta_conv=None):
    """
    Function to find the all stable phi for a given file name
       and scale. Can
    select convective viscosity.

    Parameters
    ----------
    name : str
        Name of the run. Used to save the results.
    scale : bool
        Scale factor for the shear modulus ('True' or 'False')
    eta_conv : float, optional
        Convective viscosity [Pa.s]. The default is None.
    """
    #Read file
    data = pd.read_csv(f'Crossover_points_scale_{scale}{name}.
       csv')
    #Filter convective viscosity
    if eta_conv is not None:
        data = data[data['Convective viscosity'] == eta_conv]

    #Find stable points
    data_stable = data[data['Type'] == 'Stable']
    #Find stable thicknesses
    stable_thickness = data_stable['Thickness'].values
    #Convert to melt fraction
    stable_phi = [d_norm_to_phi(thickness/R) for thickness in
       stable_thickness]
    #Correlating dissipitative viscosity
    stable_diss = data_stable['Tidal viscosity'].values
    #Correlating convective viscosity
    stable_conv = data_stable['Convective viscosity'].values

    for phi, dR, diss, conv in zip(stable_phi, stable_thickness,
```

```
        stable_diss, stable_conv):
        print(f"stable phi: {phi:.5f}",
              f"normalised thickness: {dR/R:.2f}",
                f"Tidal viscosity: {diss:.2e}",
                f"convective viscosity: {conv:.2e}",
                )

all_unstable('', True,eta_conv=1e21)
```

C.3 Plots

C.3.1 Chapter 2: Tidal

```
#Plot of tidal dissipation rate over mean motion/maxwell frequency
#Import libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from cmcrameri import cm

#Import parameters
from heat_evolution.param.parameters_rheo import rhoms, mus
from heat_evolution.param.parameters_io import R, n, Om, e
from heat_evolution.param.parameters_uni import G, pi


def sphere_diss(eta_tidal):
    """Calculate tidal dissipation in a sphere.

    Parameters
    ----------
    eta_tidal : float
        Tidal viscosity [Pa.s]

    Returns
    -------
    q_tidal : float
        Tidal dissipation rate [W]
    q_v_tidal : float
        Volumetric tidal dissipation rate [W/m^3]
    gR : float
```

```
        Gravity at the surface [m/s^2]
    """
    #Calculate gravity
    gR = 4/3 * pi * G * rhoms * R

    #Calculate complex shear modulus
    mu = mus #5e10
    omega = mu / eta_tidal

    mu_re = n**2 * mu / (n**2 + omega**2)
    mu_im = n * mu * omega / (n**2 + omega**2)
    muc = mu_re + 1j * mu_im
    mucc = mu_re - 1j * mu_im

    #Calculate tidal dissipation
    q_v_tidal = abs(mu_im/2 * Om * (3 * Om**2 * rhoms * R**2 * e
        )**2 / ((muc + 2/5*rhoms*gR*R) * (mucc + 2/5*rhoms*gR*R))
        )#W/m^3
    q_tidal = q_v_tidal * 4/3 * pi * R**3 #W

    return q_tidal, q_v_tidal

def maxwell(eta = np.logspace(10, 20, 10000)):
    """Plot tidal dissipation rate over mean motion/maxwell
        frequency.

    Parameters
    ----------
    eta : array_like
        Tidal viscosity [Pa.s]
    """
    #####Colors#####
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)
    c_teal = cm.roma(0.6)

    #Calculate tidal dissipation
    h = sphere_diss(eta)[1]

    #Calculate dimensionless a
    maxwell = mus/eta
    a = n/maxwell

    #Find maximum
    max_h = np.max(h)
```

```
    max_a = a[np.argmax(h)]

    #Plot
    sns.set_context('notebook')  #Options include 'paper', 'notebook', 'talk', and 'poster'

    plt.plot(a, h, color = c_blue)
    plt.scatter(max_a, max_h, color=c_teal, label=f'Max at $a$ = {max_a:.2f}')

    plt.xlabel(r'Mean motion / Maxwell frequency, $a=n/\omega$')
    plt.ylabel(r'Volumetric tidal dissipation rate, (W m$^{-3}$)')
    plt.xscale('log')
    plt.yscale('log')
    plt.legend()

    #Save
    plt.savefig(f'plot/max.png', dpi=500)
    plt.show()

maxwell()
```

```
#Plot of tidal dissipation with and without scaling complex shear modulus

#Import libraries
import numpy as np
import matplotlib.pyplot as plt
import math
from matplotlib.lines import Line2D
from cmcrameri import cm

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from converter import d_norm_to_phi
from fluxes import shell_diss

#####Plot of dissipation with and without scale#####
def diss_scale(eta_tidal, eta_conv, d=np.linspace(R*1e-3, R, 10000)):
    """Plot tidal dissipation with and without scaling complex shear modulus.
```

```

    Parameters
    ----------
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    d : array
        Shell thickness [m]
    """
    #####Colors#####
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)
    c_teal = cm.roma(0.5)

    #####Create figure#####
    fig = plt.figure(figsize=(8, 10))
    #Adjust to fit x labels
    grid = plt.GridSpec(3, 1, height_ratios=[1, 1, 0.00005],
                        hspace=0.3, wspace=0.3)  #Single plot
                           with space below for extended x
                           labels
    axb = fig.add_subplot(grid[1, 0]) #bottom
    axa = fig.add_subplot(grid[0, 0], sharex=axb)  #top
    plt.subplots_adjust(left=0.15, right=0.9, top=0.95)

    ######Calculate data######
    #Normalise thickness
    d_norm = d/R

    #Calculate fluxes
    for scale in ['True', 'False']:
        h_v = []; h = []

        for di in d:  #Iterating because it gets unhappy about
           the Rayleigh number check
            h_v.append(shell_diss(di, eta_tidal, scale)[1])
            h.append(shell_diss(di, eta_tidal, scale)[0])

        #Convert to numpy arrays
        h_v = np.array(h_v)
        h = np.array(h)

        #Plot
        axa.plot(d_norm, h_v, color=c_blue, linestyle='-' if
```

```
            scale == 'True' else ':')
        axb.plot(d_norm, h, color=c_blue, linestyle='--' if scale == 'True' else ':')

        if scale == 'True':
            #Find and plot maximums
            #Volumetric
            h_v_max = np.max(h_v)
            h_v_max_index = np.argmax(h_v)
            d_v_max = d[h_v_max_index]
            axa.scatter(d_v_max/R, h_v_max, color=c_teal, marker='o', s=100, zorder=10,
                        label = '$\lambda$ = ' + f'{d_v_max/R:.2f}')

            #Total
            h_max = np.max(h)
            h_max_index = np.argmax(h)
            d_max = d[h_max_index]
            axb.scatter(d_max/R, h_max, color=c_teal, marker='o', s=100, zorder=10,
                        label = '$\lambda$ = ' + f'{d_max/R:.2f}')

    ####Formatting######
    wordsize=16
    legendsize=14
    widthsize=2
    #Get lines
    for line in plt.gca().get_lines():
        line.set_linewidth(2)

    #Legends
    legend1 = axa.legend(loc='lower right', fontsize=legendsize, title_fontsize=legendsize, framealpha=1)
    legend2 = axb.legend(loc='lower right', fontsize=legendsize, title_fontsize=legendsize, framealpha=1)
    flux_lines = [
        Line2D([0], [0], color=c_blue, linestyle='--', label='$\lambda$'),
        Line2D([0], [0], color=c_blue, linestyle=':', label='No $\lambda$')
    ]
    legend3 = axb.legend(handles=flux_lines, loc='lower left', fontsize=legendsize,
```

```
                                  title_fontsize=legendsize, framealpha
                                     =1)

    axa.add_artist(legend1)
    axb.add_artist(legend2)
    axb.add_artist(legend3)

    ##########Y Axes##########
    axa.set_yscale('log')
    axa.set_ylabel('(W m$^{-3}$)', fontsize=wordsize)
    axa.tick_params(axis='y', labelsize=wordsize, width=
        widthsize)

    axb.set_yscale('log')
    axb.set_ylabel('(W)', fontsize=wordsize)
    axb.tick_params(axis='y', labelsize=wordsize, width=
        widthsize)

    #Titles
    axa.set_title('Volumetric tidal dissipation rate', fontsize=
        wordsize)
    axb.set_title('Total tidal dissipation rate', fontsize=
        wordsize)

    #########X Axes##########
    axa.tick_params(axis='x', which='both',
                     bottom=True, top=False, labelbottom=False
                     , width=widthsize)
    axa.sharex(axb)
    ax1 = axb

    #d/R
    ax1.set_xscale('log')
    ax1.set_xlabel('Thickness of shell / Radius, $d/R$', color=
        c_blue, fontsize=wordsize)
    ax1.tick_params(axis='x', colors=c_blue, labelsize=wordsize,
         width=widthsize)

    #phi
    ax2 = ax1.twiny()
    ax2.set_xscale('log')

    #Limits
    d_min, d_max = ax1.get_xlim()
    ax2.set_xlim(d_min, d_max)
```

```

    #Convert
    d_norm_values = [1e0, 1e-1, 1e-2, 1e-3]
    phi_values = [d_norm_to_phi(d) for d in d_norm_values]

    #Ticks
    ax2.set_xticks(d_norm_values) #ticks line up with d/R
    ax2.set_xticklabels(f'{phi:.3f} ' for phi in phi_values) #
        but have value of phi

    #Axis Label
    ax2.set_xlabel('Melt fraction, $\phi$', color=c_red,
        fontsize=wordsize)

    #Style secondary axis
    ax2.xaxis.set_ticks_position('bottom')
    ax2.xaxis.set_label_position('bottom')
    ax2.tick_params(axis='x', colors=c_red, which='both',
        labelsize=wordsize, width=widthsize)

    #Spines
    ax2.spines['bottom'].set_color(c_red)
    ax2.spines['bottom'].set_linewidth(widthsize/2)

    #Shifting up and down
    ax1.xaxis.set_label_coords(0.5, -0.1) #dR label
    ax2.spines['bottom'].set_position(('outward', 49)) #Melt
        fraction spines
    ax2.xaxis.set_label_coords(0.5, -0.3)  #Melt fraction label

    #Save
    plt.savefig(f'plot/scale.png', dpi=500)
    with open(f'plot/scale.txt', 'w') as f:
        f.write(f'eta_tidal = 1e{math.log10(eta_tidal)}\n')
        f.write(f'eta_conv = 1e{math.log10(eta_conv)}\n')

diss_scale(eta_tidal=1e17, eta_conv=1e21)
```

```
#Plot of complex shear modulus as a function of mean motion
#Import modules
import numpy as np
import matplotlib.pyplot as plt
from cmcrameri import cm
import seaborn as sns

```

```
#Import parameters
from heat_evolution.param.parameters_rheo import mus
from heat_evolution.param.parameters_io import n, Om, e

def tilde(a):
    """Calculate the complex shear modulus as a function of mean
        motion/maxwell frequency.

