Master Thesis

The Discontinuous Galerkin Finite Element Method and the Julia programming language for geophysical modelling

Jort Jansen
Supervisor: dr. Cedric Thieulot
Second supervisor: Lukas van de Wiel MSc
Utrecht University April 29, 2022

Abstract

Geodynamic modelling is used to enhance how we understand the physical processes which drive micro- and macro scale processes over geological time scales. With continuous development of the methods and computational techniques used, numerical models are always improving in resolution and accuracy. A technique emerging in this field is the discontinuous Galerkin Finite Element Method, this method has shown promise in other fields. So, implementing this for the heat transport equation can be the stepping stone into making this method more regularly used by Geosciences students. Another area where a lot of progress is made concerning speeding up calculations is a relatively new programming language: Julia. This thesis will compare Julia with the more widely used programming language Python. If Julia is as fast and easy to use as many believe it to be it can be a very promising programming language for future Geosciences students.

Universiteit Utrecht

Contents

1 Introduction **3**

1.1 Geodynamic modelling . . . 3

1.2 Discontinuous Galerkin . . . 3

1.3 Governing equations . . . 4

1.4 Finite elements . . . 4

1.5 Julia programming language . . . 5

1.6 Geophysical inversion . . . 5

1.7 Challenges and goals . . . 6

2 Methods for discontinuous galerkin **6**

2.1 Discontinuous Galerkin . . . 6

2.2 The Physics . . . 7

2.3 Solving . . . 8

2.3.1 Global matrix . . . 8

2.3.2 Successive substitution method . . . 8

2.3.3 q eliminated . . . 8

2.3.4 Convergence . . . 8

2.4 Time dependence . . . 9

2.5 Multidimensions . . . 9

3 Results **10**

3.1 1D steady state diffusion . . . 10

3.1.1 Derivation . . . 10

3.1.2 Stabilization parameters . . . 10

3.1.3 Error calculation . . . 10

3.1.4 Solve method . . . 12

3.2 1D transient diffusion . . . 12

3.3 1D Pure advection . . . 13

3.3.1 Experiment 1 . . . 13

3.3.2 Experiment 2 . . . 13

3.4 1D Advection-diffusion . . . 15

3.5 2D steady state diffusion . . . 15

3.6 2D pure advection . . . 16

3.7 2D Advection-Diffusion . . . 17

3.7.1 Experiment 1 . . . 17

3.7.2 Experiment 2 . . . 18

4 Methods for the geophysical inversion **20**

4.1 Geophysical inversion . . . 20

4.1.1 Forward model: gravity . . . 20

4.1.2 Forward model: Stokes . . . 21

4.1.3 Inversion . . . 21

4.2 Julia programming language . . . 22

5 Results for geophysical inversion **22**
5.1 Inversion parameters . . . 22
5.2 Julia vs Python . . . 24

6 Discussion **27**
6.1 Discontinuous Galerkin Method . . . 27
6.1.1 Numerical flux . . . 27
6.1.2 Boundary conditions . . . 27
6.1.3 The DG FEM or the CG FEM . . . 27
6.1.4 Thoughts about the DG-FEM . . . 28
6.2 Inversion . . . 28
6.3 Julia vs Python . . . 28

7 Conclusion **29**

References **30**

A 1D Steady State **32**

B 1D transient **36**

C 1D Convection **38**

D 2D Steady State diffusion **40**

E 2D pure advection **44**

F 2D advection diffusion **47**

G Julia tutorial **52**
G.1 Introduction . . . 52
G.2 executing . . . 52
G.3 Packages . . . 52
G.4 Data types . . . 52
G.5 Scalar Types . . . 52
G.6 Arrays . . . 53
G.7 Control Flow . . . 53
G.8 Input Output . . . 53
G.9 functions . . . 53
G.10 Comparison between Python and Julia . . . 54
G.11 Conclusion . . . 54

1 Introduction

1.1 Geodynamic modelling

Geodynamic modelling can be used to gain a better understanding of the physical processes which drive the tectonic evolution of the earth, these processes can range between grain scale evolution (Ricard and Bercovici, 2009) to large scale mantle convection (Schubert et al., 2001). Numerical modelling of these geodynmamic processes can be done by solving the relevant partial differential equations (PDE). Such as, the PDEs for conservation of mass, momentum and energy (Reddy (2010) ;Schubert et al. (2001)).