    Parameters
    ----------
    a : float
        Mean motion / Maxwell frequency.

    Returns
    -------
    muc : complex
        Complex shear modulus [Pa].
    """
    #Call the shear modulus
    mu = mus #5e10

    #Calculate the complex shear modulus
    mu_re = mu * a**2/ (a**2 + 1)
    mu_im = 1/a * mu * a**2 / (a**2 + 1)

    muc = mu_re + 1j * mu_im

    return muc


def plot(a=np.logspace(-1,1,100)):
    """Plot the complex shear modulus as a function of mean
       motion/maxwell frequency.

    Parameters
    ----------
    a : array
        Mean motion / Maxwell frequency.
    """
    #####Colors#####
    c_blue = cm.roma(0.9)
    c_aqua = cm.roma(0.6)

    #####Plot components#####
    sns.set_context('notebook') #Options include 'paper', '
```

```
notebook', 'talk', and 'poster'

    #Two lines show change in behavior of the complex shear modulus
    real_part=tilde(a).real
    imag_part=tilde(a).imag
    plt.plot(a, real_part, label=r'$\tilde{\mu}_{Re}$', color=c_blue)
    plt.plot(a, imag_part, label=r'$\tilde{\mu}_{Im}$', color=c_aqua)

    #Reference line a = 1
    plt.axvline(1, color='k', linestyle='--')
    plt.text(1, max(real_part) / 8, 'a=1', color='k', ha='right', va='center', rotation=90)

    #Don't want asymptotic lines to be in the legend
    plt.legend()

    #####Plot asymptotic lines#####
    #Asymptotic lines
    aleft = np.array([a[0] / 2.5, a[0]])
    aright = np.array([a[-1], a[-1] * 2.5])

    #Real left (limit a -> 0), mu_re = a**2 * mu
    rl = plt.plot(aleft, aleft**2 * mus * np.ones(len(aleft)), color=c_blue, linestyle='--',
            label=r'$a^2\mu$')
    #Real right (limit a -> infty), mu_re = mu
    rr = plt.plot(aright, mus * np.ones(len(aright)), color=c_blue, linestyle='--',
            label=r'$\mu$')
    #Imaginary left (limit a -> 0)
    il = plt.plot(aleft, aleft * mus, color=c_aqua, linestyle='--',
            label=r'$a\mu$')
    #Imaginary right (limit a -> infty), mu_im = mu / a
    ir = plt.plot(aright, 1/aright * mus, color=c_aqua, linestyle='--',
            label=r'$\mu/a$')

    #Text annotations
    lines = [rl[0], rr[0], il[0], ir[0]]
    space_x = [1.5, 1.5, 1.5, 1.5] #Control position of text
    space_y = [3, 0.9, 1.8, 0.5]
```

```
    color=[c_blue, c_blue, c_aqua, c_aqua]

    for i, line in enumerate(lines):#Iterate for each asymptotic line
        x1, x2 = line.get_xdata()[0], line.get_xdata()[-1]
        y1, y2 = line.get_ydata()[0], line.get_ydata()[-1]
        slope = np.degrees(np.arctan2(np.log10(y2/y1), np.log10(x2/x1)))

        plt.text(x1 * space_x[i], y1 * space_y[i],
                 line.get_label(), color=color[i],
                 ha='left', va='top', rotation=slope)

    #####Scale & labels#####
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Mean motion / Maxwell frequency, $a=n/\omega$')
    plt.ylabel(r'$\tilde{\mu}$ (Pa)')


    #Save & show
    plt.savefig('plot/tilde_mu.png', dpi=500, bbox_inches='tight')
    plt.show()
```

C.3.2 Chapter 3: Heat

```
#Plot of the heat fluxes and Rayleigh number as a function of normalised shell thickness or melt fraction
#Import libraries
import numpy as np
import matplotlib.pyplot as plt
import math
from cmcrameri import cm

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from converter import d_norm_to_phi
from fluxes import shell_diss, shell_out, Rayleigh

#####Plot of intersecting fluxes#####
```

```python
def fluxes(eta_tidal, eta_conv, scale='True', d=np.linspace(R*1e-3, R, 10000)):
    """Plot the heat fluxes and Rayleigh number as a function of normalised shell thickness (or melt fraction).

    Parameters
    ----------
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')
    d : numpy.ndarray
        Shell thickness [m]
    """
    #####Parameters#####
    stable_point = None
    unstable_point = None
    critical_normalised_thickness = None

    #####Colors#####
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)
    c_teal = cm.roma(0.6)

    #####Create figure #####
    fig = plt.figure(figsize=(8, 10))
    #Adjust to make space for the x axis labels
    grid = plt.GridSpec(3, 1, height_ratios=[1, 1, 0.00005], hspace=0.3, wspace=0.3)
    #Single plot with space below for extended x labels

    #Add subplots
    axb = fig.add_subplot(grid[1, 0])  #bottom
    axa = fig.add_subplot(grid[0, 0], sharex=axb)  #top, sharing x-axis with axb
    plt.subplots_adjust(left=0.15, right=0.9, top=0.95)

    ######Calculate data######
    #Normalise thickness
    d_norm = d/R

    #1 - H(d) and q(d)
    #Calculate fluxes
```

```
    h = []; q = []

    for i, di in enumerate(d): #Iterating because it gets unhappy about the Rayleigh number check
        h.append(shell_diss(di, eta_tidal, scale)[0]) #W
        q.append(shell_out(di, eta_tidal, eta_conv, scale)[0]) #W

    #Convert to numpy arrays
    h = np.array(h)
    q = np.array(q)

    #2 - Ra(d)
    Ra = Rayleigh(d, eta_tidal, eta_conv, scale)

    ######Plotting######
    #Plot 1 (above)
    axa.plot(d_norm, h, color = c_red, label = 'Tidal dissipation')
    axa.plot(d_norm, q, color = c_blue, label = 'Outward heat flux')

    #Find crossover points
    flux = h - q

    for i in range(1, len(flux)):
        if flux[i-1] * flux[i] <= 0:  #Look for sign changes (+ to - or - to +)
            if flux[i-1] - flux [i] > 0: #If the sign change is from + to - is unstable
                axa.scatter(d_norm[i], h[i], color='black', marker = 'o', edgecolors='black', zorder=5, label = f'Unstable point')
                #Save the unstable point
                unstable_point = f"{d_norm[i]*100:.1f}%"
            else:
                axa.scatter(d_norm[i], h[i], color='white', marker = 'o', edgecolors='black', zorder=5, label = f'Stable point')
                #Save the stable point
                stable_point = f"{d_norm[i]*100:.1f}%"

    #Plot 2 (below)
    axb.plot(d_norm, Ra, label = 'Shell Ra', color=c_teal)

```

```
    #Find critical Ra
    axb.axhline(y=1e3, color='k', linestyle='-', label='Critical Ra')
    if np.any(Ra >= 1e3):
        index = np.where(Ra >= 1e3)[0][0]
        axb.axvline(x=d_norm[index], color='k', linestyle='--', label=f'Critical $d/R$')
        axa.axvline(x=d_norm[index], color='k', linestyle='--')
        #Save the critical normalised thickness
        critical_normalised_thickness = f"{d_norm[index]*100:.1f}%"
        critical_Ra = "1000"

    ####Formatting######
    wordsize=16
    legendsize=14
    widthsize=2
    #Lines
    for ax in [axa, axb]:
        #Lines
        for line in ax.get_lines():
            line.set_linewidth(2)
        #Dots
        for scatter in ax.collections:
            scatter.set_sizes([200])

    #Legends
    legend1 = axa.legend(loc='lower right', fontsize=legendsize, title_fontsize=legendsize, framealpha=1)
    legend2 = axb.legend(loc='lower right', fontsize=legendsize, title_fontsize=legendsize, framealpha=1)

    axa.add_artist(legend1)
    axb.add_artist(legend2)

    ##########Y Axes##########
    #Y axis
    axa.set_yscale('log')
    axa.set_ylabel('Heat flux (W)', fontsize=wordsize)
    axa.tick_params(axis='y', labelsize=wordsize, width=widthsize)

    axb.set_yscale('log')
    axb.set_ylabel('Rayleigh number', fontsize=wordsize)
```

```
    axb.tick_params(axis='both', labelsize=wordsize, width=
       widthsize)

    #########X Axes##########
    #Shared x axis
    axa.tick_params(axis='x', which='both',
                    bottom=True, top=False, labelbottom=False
                    ,width=widthsize)
    axa.sharex(axb)
    ax1 = axb

    #d/R
    ax1.set_xscale('log')
    ax1.set_xlabel('Thickness of shell / Radius, $d/R$', color=
       c_blue, fontsize=wordsize)
    ax1.tick_params(axis='x', colors=c_blue, labelsize=wordsize,
        width=widthsize)

    #phi
    ax2 = ax1.twiny()
    ax2.set_xscale('log')

    #Limits
    d_min, d_max = ax1.get_xlim()
    ax2.set_xlim(d_min, d_max)

    #Convert
    d_norm_values = [1e0, 1e-1, 1e-2, 1e-3]
    phi_values = [d_norm_to_phi(d) for d in d_norm_values]

    #Ticks
    ax2.set_xticks(d_norm_values) #ticks line up with d/R
    ax2.set_xticklabels(f'{phi:.3f} ' for phi in phi_values) #
       but have value of phi

    #Axis Label
    ax2.set_xlabel('Melt fraction, $\phi$', color=c_red,
       fontsize=wordsize)

    #Style secondary axis
    ax2.xaxis.set_ticks_position('bottom')
    ax2.xaxis.set_label_position('bottom')
    ax2.tick_params(axis='x', colors=c_red, which='both',
       labelsize=wordsize, width=widthsize)

```

```
    #Spines
    ax2.spines['bottom'].set_color(c_red)
    ax2.spines['bottom'].set_linewidth(widthsize/2)

    #Shifting up and down
    ax1.xaxis.set_label_coords(0.5, -0.1) #dR label
    ax2.spines['bottom'].set_position(('outward', 49)) #Melt
        fraction spines
    ax2.xaxis.set_label_coords(0.5, -0.3)  #Melt fraction label

    #Save
    plt.savefig(f'plot/scale={scale}/Fluxes.png', dpi=500)
    with open(f'plot/scale={scale}/Fluxes.txt', 'w') as f:
        f.write(f'eta_tidal = 1e{math.log10(eta_tidal)}\n')
        f.write(f'eta_conv = 1e{math.log10(eta_conv)}\n')

    plt.show()
```

```
#Plot of intersecting fluxes, with and without scaling the shear
    modulus
#Import libraries
import numpy as np
import matplotlib.pyplot as plt
import math
from matplotlib.lines import Line2D
import seaborn as sns
from cmcrameri import cm

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from converter import d_norm_to_phi
from fluxes import shell_diss, shell_out, Rayleigh


#####Plot of intersecting fluxes#####
def fluxes_scale(eta_tidal, eta_conv, d=np.linspace(R*1e-3, R,
    10000)):
    """Plot the fluxes in the shell as a function of normalised
        thickness (or melt fraction).
    With and without scaling the shear modulus.

    Parameters
    ----------
```

```
    eta_tidal : float
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    d : numpy.ndarray
        Shell thickness [m]
    """
    #####Parameters#####
    #True and False are used to indicate whether the shear
       modulus is scaled or not
    stable_point_True = None
    stable_point_False = None
    unstable_point_True = None
    unstable_point_False = None
    critical_normalised_thickness_True = None
    critical_normalised_thickness_False = None

    #####Colors#####
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)

    #####Create figure#####
    fig = plt.figure(figsize=(10, 8))
    #Adjust to make space for the x axis labels
    grid = plt.GridSpec(2, 1, height_ratios=[6.99995, 0.00005],
       hspace=0.3)  #Single plot with space below for extended x
        labels
    ax = fig.add_subplot(grid[0, 0])
    plt.subplots_adjust(left=0.1, right=0.975, top=0.975)

    ######Calculate data######
    #Normalise thickness
    d_norm = d/R

    #Calculate fluxes
    for scale in ['True', 'False']:
        h = []; q = []

        for di in d:  #Iterating because it gets unhappy about
           the Rayleigh number check
            h.append(shell_diss(di, eta_tidal, scale)[0])
            q.append(shell_out(di, eta_tidal, eta_conv, scale)
               [0])

        h = np.array(h)
```

```
        q = np.array(q)

        #Plot
        plt.plot(d_norm, h, color=c_red, linestyle='-' if scale == 'True' else ':')
        plt.plot(d_norm, q, color=c_blue, linestyle='-' if scale == 'True' else ':')

        #Find crossover points
        flux = h - q

        for i in range(1, len(flux)):
            if flux[i-1] * flux[i] <= 0:  #Look for sign changes (+ to - or - to +)
                if flux[i-1] - flux [i] > 0: #If the sign change is from + to - is unstable
                    ax.scatter(d_norm[i], h[i], color='black', s=300, marker = 'o', edgecolors='black', zorder=5, label = 'Unstable point')
                    #Save unstable point
                    unstable_point_True = f"{d_norm[i]*100:.1f}%" if scale == 'True' else unstable_point_True
                    unstable_point_False = f"{d_norm[i]*100:.1f}%" if scale == 'False' else unstable_point_False
                else:
                    ax.scatter(d_norm[i], h[i], color='white', s=300, marker = 'o', edgecolors='black', zorder=5, label = 'Stable point')
                    #Save stable point
                    stable_point_True = f"{d_norm[i]*100:.1f}%" if scale == 'True' else stable_point_True
                    stable_point_False = f"{d_norm[i]*100:.1f}%" if scale == 'False' else stable_point_False


    #Calculate Ra
    scale_lines = [] #To store the lines for the legend
    scale_labels = []

    for i, scale in enumerate(['True', 'False']):
        Ra = np.array([Rayleigh(di, eta_tidal, eta_conv, scale) for di in d])  #Compute Ra for each thickness
```

```
        if np.any(Ra >= 1e3):
            index = np.where(Ra >= 1e3)[0][0]
            line = ax.axvline(x=d_norm[index], color='k', linestyle='-' if scale == 'True' else ':')

            #Save critical normalised thickness
            critical_normalised_thickness_True = f"{d_norm[index]*100:.1f}%" if scale == 'True' else critical_normalised_thickness_True
            critical_normalised_thickness_False = f"{d_norm[index]*100:.1f}%" if scale == 'False' else critical_normalised_thickness_False
            critical_Ra = "1000"

            #Save line for the legend
            scale_lines.append(line)
            scale_labels.append(f'{scale}')

    ####Formatting######
    wordsize=16
    legendsize=14
    widthsize=2
    #Lines
    for line in ax.get_lines():
        line.set_linewidth(2)
    #Dots
    for scatter in ax.collections:
        scatter.set_sizes([200])
    #Legends
    legend1 = ax.legend(scale_lines, scale_labels, loc='lower right',
                        title='$\lambda$', fontsize=legendsize, title_fontsize=legendsize)
    ax.add_artist(legend1)#For scale vs not scaled

    flux_lines = [
        Line2D([0], [0], color=c_red, label='Tidal dissipation'),
        Line2D([0], [0], color=c_blue, label='Outward heat flux'),
        Line2D([0], [0], color='black', label='Critical $d/R$')
    ]
    legend2 = ax.legend(handles=flux_lines, loc='lower left', fontsize=legendsize,
                        title_fontsize=legendsize, framealpha=1)
```

```
    ax.add_artist(legend2)#For fluxes etc


    #Y axis
    ax.set_yscale('log')
    ax.set_ylabel('Heat flux (W)', fontsize=wordsize)
    ax.tick_params(axis='y', labelsize=wordsize, width=widthsize
        )

    ##########X Axes##########
    ax1 = ax

    #d/R
    ax1.set_xscale('log')
    ax1.set_xlabel('Thickness of shell / Radius, $d/R$', color=
        c_blue, fontsize=wordsize)
    ax1.tick_params(axis='x', colors=c_blue, labelsize=wordsize,
         width=widthsize)

    #phi
    ax2 = ax1.twiny()
    ax2.set_xscale('log')

    #Limits
    d_min, d_max = ax1.get_xlim()
    ax2.set_xlim(d_min, d_max)

    #Convert
    d_norm_values = [1e0, 1e-1, 1e-2, 1e-3]
    phi_values = [d_norm_to_phi(d) for d in d_norm_values]

    #Ticks
    ax2.set_xticks(d_norm_values) #ticks line up with d/R
    ax2.set_xticklabels(f'{phi:.3f} ' for phi in phi_values) #
        but have value of phi

    #Axis Label
    ax2.set_xlabel('Melt fraction, $\phi$', color=c_red,
        fontsize=wordsize)

    #Style secondary axis
    ax2.xaxis.set_ticks_position('bottom')
    ax2.xaxis.set_label_position('bottom')
    ax2.tick_params(axis='x', colors=c_red, which='both',
        labelsize=wordsize, width=widthsize)
```

```python

    #Spines
    ax2.spines['bottom'].set_color(c_red)
    ax2.spines['bottom'].set_linewidth(widthsize/2)

    #Shifting up and down
    ax1.xaxis.set_label_coords(0.5, -0.06) #dR label
    ax2.spines['bottom'].set_position(('outward', 49)) #Melt fraction spines
    ax2.xaxis.set_label_coords(0.5, -0.16)  #Melt fraction label

    #Save
    plt.savefig(f'plot/Fluxes_scale.png', dpi=500)

    with open(f'plot/Fluxes_scale.txt', 'w') as f:
        f.write(f'eta_tidal = 1e{math.log10(eta_tidal)}\n')
        f.write(f'eta_conv = 1e{math.log10(eta_conv)}\n')
        f.write(f'Critical Rayleigh number = {critical_Ra}\n')
        f.write(f'd/R = True, Stable point at d/R = {stable_point_True}\n')
        f.write(f'd/R = True, Unstable point at d/R = {unstable_point_True}\n')
        f.write(f'd/R = True, Critical normalised thickness = {critical_normalised_thickness_True}\n')
        f.write(f'd/R = False, Stable point at d/R = {stable_point_False}\n')
        f.write(f'd/R = False, Unstable point at d/R = {unstable_point_False}\n')
        f.write(f'd/R = False, Critical normalised thickness = {critical_normalised_thickness_False}\n')
```

```python
#Plot of heating regime in the shell as a function of shell thickness and tidal viscosity
#Import modules
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
import math
from cmcrameri import cm

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from converter import d_norm_to_phi
```

```
from fluxes import shell_diss, shell_out

#####Plot of heating regime#####
def heating(scale, eta_tidal = np.logspace(12, 17, 24), eta_conv
    = 1e19, d = np.linspace(R*1e-3, R, 1000)):
    """Plot the heating regime in the shell, as a function of
       shell thickness and Tidal viscosity.