In recent years these numeric models are increasing in complexity, due to increased computing capabilities. Geodynamic models started with two-dimensional models which evolved into more complex models. Nowadays these models are able to deal with complex 3D geometries, at high resolution and small time steps. Models such as Stag3D (Tackley and Xie, 2003) and ASPECT (Kronbichler et al., 2012) are getting more advanced due to an increase in techniques and available computing power.

A numerical technique often used in computational modelling is the Finite Element Method (FEM), this method forms the backbone of many geodynamic codes. These codes are used to tackle questions involving the evolution of the Earth, such as the origin of the Large Low Shear Velocity Provinces (LLSVPs)(Mulyukova et al., 2015), the processes behind subduction dynamics and whether there is whole- or layered mantle convection (Schubert, 1992). These processes are characterized by temperature differences and sharp boundaries between chemically distinct regions. The most used FEM method is the continuous Galerkin Finite Element Method (CG-FEM), this can be an appropriate method for solving geodynamical problems because it gives an easy representation of the total solution and it is capable of dealing with complicated geometries (Reddy, 1993).

Unfortunately, the time evolution of the thermo-chemical system is challenging because the boundaries can be non-diffusive and nearly discontinuous. By construction, the CG-FEM is not capable of dealing with these boundaries since this method requires the values on the nodes to be continuous. When dealing with discontinuities, the streamline upwind Petrov-Galerkin (SUPG) method can be used, although this method still yields under- and overshoots (Brooks and Hughes, 1982). An alternative for these methods is the discontinuous Galerkin Finite Element Methods (DG-FEM). This method has been demonstrated to be very effective for solving seismic wave propagation in seismic media (Wilcox et al. (2010)).

1.2 Discontinuous Galerkin

The DG-FEM differs from the CG-FEM because it allows discontinuities between elements since it uses discontinuous discrete functions (Puckett et al., 2018). This results in the fact that the DG-FEM requires more degrees of freedom (DoF). This method can be seen as a combination of the CG-FEM and the Finite Volume Method (FVM), which are combined in such a way that it entails the advantages of both methods. It has been shown by Cockburn et al. (2012), that it can be successfully used in convection-dominated applications, while it maintains its geometric flexibility and higher local approximations through the use of higher-order elements.

The DG-FEM was introduced in 1973 by Reed and Hill and in recent years it has been gaining a lot of traction in various fields, because it has shown to be a numerical technique which can be successfully applied for relatively low computational costs. However, there have been limited studies in geodynamic applications for these methods with the notable exception of He et al. (2017). In this master thesis, the DG-FEM will be used to solve the heat equation, in the case of the 1-and 2-dimensional diffusion equation and the advection equation in 1- and 2-dimensions. For simplicity we start with the 1D steady state conduction equation. To achieve this special attention has to be given to the numerical flux between elements, how the outer boundaries are defined and what kind of time discretization has to be implemented (Bathe, 2001).

1.3 Governing equations

The advection-diffusion equation is an important equation to study since it has applications in fluid dynamics and thus in geodynamics, for example in rising plumes within the earth mantle or it can be used to determine the cooling of a subducting slab. This heat transfer equation consist of a advection and a conduction component. The conduction heat transfer component is defined by a diffusion process, this heat flow is proportional to the temperature gradient (Incropera et al., 1996). The convection heat transfer is defined by the motion of a fluid or on geological time scale the Earth's mantle (Reddy and Gartling, 2010). To compute the temperature field within a system the heat transport or energy equation has to be solved.

$$\rho C_p \left(\frac{\partial T}{\partial t} + u \cdot \nabla T \right) = \nabla \cdot (k \nabla T) + H \tag{1}$$

Here T is the temperature, k the thermal conductivity, C_p the heat capacity, u the velocity and H the heat production.

1.4 Finite elements

In order to solve the advection-diffusion equation this thesis uses the Finite Element Method (FEM). The FEM is a very useful method for solving Partial differential equations (Reddy and Gartling, 2010). The solutions of the FEM are a set of unknowns and functions, these functions are an approximation which follow boundary conditions. Constructing a FEM calculation consists of the discretization of the PDE, rewriting this PDE to the weak form, setup of the equations, assembly of the elements or use element-by-element calculations, imposing the boundary conditions and the solution of the equations.