    Parameters
    ----------
    scale : str
        Scale factor for the shear modulus ('True' or 'False')
    eta_tidal : numpy.ndarray
        Tidal viscosity [Pa.s]
    eta_conv : float
        Convective viscosity [Pa.s]
    d : numpy.ndarray
        Shell thickness [m]
    """
    #####Colors#####
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)

    #####Create figure#####
    fig = plt.figure(figsize=(10, 8))
    #Adjust to make space for the x axis labels
    grid = plt.GridSpec(2, 1, height_ratios=[6.99995, 0.00005],
       hspace=0.3)
    ax = fig.add_subplot(grid[0, 0])
    plt.subplots_adjust(left=0.1, right=0.975, top=0.95)

    #####Calculate data######
    E = np.zeros((len(eta_tidal), len(d)))
    Q = np.zeros((len(eta_tidal), len(d)))

    for i, eta_tidal_i in enumerate(eta_tidal):
        for j, di in enumerate(d):
            E[i,j] = shell_diss(d = di, eta_tidal=eta_tidal_i,
               scale = scale)[0] #W
            Q[i,j] = shell_out(d = di, eta_tidal=eta_tidal_i,
               eta_conv=eta_conv, scale = scale)[0] #W

    #Total heat flux
    F = E - Q
    norm = Normalize(vmin=-1e14, vmax=1e14) #Clip the data to
```

```
    clearly show heating and cooling
F = np.clip(F, norm.vmin, norm.vmax)

#####Plot data#####
#Colour map
cf = plt.contourf(d/R, eta_tidal, F, cmap=cm.vik,
                  levels=10000, extend='both')

#Contour of q=H
contour1 = plt.contour(d/R, eta_tidal, F, levels=[0], colors
    ='black',
                       linestyles='dashed', label = 'q = H',
                       )

#Maximum heating
max_h_index = np.unravel_index(np.argmax(F, axis=None), F.
    shape)
max_h_d = d[max_h_index[1]]
max_h_eta = eta_tidal[max_h_index[0]]

####Formatting######
wordsize=16
widthsize=2
linewidth=2
#Lines
for line in ax.get_lines():
    line.set_linewidth(linewidth)
#Legends
plt.clabel(contour1, inline=True, fontsize=wordsize,
           fmt={0: '$\dot{E}_V = q_A$'}, manual=[(0.2, 1e15)
               ],
           inline_spacing=15)

#Colorbar
cbar = plt.colorbar(cf, label='Total heat flux (W)')
cbar.ax.set_ylabel('Total heat flux (W)', fontsize=wordsize)
cbar.ax.tick_params(labelsize=wordsize, width=widthsize)
cbar.ax.yaxis.offsetText.set_fontsize(wordsize)

#Y axis
plt.yscale('log')
plt.ylabel(r'Tidal viscosity, $\eta_{tidal}$ (Pa s)',
    fontsize=wordsize)
plt.tick_params(axis='y', labelsize=wordsize, width=
    widthsize)
```

```


    ##########X Axis##########
    ax1 = ax

    #d/R
    ax1.set_xscale('log')
    ax1.set_xlabel('Thickness of shell / Radius, $d/R$', color=
        c_blue, fontsize=wordsize)
    ax1.tick_params(axis='x', colors=c_blue, labelsize=wordsize,
        width=widthsize)

    #phi
    ax2 = ax1.twiny()
    ax2.set_xscale('log')

    #Limits
    d_min, d_max = ax1.get_xlim()
    ax2.set_xlim(d_min, d_max)

    #Convert
    d_norm_values = [1e0, 1e-1, 1e-2, 1e-3]
    phi_values = [d_norm_to_phi(d) for d in d_norm_values]

    #Ticks
    ax2.set_xticks(d_norm_values) #ticks line up with d/R
    ax2.set_xticklabels(f'{phi:.3f} ' for phi in phi_values) #
        but have value of phi

    #Axis Label
    ax2.set_xlabel('Melt fraction, $\phi$', color=c_red,
        fontsize=wordsize)

    #Style secondary axis
    ax2.xaxis.set_ticks_position('bottom')
    ax2.xaxis.set_label_position('bottom')

    ax2.spines['bottom'].set_color(c_red)
    ax2.spines['bottom'].set_linewidth(widthsize/2)
    ax2.tick_params(axis='x', colors=c_red, which='both',
        labelsize=wordsize, width=widthsize)

    #Shifting up and down
    ax1.xaxis.set_label_coords(0.5, -0.06) #dR label
    ax2.spines['bottom'].set_position(('outward', 49)) #Melt
```

```
    fraction spines
    ax2.xaxis.set_label_coords(0.5, -0.17) #Melt fraction label

    #Save
    plt.savefig(f'plot/scale={scale}/Heat_regime.png', dpi=500)

    with open(f'plot/scale={scale}/Heat_regime.txt', 'w') as f:
        f.write(f'eta_conv = 1e{math.log10(eta_conv)}\n')
        f.write(f'Max at $\eta_{{tidal}}$ = {max_h_eta:.1e} and
            d/R = {max_h_d/R:.2f} \n')

    plt.show()
```

```
#Plot of stable melt fraction as a function of the tidal and
    convective viscosities
#Import libraries
import pandas as pd
import matplotlib.pyplot as plt
from cmcrameri import cm
import matplotlib.colors as colors

#Import parameters
from heat_evolution.param.parameters_io import R

#Import functions
from converter import d_norm_to_phi

def viscosities(scale='True', eta_conv=1e21):
    """
    Plot of stable melt fraction as a function of the tidal and
        convective viscosities.

    Parameters
    ----------
    scale : str, optional
        Scale factor for the shear modulus ('True' or 'False').
            The default is 'True'.
    eta_conv : float, optional
        Convective viscosity. The default is 1e21.
    """
    ######Create figure######
    fig = plt.figure(figsize=(14, 9))
    #Adjust for colorbar
    grid = plt.GridSpec(2, 2,
                        height_ratios=[1, 2], width_ratios=[7,
```

```
                                 1],
                              hspace=0.2, wspace=0.05)
    ax1 = fig.add_subplot(grid[0, 0])
    ax2 = fig.add_subplot(grid[1, 0])
    plt.subplots_adjust(left=0.1, right=0.9, top=0.975)

    #####Colors#####
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)

    #####Bottom plot#####
    #Read data
    data = pd.read_csv(f'Crossover_points_scale_{scale}.csv')
    data_stable = data[data['Type'] == 'Stable']
    data_none = data[(data['Type'] != 'Stable') & (data['Type'] != 'Unstable')]

    #Limit the none points plotted
    #Just one row above and one row below the stable points
    #Requires convective viscosity to be in orders of magnitude
    if len(data_none) > 0:
        #Convective extremes
        conv_data_stable_max = data_stable['Convective viscosity'].max()
        conv_data_stable_min = data_stable['Convective viscosity'].min()

        data_none = data_none[
             (data_none['Convective viscosity'] <= 10 * conv_data_stable_max) &
                (data_none['Convective viscosity'] >= 0.1 * conv_data_stable_min)
        ]

        #Tidal extremes
        diss_data_stable_max = data_stable['Tidal viscosity'].max()
        diss_data_stable_min = data_stable['Tidal viscosity'].min()

        data_none = data_none[
                (data_none['Tidal viscosity'] <= 10 * diss_data_stable_max) &
                    (data_none['Tidal viscosity'] >= 0.1 * diss_data_stable_min)
```

```
                 ]

    #Plot stable melt fraction & none points
    stable_d_norm = data_stable['Thickness'] / R
    norm = colors.LogNorm(vmin=stable_d_norm.min(), vmax=stable_d_norm.max()) #Log-scale for colormap

    scatter = ax2.scatter(data_stable['Tidal viscosity'], data_stable['Convective viscosity'],
                                        c=stable_d_norm, cmap=cm.roma,
                                        alpha=0.8, marker='o', s=300,
                                        norm=norm)
    ax2.scatter(data_none['Tidal viscosity'], data_none['Convective viscosity'],
                                             color='black',
                                             alpha=0.8,  marker='x', s = 100, label='Cools')

    #Cross-section line
    ax2.axhline(y=eta_conv, color='k', linestyle=':')

    #####Top plot#####
    #Get data for cross-section
    #Limit the none points plotted
    diss_min, diss_max = data_none['Tidal viscosity'].min(), data_none['Tidal viscosity'].max()
    #Just for one convective viscosity
    topset_stable = data[(data['Type'] == 'Stable') & (data['Convective viscosity'] == eta_conv)
                                 & (data['Tidal viscosity'] >= diss_min)
                                     & (data['Tidal viscosity'] <= diss_max)]
    topset_none = data[(data['Type'] != 'Stable') & (data['Type'] != 'Unstable') & (data['Convective viscosity'] == eta_conv)
                               & (data['Tidal viscosity'] >= diss_min) & (data['Tidal viscosity'] <= diss_max)
                                   ]
    topset_none['Thickness'] = R #Fully cooled

    #Plot equilbrium melt fraction & none-points
    topset = pd.concat([topset_stable, topset_none]).sort_values('Tidal viscosity')

    ax1.plot(topset['Tidal viscosity'], topset['Thickness']/R,
```

```
                    color='k', linestyle=':')

    ax1.scatter(topset_stable['Tidal viscosity'], topset_stable[
        'Thickness']/R,
                                        c=topset_stable['Thickness']/R,
                                            cmap=cm.roma,
                                        alpha=0.8, marker='o', s=300,
                                        norm=norm)
    ax1.scatter(topset_none['Tidal viscosity'], topset_none['
        Thickness']/R,
                                             color='black',
                                             alpha=0.8,  marker='x', s =
                                                 100, label='Cools')


    ######Formatting######
    wordsize=16
    legendsize=14
    widthsize=2
    linewidth=2
    #Lines
    for line in plt.gca().get_lines():
        line.set_linewidth(linewidth)
    #Legend
    ax2.legend(loc='lower left', framealpha=1,
                fontsize=legendsize, edgecolor='black')

    ##########Color Bar##########
    #Positioning colorbar
    pos_bottom = ax2.get_position()
    cax = fig.add_axes([pos_bottom.x1 + 0.08,
                              pos_bottom.y0,
                              0.03,
                              pos_bottom.height])

    #Create colorbar with d/R scale
    cbar_d = fig.colorbar(scatter, cax=cax, location='right',
        pad=0.1)

    #Ticks
    d_ticks = [0.02, 0.1, 0.2, 0.4]
    cbar_d.set_ticks(d_ticks)
    cbar_d.set_ticklabels([f'{tick}' for tick in d_ticks])

    #Keep d/R on the left side
```

```
    cbar_d.set_label('Stable $d/R$', color=c_blue, fontsize=wordsize)
    cbar_d.ax.yaxis.set_label_position('left')
    cbar_d.ax.tick_params(axis='y', colors=c_blue, labelsize=wordsize, width=widthsize)
    cbar_d.ax.yaxis.set_ticks_position('left')

    #Calculate phi values
    phi_ticks = [d_norm_to_phi(d) for d in d_ticks] #convert to phi

    #Add second scale directly to the existing colorbar
    cbar_phi = cbar_d.ax.secondary_yaxis('right')
    cbar_phi.tick_params(axis='y', color=c_red, labelsize=wordsize, width=widthsize)
    cbar_phi.yaxis.set_tick_params(labelcolor=c_red)

    cbar_phi.set_ylabel('Stable $\phi$', color=c_red, fontsize=wordsize)

    cbar_phi.set_yticks(d_ticks) #Ticks at d
    cbar_phi.set_yticklabels([f'{phi:.2f}' for phi in phi_ticks]) #But values are phi