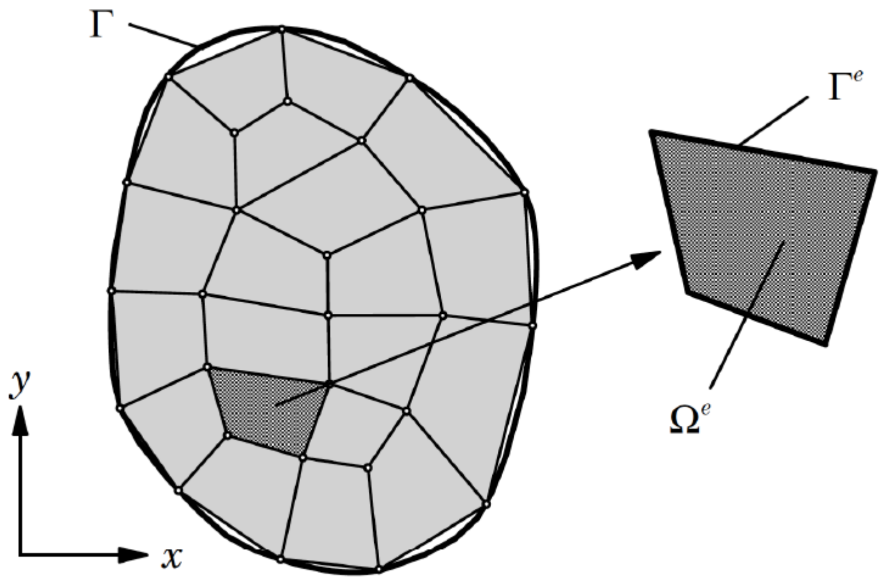


Figure 1: Two dimensional grid with highlighted element (Reddy and Gartling, 2010).

First the region of interest is subdivided into a set of subdomains, these are the so called finite elements (Figure 1). These elements can have each geometric shape for which the approximation functions can be uniquely defined, so they can be triangles or squares for example. For each element the symbol used to resemble each elements domain is Ω^e which is bounded by Γ^e as is shown in Figure 1. When dealing with a dependent unknown, such as temperature, within the domain Ω this unknown temperature can be approximated over a finite element Ω^e by the expression:

$$T(x,y) \approx T^e(x,y) = \sum_{j=1}^{n} T_j^e N_j^e(x,y) \tag{2}$$

T^e is an approximation of the temperature T over a single element Ω, T_j^e indicates the values of the function at the nodes of the element and N_j^e are the functions related to the element.

Then the continuous PDE of interest has to be converted from the strong from to the weak form. The strong form is valid everywhere, whereas the weak form is only true on the nodes. Interpolation between the nodes can be done by the use of shape functions. This study uses linear shape functions between the nodes which gives the weak form of the PDE over one element. These shape functions are used to describe the geometry of each element. This leaves the element-based equations, which together with how the elements and nodes are linked, or the connectivity, can be used to construct a linear set of equations in the form of $A\vec{x} = \vec{b}$ where A and $\vec{b}$ are the known and the vector $\vec{x}$ consists of unknowns.

1.5 Julia programming language

Alongside constructing a discontinuous Galerkin FEM this thesis will consist of making a comparison between the Julia and Python programming languages. This will be done by converting several Fieldstones originally written by Cedric Thieulot in the python programming language to Julia (Thieulot, 2019). Fieldstone is a teaching tool for students to get more acquainted with particular topics in computational geodynamics. Julia is a programming language that combines the ease-of-use of high-level languages such as MATLAB and Python but with performance levels similar to FORTRAN and C (Nagar et al., 2017).

Julia has been developed for efficient, fast numerical computing and to bridge the gap between performance and productivity (Bezanson et al., 2018). Julia offers help with the trade-off between fast executing and fast coding, this is done with the help of the just-in-time (JIT) compilers (Bezanson et al., 2017). A JIT compiler is part of the run-time interpreter, which is a way of compiling that occurs during the executing of the program. Whereas a normal compiler converts the source code in its entirety to machine code. Julia is a relatively new programming language so it has a less extensive library than for example Python, but Julia libraries are written in the Julia language, which makes it less necessary to learn other programming languages. Julia is emerging in the geophysical community, for instance in the development of a framework for seismic inversion (Witte et al., 2019) but is still relatively small (Kaus et al., 2022).