    #Find all colorbar axes and remove any that are not our main one
    for ax in ax2.figure.axes:
        if ax != cbar_d.ax and ax != cbar_phi and isinstance(ax, plt.matplotlib.colorbar.ColorbarBase):
            ax.remove()

  ##########X Axes##########
    #Bottom plot
    ax2.set_xscale('log')
    ax2.set_xlabel('Tidal viscosity, $\eta_{tidal}$ (Pa s)', fontsize=wordsize)
    ax2.tick_params(axis='x', labelsize=wordsize, width=widthsize)

    #Top plot
    ax1.set_xscale('log')
    ax1.tick_params(axis='x', length=widthsize, width=widthsize)
    plt.setp(ax1.get_xticklabels(), visible=False)

```

```python

    ##########Y Axes##########
    #Bottom plot
    ax2.set_yscale('log')
    ax2.set_ylabel('Convective viscosity, $\eta_{conv}$ (Pa s)', fontsize=wordsize)
    ax2.tick_params(axis='y', labelsize=wordsize, width=widthsize)

    #Top plot
    #d/R
    d_ticks = [0.02, 0.1, 0.4]
    phi_ticks = [d_norm_to_phi (d) for d in d_ticks] #convert to phi

    ax1.set_yscale('log')
    ax1.set_ylabel('Stable $d/R$', fontsize=wordsize, color=c_blue)
    ax1.spines['left'].set_color(c_blue)
    ax1.tick_params(axis='y', colors=c_blue, labelsize=wordsize, width=widthsize)

    ax1.set_yticks(d_ticks)
    ax1.set_yticklabels([f'{tick}' for tick in d_ticks])

    #Phi
    ax3 = ax1.twinx()
    ax3.set_yscale('log')

    #Limits
    d_min, d_max = ax1.get_ylim()
    ax3.set_ylim(d_min, d_max)

    #Values
    ax3.set_yticks(d_ticks) #ticks at d
    ax3.set_yticklabels([f'{phi:.2f}' for phi in phi_ticks], color=c_red) #but values are phi

    #Tick formatting
    ax3.yaxis.set_ticks_position('right')
    ax3.yaxis.set_tick_params(color=c_red, labelsize=wordsize, width=widthsize)

    #Spine
    ax3.spines['right'].set_color(c_red)
```

```
    ax3.spines['right'].set_linewidth(widthsize/2)

    #Label
    ax3.set_ylabel('Stable $\phi$', color=c_red, fontsize=wordsize)
    ax3.yaxis.set_label_position('right')

    ax1.spines['left'].set_color(c_blue) #Need to be set again because it breaks

    #Save
    plt.savefig(f'plot/scale={scale}/Effect_viscosities_full.png', dpi=500)
    plt.show()
```

```
#Plot of equilibrium melt fraction, with results of heat evolution runs.

#Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import griddata
from cmcrameri import cm
import matplotlib.colors as colors
import math

#Import parameters
from heat_evolution.param.parameters_io import R
from matplotlib.patches import Patch

#Import functions
from converter import d_norm_to_phi, phi_to_d_norm

#####Plot#####
def plot(d=np.linspace(R*1e-3, R, 1000)):
    """
    Plot the heat evolution data with the theoretical data.

    Parameters
    ----------
    d : array
        Array of initial thicknesses to plot. Default is np.linspace(R*1e-3, R, 1000). [m]
    """
```

```
    #####Heat evolution data#####
    #Read data
    data_evol = pd.read_csv(f'runs/Heat_Evolution.csv') #scale is True

    #Get the parameters
    scale = data_evol['scale'].iloc[0] #In case it is not True
    eta_conv = data_evol['eta_conv'].iloc[0]

    #####Theoretical data#####
    #Read data
    data_theory = pd.read_csv(f'Crossover_points_scale_{scale}.csv')
    #Mask convective viscosity
    data_theory = data_theory[data_theory['Convective viscosity'] == eta_conv]

    #Lists to store data
    data_rows = []

    #Find each unique tidal viscosity
    viscosities = sorted(data_theory['Tidal viscosity'].unique())
    #Markers to limit how many cooling rows to add
    hot_row = None
    second_cool_row = None

    #Calculate final thicknesses
    for eta_tidal_i in viscosities:
        #Get the stable thickness
        stable_mask = (data_theory['Tidal viscosity'] == eta_tidal_i) & (data_theory['Type'] == 'Stable')
        stable_d = data_theory[stable_mask]['Thickness'].values[0] if stable_mask.any() else None

        #Get the unstable thickness
        unstable_mask = (data_theory['Tidal viscosity'] == eta_tidal_i) & (data_theory['Type'] == 'Unstable')
        unstable_d = data_theory[unstable_mask]['Thickness'].values[0] if unstable_mask.any() else None

        #Limit how many cooling rows are added
        if stable_d is None:
            if hot_row == None:  #If havent had a hot row yet
                first_cool_row = eta_tidal_i #Will overwrite
```

```
                    itself until start finding hot rows
                    #Results in one cool row below the first hot row
                else: #If have had a hot row, save the second cool_row
                    second_cool_row = eta_tidal_i if second_cool_row is None else second_cool_row
                    #But don't overwrite, just save the first
                    #Results in one cool row above the last hot row

        #If have crossover points, determine the equilibrium thickness
        else:
            hot_row = True #If have found a stable row, set hot_row to True
            for di in d:
                #Scale to d/R
                initial_d_norm = di/R

                #If above critical thickness, will cool
                if unstable_d is not None and di > unstable_d:
                    final_d_norm = 1
                #If below critical thickness, will reach stable d
                else:
                    final_d_norm = stable_d/R

                #Save
                data_rows.append({
                    'initial_d_norm': initial_d_norm,
                    'eta_tidal': eta_tidal_i,
                    'final_d_norm': final_d_norm
                })
    #Save the first cool row
    for di in d:
        data_rows.append({
            'initial_d_norm': di/R,
            'eta_tidal': first_cool_row,
            'final_d_norm': 1
        })
    #Save the final cool row
    for di in d:
        data_rows.append({
            'initial_d_norm': di/R,
            'eta_tidal': second_cool_row,
            'final_d_norm': 1
```

```
        })

    #Make data_rows a data frame for pcolormesh to work
    df = pd.DataFrame(data_rows)


    #####Create heatmap#####
    #Grid to colour on
    x_grid = np.logspace(np.log10(df[df['final_d_norm'] < 1]['initial_d_norm'].min()),
                         np.log10(df[df['final_d_norm'] < 1]['initial_d_norm'].max()), 500)
    y_grid = np.logspace(np.log10(df[df['final_d_norm'] < 1]['eta_tidal'].min()),
                         np.log10(df[df['final_d_norm'] < 1]['eta_tidal'].max()), 500)
    X, Y = np.meshgrid(x_grid, y_grid)

    #First interpolate over entire area
    Z = griddata((df['initial_d_norm'], df['eta_tidal']), df['final_d_norm'],
                 (X, Y), rescale=True, method='linear')    # Options are nearest, linear, cubic

    #Then mask Z to remove values where final_d_norm is 1
    mask = griddata((df['initial_d_norm'], df['eta_tidal']), df['final_d_norm'] == 1,
                    (X, Y), method='nearest')
    Z_masked = np.where(mask, np.nan, Z)

    #Log-scale for Z
    norm = colors.LogNorm(vmin=df['final_d_norm'].min(),
                          vmax=df['final_d_norm'][df['final_d_norm'] < 1].max())

    ######Create figure######
    fig = plt.figure(figsize=(11, 8))
    #Adjust to fit colorbar and x axis labels
    grid = plt.GridSpec(2, 1, height_ratios=[6.99995, 0.00005],
                        hspace=0.3)  #Single plot with space below for extended x labels
    ax = fig.add_subplot(grid[0, 0])
    plt.subplots_adjust(left=0.1, right=0.95, top=0.95)

    #####Colors#####
```

```
    c_red = cm.roma(0)
    c_blue = cm.roma(0.9)
    c_max = c_blue

    ##########Plot data##########
    #Background will be same as cooling points
    ax.set_facecolor(c_max)
    ax.set_xlim(df['initial_d_norm'].min(), df['initial_d_norm'
        ].max()) #Limit to relevant region
    ax.set_ylim(df['eta_tidal'].min(), df['eta_tidal'].max())

    #Colormap
    cmap = colors.LinearSegmentedColormap.from_list(
        'roma_limited', cm.roma(np.linspace(0, 0.8, 256))) #
            Doesn't go all the way to dark blue (fully cooled)
    #Plot
    im = plt.pcolormesh(X, Y, Z_masked, cmap=cmap, shading='auto
        ',
                                norm=norm)

    ####Formatting######
    wordsize=16
    linewidth=2
    widthsize=2
    scattersize=200

    ######Plot heat evolution points#####
    for index, row in data_evol.iterrows():
        eta_tidal = row['eta_tidal']
        initial_melt_fraction = row['Initial Melt Fraction']
        initial_d_norm = phi_to_d_norm(row['Initial Melt
            Fraction'])
        final_d_norm = phi_to_d_norm(row['Final Melt Fraction'])

        ax.scatter(initial_d_norm, eta_tidal, c=final_d_norm if
            final_d_norm < 1 else c_max,
                    s=scattersize, edgecolor='black', linewidth=
                        linewidth,
                    marker = 'o' if initial_melt_fraction >= 0.5
                        else 's', #Shape depends on hot vs cold
                        start
                     cmap=cmap if final_d_norm < 1 else None, #If
                          fully cooled, don't use colormap, uses
                         background
                     norm=norm)
```

```

    #######Legend######
    #To explain the background
    fully_cooled_patch = [Patch(facecolor=c_max, edgecolor='none',
                                label='Fully cooled (d/R = 1)')
                                ]
    legend1 = ax.legend(handles=fully_cooled_patch, loc='lower right', fontsize=wordsize,
                frameon=True, framealpha=0.8, facecolor='white', edgecolor='none')

    #For heat evolution runs
    hot_vs_cold = [
            plt.Line2D([], [], label='Hot start', color='none', linestyle='None',
                       marker = 'o', markeredgecolor='black',
                       markeredgewidth=linewidth, markersize=scattersize/20),

            plt.Line2D([], [], label='Cold start', color='none', linestyle='None',
                       marker = 's', markeredgecolor='black',
                       markeredgewidth=linewidth, markersize=scattersize/20)
                       ]

    legend2 = ax.legend(handles=hot_vs_cold, loc='lower left', fontsize=wordsize,
                        frameon=True, framealpha=0.8, facecolor='white', edgecolor='none')

    #Add the legends to the plot
    ax.add_artist(legend1)
    ax.add_artist(legend2)

    ##########Color Bar##########
    cbar_d = plt.colorbar(im)
    cbar_d.mappable.set_clim(vmin=df['final_d_norm'].min(),
                             vmax=df['final_d_norm'][df['final_d_norm'] < 1].max())
    #Limited to stable values, not fully cooled

    #Ticks
    #Include nice reference d/R values
```

```
    d_ticks = [df['final_d_norm'].min(), 0.02, 0.05, df['final_d_norm'][df['final_d_norm'] < 1].max()]
    d_ticks.sort()
    cbar_d.set_ticks(d_ticks)
    cbar_d.set_ticklabels([f'{d:.2f}' for d in d_ticks], color=c_blue)
    cbar_d.ax.minorticks_off()

    #d/R
    cbar_d.set_label('Stable $d/R$', color=c_blue, fontsize=wordsize)
    cbar_d.ax.yaxis.set_label_position('left')
    cbar_d.ax.yaxis.set_tick_params(color=c_blue, labelcolor=c_blue, labelsize=wordsize, width=widthsize)
    cbar_d.ax.yaxis.set_ticks_position('left')

    #phi
    cbar_phi = cbar_d.ax.secondary_yaxis('right')

    #Convert
    phi_ticks = [d_norm_to_phi(d) for d in d_ticks]

    #Ticks
    cbar_phi.set_yticks(d_ticks) #Ticks at d
    cbar_phi.set_yticklabels([f'{phi:.2f}' for phi in phi_ticks], color=c_red) #But values are phi
    cbar_phi.minorticks_off()

    #Axis label
    cbar_phi.set_ylabel('Stable $\phi$', color=c_red, fontsize=wordsize)

    #Style secondary axis
    cbar_phi.tick_params(axis='y', colors=c_red, labelsize=wordsize, width=widthsize)

    #Find all colorbar axes and remove any that are not our main one
    for ax in plt.gcf().get_axes():
        if ax != cbar_d.ax and ax != cbar_phi and isinstance(ax, plt.matplotlib.colorbar.ColorbarBase):
            ax.remove()

    #Move cbar to the right
    cbar_d.ax.set_position([cbar_d.ax.get_position().x0 + 0.05,
```

```
                    cbar_d.ax.get_position().y0,
                    cbar_d.ax.get_position().width,
                    cbar_d.ax.get_position().height])


    ##########Y Axis##########
    plt.yscale('log')
    plt.ylabel('Tidal viscosity, $\eta_{tidal}$ (Pa s)', fontsize=wordsize)
    plt.tick_params(axis='y', labelsize=wordsize, width=widthsize)

    ##########X Axes##########
    ax1 = plt.gca()

    #d/R
    ax1.set_xscale('log')
    ax1.set_xlabel('Initial thickness of shell / Radius, $d/R$', color=c_blue, fontsize=wordsize)
    ax1.tick_params(axis='x', colors=c_blue, labelsize=wordsize, width=widthsize)

    #phi
    ax2 = ax1.twiny()
    ax2.set_xscale('log')

    #Limits
    d_min, d_max = ax1.get_xlim()
    ax2.set_xlim(d_min, d_max)

    #Convert
    d_norm_values = [1e0, 1e-1, 1e-2, 1e-3]
    phi_values = [d_norm_to_phi(d) for d in d_norm_values]

    #Ticks
    ax2.set_xticks(d_norm_values) #ticks line up with d/R
    ax2.set_xticklabels(f'{phi:.3f} ' for phi in phi_values) # but have value of phi

    #Axis Label
    ax2.set_xlabel('Initial melt fraction, $\phi$', color=c_red, fontsize=wordsize)

    #Style secondary axis
    ax2.xaxis.set_ticks_position('bottom')
```

```
    ax2.xaxis.set_label_position('bottom')
    ax2.tick_params(axis='x', colors=c_red, which='both',
        labelsize=wordsize, width=widthsize)

    #Spines
    ax2.spines['bottom'].set_color(c_red)
    ax2.spines['bottom'].set_linewidth(widthsize/2)

    #Shifting up and down
    ax1.xaxis.set_label_coords(0.5, -0.06) #d/R
    ax2.spines['bottom'].set_position(('outward', 44)) #Melt
        fraction spines
    ax2.xaxis.set_label_coords(0.5, -0.17)  #Melt fraction label


    #####Save######
    plt.savefig(f'plot/scale={scale}/Heat_evol.png', dpi=500)

    with open(f'plot/scale={scale}/Heat_evol.txt', 'w') as f:
        f.write(f'eta_conv = 1e{math.log10(eta_conv)}\n')
        f.write(f'eta_tidal = 1e{math.log10(eta_tidal)}\n')


    plt.show()
```

C.3.3 Chapter 4: Discussion

```
#Plot of observed k2 and modelled k2

#Import modules
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from cmcrameri import cm
import math

#Import parameters
from heat_evolution.param.parameters_rheo import rhoms, rhoml,
    mus
from heat_evolution.param.parameters_io import R, n
from heat_evolution.param.parameters_uni import G, pi

#Functions
```

```python
def k2(d, eta_tidal, scale):
    """Calculate the Love number k2 for a given shell thickness
       and tidal viscosity.

    Parameters
    ----------
    d : float
        Shell thickness [m]
    eta_tidal : float
        Tidal viscosity [Pa.s]
    scale : str
        Scale factor for the shear modulus ('True' or 'False')
    """

    #Calculate gravity
    Rprime = R - d
    gR = 4/3 * pi * G / R**2 * (rhoml*Rprime**3 + rhoms*(R**3 -
       Rprime**3)) #rhoms=rhoml

    #Calculate complex shear modulus
    mu = mus #5e10
    omega = mu / eta_tidal

    if scale == 'True':
        mu_re = d/R*(n**2 * mu / (n**2 + omega**2))
        mu_im = d/R*(n * mu * omega / (n**2 + omega**2))
    elif scale == 'False':
        mu_re = n**2 * mu / (n**2 + omega**2)
        mu_im = n * mu * omega / (n**2 + omega**2)
    else:
        print('Error with d/R')

    muc = mu_re + 1j * mu_im

    #Calculate k2
    k2 = 3/2 / (1 + (5*muc)/(2*rhoms*gR*R))
    k2_re = k2.real
    k2_im = k2.imag

    return k2_re, k2_im

#Plot of model and observed k2
def plot_k2(scale='True', eta_conv=1e21, eta_tidal = np.logspace
   (13,17,3)):
```

```
    """Plot the Love number k2 as a function of tidal viscosity, and compare it to the observed value.

    Parameters
    ----------
    scale : str
        Scale factor for the shear modulus ('True' or 'False'). The default is 'True'.
    eta_conv : float
        Convective viscosity [Pa.s]. The default is 1e21.
    eta_tidal : numpy.ndarray
        Tidal viscosity [Pa.s]. The default is np.logspace(13,17,3).
    """
    ####Formatting######
    plt.figure(figsize=(10, 7))
    sns.set_context('talk') #Options include 'paper', 'notebook', 'talk', and 'poster'
    cmap = cm.roma

    ####Get data####
    for i, eta_tidal_i in enumerate(eta_tidal):
        #Read data
        data = pd.read_csv(f'Crossover_points_scale_{scale}.csv')

        #Mask for convective and tidal viscosity, get stable points
        mask = (data['Convective viscosity'] == eta_conv) & (data['Tidal viscosity'] == eta_tidal_i) & (data['Type'] == 'Stable')
        data = data[mask]

        #Get stable thicknesses
        d = data['Thickness'].values[0] if mask.any() else None

        if d is not None:
            #Calculate k2
            k2_re, k2_im = k2(d, eta_tidal_i, scale)

            #Plot
            plt.scatter(k2_re, -k2_im, c=cmap(i/len(eta_tidal)),
                        marker='D', s=300, alpha=1,
                        label=f'$\eta_{{tidal}}$ = {eta_tidal_i:.0e}')
```

```

    #Plot observed value
    plt.errorbar(0.125, 0.0109, xerr=0.047, yerr=0.0054, fmt='
       none', color = 'black', label = 'Measured')

    #Plot formatting
    plt.legend()
    plt.xlabel(r'$Re(k_2)$')
    plt.ylabel(r'$-Im(k_2)=|k_2|/Q$')
    plt.yscale('log')
    plt.xscale('log')

    #Save
    plt.savefig(f'plot/scale={scale}/k2.png', dpi=500)
    with open(f'plot/scale={scale}/k2.txt', 'w') as f:
        f.write(f'eta_conv = 1e{math.log10(eta_conv)}\n')

        plt.show()
```

C.4 Requirements

```
Bottleneck @ file:///private/var/folders/nz/
    j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_55txi4fy1u/croot/
    bottleneck_1731058642212/work
Brotli @ file:///private/var/folders/k1/30
    mswbxs7r1g6zwn8y4fyt500000gp/T/abs_f7i0oxypt6/croot/brotli-
    split_1736182464088/work
cmcrameri @ file:///home/conda/feedstock_root/build_artifacts/
    cmcrameri_1735373533272/work
contourpy @ file:///private/var/folders/k1/30
    mswbxs7r1g6zwn8y4fyt500000gp/T/abs_2cvjf0v4ux/croot/
    contourpy_1732540055997/work
cycler @ file:///tmp/build/80754af9/cycler_1637851556182/work
fonttools @ file:///private/var/folders/nz/
    j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_ce1jt_55vl/croot/
    fonttools_1737039388732/work
gmpy2 @ file:///private/var/folders/nz/
    j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_51391juwln/croot/
    gmpy2_1738085477864/work
kiwisolver @ file:///private/var/folders/nz/
    j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_cc2l_z_0ri/croot/
    kiwisolver_1737039586949/work
matplotlib==3.10.0
```