1.6 Geophysical inversion

To get more acquainted with Julia, it is used to solve a geophysical inversion problem. Geophysical inversions can be used to recover information about how a physical property is distributed in the subsurface. This is done by inverting forward models and minimizing the difference between real-earth data and computed data. Inversions have been applied to multiple different observations such as, time tomography (Bijwaard et al., 1998), post-glacial rebounds(Hager and O'Connell, 1979) and to constrain the rheology of the upper mantle (Baumann et al., 2014). A problem with inverse problems is the fact that the solution can be non-unique. Fortunately, there are multiple methods to overcome this non-uniqueness, by including a priori constrains (Jackson (1979) , Cercato (2009)), with the help of a Tikhonov regularization (Tikhonov, 1963) or by means of a joint inversion (Vozoff and Jupp (1975), Baumann et al. (2014)). A joint inversion can be used to recover by means of the inversion process which consists of multiple datasets. In this thesis, the inversion of gravity and velocity will be joint, which can help to constrain the rheology of the upper mantle, similar as has been done by Baumann et al. (2014).

1.7 Challenges and goals

Because there are chemically distinct regions within the earth's crust and mantle, it is important to model the thermo-chemical evolution of how these systems interact (Hoffman and McKenzie, 1985). However, these fields are non-diffuse so we require methods that can deal with sharp boundaries (He et al., 2017). This thesis will attempt to solve this problem by using the discontinuous Galerkin method.

Besides the DG-FEM increasing the resolution can also be used to gain better constrains on geodynamic models. This will be done by rewriting a geophysical inversion code from python to Julia. This programming language has shown to be an easy-to-use and fast programming language. To compare these two programming languages the inversion code is put to the test.

These codes will be added to the Fieldstone project by Cedric Thieulot and will provide a starting point for implementing the DG-FEM to solve flow equations and allowing Earth Science students to explore geodynamics using Julia.

2 Methods for discontinuous galerkin

2.1 Discontinuous Galerkin

The discontinuous Galerkin method hasn't been used extensively in the field of geodynamics, this method differentiates itself from other Galerkin methods since a node is split up in such a way that it is possible for a node to have multiple values. For a 1D case this is shown in Figure 2. When the partial differential equation of interest is a second order PDE, the PDE is split up in two first-order PDEs. After this separation a new variable is introduced, the numerical flux $\hat{q}$. The weak form is found by taking the integral between the nodes and by multiplying the PDE with test functions. These test functions are then taken to be the shape functions that describe the linear element. This results in a set of equations of the form: $A \cdot \vec{x} = \vec{B}$, where $\vec{x}$ describes the set of unknowns.

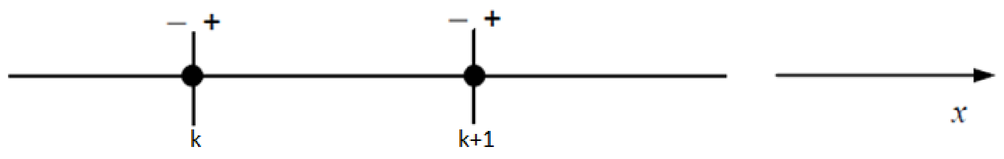


Figure 2: One dimensional illustration of the jump across an element where k and k+1 denote the boundaries of that element.