```
mpmath @ file:///Users/builder/cbouss/perseverance-python-buildout/croot/mpmath_1728592510324/work
numexpr @ file:///Users/builder/cbouss/perseverance-python-buildout/croot/numexpr_1731706619640/work
numpy @ file:///private/var/folders/k1/30mswbxs7r1g6zwn8y4fyt500000gp/T/abs_2bgq2wg6lu/croot/numpy_and_numpy_base_1738078509889/work/dist/numpy-2.2.2-cp313-cp313-macosx_11_0_arm64.whl#sha256=a7758b7132f833d3337b3ca434305ea8c009b92bf911cd866cef76aa80ebb8a0
packaging @ file:///private/var/folders/nz/j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_a6_qk3qyg7/croot/packaging_1734472142254/work
pandas @ file:///private/var/folders/nz/j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_4aifrweohv/croot/pandas_1732735109535/work/dist/pandas-2.2.3-cp313-cp313-macosx_11_0_arm64.whl#sha256=8f01410ead9f51bf614097bf4230440a4461983f8acb57141e819b93d81e26d6
pillow @ file:///private/var/folders/k1/30mswbxs7r1g6zwn8y4fyt500000gp/T/abs_85xj2lenf1/croot/pillow_1738010251686/work
pyparsing @ file:///Users/builder/cbouss/buildout/croot/pyparsing_1735850351026/work
python-dateutil @ file:///Users/builder/cbouss/perseverance-python-buildout/croot/python-dateutil_1728585579455/work
pytz @ file:///Users/builder/cbouss/perseverance-python-buildout/croot/pytz_1728586101417/work
scipy @ file:///private/var/folders/nz/j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_4c2xem2hut/croot/scipy_1737122914559/work/dist/scipy-1.15.1-cp313-cp313-macosx_11_0_arm64.whl#sha256=2adbb89ee8674c35ca4340464cf1e0513b1d490d3f1061403ef247bb97246b6b
seaborn==0.13.2
setuptools==72.1.0
six @ file:///tmp/build/80754af9/six_1644875935023/work
sympy @ file:///private/var/folders/nz/j6p8yfhx1mv_0grj5xl4650h0000gp/T/abs_9edgwet_ug/croot/sympy_1738108502398/work
tornado @ file:///Users/builder/cbouss/buildout/croot/tornado_1735843024083/work
tzdata @ file:///croot/python-tzdata_1690578112552/work
wheel==0.44.0
```

References

Anderson, J. D., W. L. Sjogren and G. Schubert (May 1996). 'Galileo Gravity Results and the Internal Structure of Io'. In: *Science* 272.5262, pp. 709–712. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.272.5262.709. URL: https://www.science.org/doi/10.1126/science.272.5262.709 (visited on 05/04/2025).

Archer, Allen W. (Feb. 2013). 'World's highest tides: Hypertidal coastal systems in North America, South America and Europe'. In: *Sedimentary Geology* 284-285, pp. 1–25. ISSN: 0037-0738. DOI: 10.1016/j.sedgeo.2012.12.007. URL: https://www.sciencedirect.com/science/article/pii/S0037073812003326 (visited on 03/05/2025).

Auclair-Desrotour, P., C. Le Poncin-Lafitte and S. Mathis (Jan. 2014). 'Impact of the frequency dependence of tidal Q on the evolution of planetary systems'. In: *Astronomy & Astrophysics* 561, p. L7. ISSN: 0004-6361, 1432-0746. DOI: 10.1051/0004-6361/201322782. URL: https://www.aanda.org/articles/aa/abs/2014/01/aa22782-13/aa22782-13.html (visited on 03/05/2025).

Aygün, B. and O. Čadek (2024a). 'Tidal Heating in a Subsurface Magma Ocean on Io Revisited'. In: *Geophysical Research Letters* 51.10, e2023GL107869. ISSN: 1944-8007. DOI: 10.1029/2023GL107869. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/2023GL107869 (visited on 21/01/2025).

Aygün, Burak and Ondřej Čadek (Dec. 2024b). *Love numbers for Io with a magma ocean.* DOI: 10.22541/au.173387131.12714167/v1. URL: https://essopenarchive.org/users/709584/articles/1247828-love-numbers-for-io-with-a-magma-ocean?commit=4c790525b0ecb6e0847a28ed1092e54377792c39 (visited on 24/04/2025).

Bierson, C. J. and F. Nimmo (2016). 'A test for Io's magma ocean: Modeling tidal dissipation with a partially molten mantle'. In: *Journal of Geophysical Research: Planets* 121.11, pp. 2211–2224. ISSN: 2169-9100. DOI: 10.1002/2016JE005005. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/2016JE005005 (visited on 23/03/2025).

Bland, Michael T. et al. (2012). 'Enceladus' extreme heat flux as revealed by its relaxed craters'. In: *Geophysical Research Letters* 39.17. ISSN: 1944-8007. DOI: 10.1029/2012GL052736. (Visited on 07/04/2025).

Borg, Lars E. et al. (Oct. 2019). 'Isotopic evidence for a young lunar magma ocean'. In: *Earth and Planetary Science Letters* 523, p. 115706. ISSN: 0012-821X. DOI: 10.1016/j.epsl.2019.07.008. URL: https://www.sciencedirect.com/science/article/pii/S0012821X19303929 (visited on 07/04/2025).

Boukaré, Charles-Édouard, James Badro and Henri Samuel (Apr. 2025). 'Solidification of Earth's mantle led inevitably to a basal magma ocean'. In: *Nature* 640.8057, pp. 114–119. ISSN: 1476-4687. DOI: 10.1038/s41586-025-08701-z. URL: https://www.nature.com/articles/s41586-025-08701-z (visited on 05/05/2025).

Breuer, Doris, Christopher W. Hamilton and Krishan Khurana (Dec. 2022). 'The Internal Structure of Io'. In: *Elements* 18.6, pp. 385–390. ISSN: 1811-5217, 1811-5209. DOI: 10.2138/gselements.18.6.385. URL: https://pubs.geoscienceworld.org/elements/article/18/6/385/621179/The-Internal-Structure-of-Io (visited on 16/11/2024).

Davies, Ashley Gerard et al. (Oct. 2023). *Io's polar volcanic thermal emission indicative of magma ocean and shallow tidal heating models*. DOI: 10.48550/arXiv.2310.12382. URL: http://arxiv.org/abs/2310.12382 (visited on 05/04/2025).

De Kleer, Katherine, Ryan S. Park and Alfred McEwen (June 2019). *Tidal Heating: Lessons from Io and the Jovian System*. Tech. rep.

Fischer, H.-J. and T. Spohn (Jan. 1990). 'Thermal-orbital histories of viscoelastic models of Io (J1)'. In: *Icarus* 83.1, pp. 39–65. ISSN: 00191035. DOI: 10.1016/0019-1035(90)90005-T. URL: https://linkinghub.elsevier.com/retrieve/pii/001910359090005T (visited on 24/10/2024).

Hay, Hamish and Isamu Matsuyama (Apr. 2019). 'Tides between the TRAPPIST-1 planets'. In: *The Astrophysical Journal* 875.1, p. 22. ISSN: 0004-637X, 1538-4357. DOI: 10.3847/1538-4357/ab0c21. URL: http://arxiv.org/abs/1903.04501 (visited on 03/12/2024).

Henning, Wade G. and Terry Hurford (June 2014). 'TIDAL HEATING IN MULTILAYERED TERRESTRIAL EXOPLANETS'. In: *The Astrophysical Journal* 789.1, p. 30. ISSN: 0004-637X, 1538-4357. DOI: 10.1088/0004-637X/789/1/30. URL: https://iopscience.iop.org/article/10.1088/0004-637X/789/1/30 (visited on 21/04/2025).

Khurana, K. K. et al. (Oct. 1998). 'Induced magnetic fields as evidence for subsurface oceans in Europa and Callisto'. In: *Nature* 395.6704, pp. 777–780. ISSN: 1476-4687. DOI: 10.1038/27394. URL: https://www.nature.com/articles/27394 (visited on 08/04/2025).

Khurana, Krishan K. et al. (June 2011). 'Evidence of a Global Magma Ocean in Io's Interior'. In: *Science* 332.6034, pp. 1186–1189. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1201425. URL: https://www.science.org/doi/10.1126/science.1201425 (visited on 05/04/2025).

Lambeck, Kurt, Paul Johnston and Masao Nakada (1990). 'Holocene glacial rebound and sea-level change in NW Europe'. In: *Geophysical Journal International* 103.2, pp. 451–468. ISSN: 1365-246X. DOI: 10.1111/j.1365-246X.1990.tb01784.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-246X.1990.tb01784.x (visited on 06/04/2025).

Langseth, Marcus G. and Stephen J. Keihm (1977). 'In-Situ Measurementsof Lunar Heat Flow'. In.

Levin, Steven et al. (Dec. 2024). 'Constraints on Europa's Sub-Surface Ice from Juno Microwave Radiometer Observations'. In: AGU. URL: https://agu.confex.com/agu/agu24/meetingapp.cgi/Paper/1558618 (visited on 08/04/2025).

Lopes, Rosaly M C and David A Williams (Feb. 2005). 'Io after *Galileo*'. In: *Reports on Progress in Physics* 68.2, pp. 303–340. ISSN: 0034-4885, 1361-6633. DOI: 10.1088/0034-

4885/68/2/R02. URL: https://iopscience.iop.org/article/10.1088/0034-4885/68/2/R02 (visited on 07/04/2025).

Love, A. E. H. (Feb. 1909). 'The yielding of the earth to disturbing forces'. In: DOI: 10.1098/rspa.1909.0008. URL: https://royalsocietypublishing.org/doi/epdf/10.1098/rspa.1909.0008 (visited on 15/11/2024).

Love, A. E. H. (Augustus Edward Hough) (1911). *Some problems of geodynamics; being an essay to which the Adams prize in the University of Cambridge was adjudged in 1911.* Cambridge, University Press. URL: http://archive.org/details/cu31924060184367 (visited on 03/12/2024).

Matsuyama, Isamu N., Teresa Steinke and Francis Nimmo (Dec. 2022). 'Tidal Heating in Io'. In: *Elements* 18.6, pp. 374–378. ISSN: 1811-5209. DOI: 10.2138/gselements.18.6.374. URL: https://doi.org/10.2138/gselements.18.6.374 (visited on 16/11/2024).

McKinnon, William B. (2007). 'Formation and early evolution of Io'. In: *Io After Galileo.*

— (2023). 'Chapter 3 - Setting the Stage: Formation and Earliest Evolution of Io'. In: *Io: A New View of Jupiter's Moon.* Ed. by Rosaly M C Lopes, Katherine De Kleer and James Tuttle Keane. Vol. 468. Astrophysics and Space Science Library. Cham: Springer International Publishing. ISBN: 978-3-031-25669-1 978-3-031-25670-7. URL: https://link.springer.com/10.1007/978-3-031-25670-7 (visited on 06/04/2025).

Miyazaki, Yoshinori and David J. Stevenson (Nov. 2022). 'A Subsurface Magma Ocean on Io: Exploring the Steady State of Partially Molten Planetary Bodies'. In: *The Planetary Science Journal* 3.11, p. 256. ISSN: 2632-3338. DOI: 10.3847/PSJ/ac9cd1. URL: https://iopscience.iop.org/article/10.3847/PSJ/ac9cd1 (visited on 05/04/2025).

Moore, W. B. (Aug. 2003). 'Tidal heating and convection in Io'. In: *Journal of Geophysical Research: Planets* 108.E8. ISSN: 2156-2202. DOI: 10.1029/2002JE001943. URL: https://onlinelibrary-wiley-com.ezproxy-prd.bodleian.ox.ac.uk/doi/abs/10.1029/2002JE001943 (visited on 06/08/2024).

Mosenfelder, Jed L., Paul D. Asimow and Thomas J. Ahrens (2007). 'Thermodynamic properties of Mg2SiO4 liquid at ultra-high pressures from shock measurements to 200 GPa on forsterite and wadsleyite'. In: *Journal of Geophysical Research: Solid Earth* 112.B6. ISSN: 2156-2202. DOI: 10.1029/2006JB004364. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/2006JB004364 (visited on 16/04/2025).

Murray, Carl D and Stanley F Dermott (2000). 'Tides, Rotation, and Shape'. In: *Solar System Dynamics.* URL: https://www-cambridge-org.ezproxy-prd.bodleian.ox.ac.uk/core/books/solar-system-dynamics/108745217E4A18190CBA340ED5E477A2 (visited on 21/01/2025).

NASA Exoplanet Archive (2025). URL: https://exoplanetarchive.ipac.caltech.edu/docs/counts_detail.html (visited on 21/04/2025).

Nimmo, Francis, Marc Neveu and Carly Howett (Oct. 2023). 'Origin and Evolution of Enceladus's Tidal Dissipation'. In: *Space Science Reviews* 219.7, pp. 1–20. ISSN: 1572-9672. DOI: 10.1007/s11214-023-01007-4. URL: https://link.springer.com/article/10.1007/s11214-023-01007-4 (visited on 05/04/2025).

Norsen, Travis, Mackenzie Dreese and Christopher West (Sept. 2017). 'The gravitational self-interaction of the Earth's tidal bulge'. In: *American Journal of Physics* 85.9, pp. 663–669. ISSN: 0002-9505. DOI: 10.1119/1.4985124. URL: https://doi.org/10.1119/1.4985124 (visited on 01/08/2024).

Papaloizou, J C B, Ewa Szuszkiewicz and Caroline Terquem (June 2018). 'The TRAPPIST-1 system: orbital evolution, tidal dissipation, formation and habitability'. In: *Monthly Notices of the Royal Astronomical Society* 476.4, pp. 5032–5056. ISSN: 0035-8711. DOI: 10.1093/mnras/stx2980. URL: https://doi.org/10.1093/mnras/stx2980 (visited on 21/04/2025).

Park, R. S. et al. (2024). 'The Global Shape, Gravity Field, and Libration of Enceladus'. In: *Journal of Geophysical Research: Planets* 129.1, e2023JE008054. ISSN: 2169-9100. DOI: 10.1029/2023JE008054. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/2023JE008054 (visited on 06/04/2025).

Park, R. S. et al. (Feb. 2025). 'Io's tidal response precludes a shallow magma ocean'. In: *Nature* 638.8049, pp. 69–73. ISSN: 1476-4687. DOI: 10.1038/s41586-024-08442-5. URL: https://www.nature.com/articles/s41586-024-08442-5 (visited on 17/03/2025).

Peale, S. J., P. Cassen and R. T. Reynolds (Mar. 1979). 'Melting of Io by Tidal Dissipation'. In: *Science* 203.4383, pp. 892–894. DOI: 10.1126/science.203.4383.892. URL: https://www.science.org/doi/10.1126/science.203.4383.892 (visited on 09/04/2025).

Pommier, Anne and Alfred McEwen (Dec. 2022). 'Io: A Unique World in our Solar System'. In: *Elements* 18.6, pp. 368–373. ISSN: 1811-5217, 1811-5209. DOI: 10.2138/gselements.18.6.368. URL: https://pubs.geoscienceworld.org/elements/article/18/6/368/621180/Io-A-Unique-World-in-our-Solar-System (visited on 22/01/2025).

Porco, C. C. et al. (Mar. 2006). 'Cassini Observes the Active South Pole of Enceladus'. In: *Science* 311.5766, pp. 1393–1401. DOI: 10.1126/science.1123013. URL: https://www.science.org/doi/10.1126/science.1123013 (visited on 08/04/2025).

Renaud, Joe P. and Wade G. Henning (Apr. 2018). 'Increased Tidal Dissipation Using Advanced Rheological Models: Implications for Io and Tidally Active Exoplanets'. In: *The Astrophysical Journal* 857.2, p. 98. ISSN: 0004-637X, 1538-4357. DOI: 10.3847/1538-4357/aab784. URL: https://iopscience.iop.org/article/10.3847/1538-4357/aab784 (visited on 24/10/2024).

Roberts, James H. and Francis Nimmo (Apr. 2008). 'Tidal heating and the long-term stability of a subsurface ocean on Enceladus'. In: *Icarus* 194.2, pp. 675–689. ISSN: 00191035. DOI: 10.1016/j.icarus.2007.11.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0019103507005593 (visited on 06/04/2025).

Roberts, James H. et al. (Sept. 2023). 'Exploring the Interior of Europa with the Europa Clipper'. In: *Space Science Reviews* 219.6, p. 46. ISSN: 0038-6308, 1572-9672. DOI: 10.1007/s11214-023-00990-y. URL: https://link.springer.com/10.1007/s11214-023-00990-y (visited on 05/04/2025).

Ross, Martin and Gerald Schubert (1986). 'Tidal dissipation in a viscoelastic planet'. In: *Journal of Geophysical Research: Solid Earth* 91.B4, pp. 447–452. ISSN: 2156-2202. DOI: 10.1029/JB091iB04p0D447. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/JB091iB04p0D447 (visited on 18/04/2025).

Schubert, Gerald, Donald L. Turcotte and Peter Olson (Sept. 2001). '13 - Thermal History of the Earth'. In: *Mantle Convection in the Earth and Planets*.

Segatz, M. et al. (Aug. 1988). 'Tidal dissipation, surface heat flow, and figure of viscoelastic models of Io'. In: *Icarus* 75.2, pp. 187–206. ISSN: 00191035. DOI: 10.1016/0019-1035(88)90001-2. URL: https://linkinghub.elsevier.com/retrieve/pii/0019103588900012 (visited on 19/04/2025).

Smith, Bradford A. et al. (June 1979). 'The Jupiter System Through the Eyes of Voyager 1'. In: *Science* 204.4396, pp. 951–972. DOI: 10.1126/science.204.4396.951. URL: https://www.science.org/doi/10.1126/science.204.4396.951 (visited on 07/04/2025).

Turcotte, Donald and Gerald Schubert (Apr. 2014). *Geodynamics*. URL: https://www.cambridge.org/highereducation/books/geodynamics/E0E847DA9FE68BDB90C2E457791F0C98 (visited on 01/02/2025).

Tyler, Robert H., Wade G. Henning and Christopher W. Hamilton (June 2015). 'TIDAL HEATING IN A MAGMA OCEAN WITHIN JUPITER'S MOON Io'. In: *The Astrophysical Journal Supplement Series* 218.2, p. 22. ISSN: 1538-4365. DOI: 10.1088/0067-0049/218/2/22. URL: https://iopscience.iop.org/article/10.1088/0067-0049/218/2/22 (visited on 24/10/2024).

Vogel, Tracy (June 2023). *Tides - NASA Science*. URL: https://science.nasa.gov/moon/tides/ (visited on 14/04/2025).

Voosen, Paul (Dec. 2024). *Surprisingly thick ice on Jupiter's moon Europa complicates hunt for life*. URL: https://www.science.org/content/article/surprisingly-thick-ice-jupiter-s-moon-europa-complicates-hunt-life (visited on 08/04/2025).

Vuik, Kees et al. (Feb. 2023). *Numerical Methods for Ordinary Differential Equations*. TU Delft OPEN Books. ISBN: 978-94-6366-665-7. URL: https://books.open.tudelft.nl/home/catalog/book/165 (visited on 21/04/2025).

Wakita, Shigeru et al. (Mar. 2024). 'Multiring basin formation constrains Europa's ice shell thickness'. In: *Science Advances* 10.12, eadj8455. DOI: 10.1126/sciadv.adj8455. URL: https://www.science.org/doi/10.1126/sciadv.adj8455 (visited on 08/04/2025).

Walterová, Michaela et al. (July 2023). *Andrade rheology in planetary science*. DOI: 10.22541/essoar.169008299.99203183/v1. URL: https://essopenarchive.org/users/563776/articles/653374-andrade-rheology-in-planetary-science?commit=3461d271a9982663df08e228f95cfce447534220 (visited on 05/05/2025).

Williams, Dave (Apr. 2016). *Solar System Small Worlds Fact Sheet*. URL: https://nssdc.gsfc.nasa.gov/planetary/factsheet/galileanfact_table.html (visited on 17/04/2025).

Wolszczan, A. and D. A. Frail (Jan. 1992). 'A planetary system around the millisecond pulsar PSR1257 + 12'. In: *Nature* 355.6356, pp. 145–147. ISSN: 1476-4687. DOI: 10.1038/355145a0. URL: https://www.nature.com/articles/355145a0 (visited on 21/04/2025).