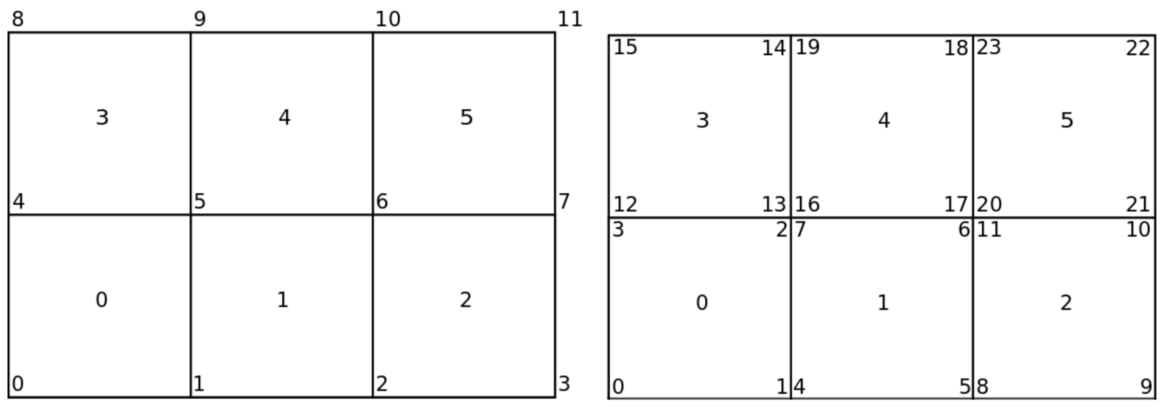


Figure 3: a. Two dimensional connectivity for the continuous galerkin method. b. Two dimensional connectivity for the discontinuous galerkin method

Before discretising the PDEs and then solving them, the domain has to be divided into polygons, e.g. triangles or quadrilaterals. These polygons together make up the grid for the area of interest. In Figure 3 it is shown how these elements and their local numbering are related to each other for a CG and DG grid. Here we see that the DG method uses more nodes than the CG method.

2.2 The Physics

To compute the temperature field within a system. The heat transport or energy equation has to be solved.

$$\rho C_p \left(\frac{\partial T}{\partial t} + u \cdot \nabla T \right) = \nabla \cdot (k \nabla T) + H \tag{3}$$

Here T is the temperature, k the thermal conductivity, C_p the heat capacity, u the velocity and H the heat production. If we regard the 1-D diffusion steady state case there is no time dependence so, $\frac{\partial T}{\partial t} = 0$, the velocity=0 and the gradient is only the derivative in one direction. So only the right hand side terms are preserved:

$$\frac{\partial T}{\partial x} \cdot (k \frac{\partial T}{\partial x}) + H = 0 \tag{4}$$

When we regard a transient 1D case we have to account for the time dependence, equation 3 becomes:

$$\rho C_p \frac{\partial T}{\partial t} - \frac{\partial T}{\partial x} \cdot (k \frac{\partial T}{\partial x}) = H \tag{5}$$

For the pure advection problem there is no diffusion term so equation 1 becomes:

$$\rho C_p (\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x}) = H \tag{6}$$

For the advection-diffusion combined case in 1D the equation is:

$$\rho C_p (\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x}) = \frac{\partial T}{\partial x} \cdot (k \frac{\partial T}{\partial x}) + H \tag{7}$$

Equations 4, 5, 6 and 7 are all regarded in 1 dimension, respectively in 2D these equations would be:

$$\vec{\nabla} \cdot k \vec{\nabla} T + H = 0 \tag{8}$$

$$\rho C_p \frac{\partial T}{\partial t} - \vec{\nabla} T \cdot (k \vec{\nabla} T) = H \tag{9}$$

$$\rho C_p (\frac{\partial T}{\partial t} + \vec{u} \vec{\nabla} T) = H \tag{10}$$

With the DG method the heat flow between two elements has become an unknown, so the heat flux $\vec{q}$ is introduced. This is coupled to the heat transfer equation since the diffusion term is: $\vec{q} = -k\vec{\nabla}T$. So, $\vec{q}$ is another unknown which can be added to the system of linear equations.

2.3 Solving

When solving a PDE that has been discretized we get a system of linear equations in the form of:

$$\mathbf{A}\vec{x} = \vec{b} \tag{11}$$

Here $\mathbf{A}$ is the known linear operator of size n $\times$ n, $\vec{b}$ is the solution vector of size n which is also known and $\vec{x}$ is the vector of unknowns. There are several methods for solving this problem. Such as, the Successive substitution method, forming a global matrix which can be inverted and by eliminating for the variable $\vec{q}$.

2.3.1 Global matrix

The global matrix can be formed by putting all the elemental equations together. This large matrix then has to be inverted in order to determine the values at all the unknowns. In the next iteration this updated data is used and similar to the successive substitution method this will go on until convergence has been reached. Here the system will become $\vec{x} = \mathbf{A}^{-1}\vec{b}$ however $\mathbf{A}$ will be a very large system in 1D approximately twice as large as for the CG-FEM and in 2D four times as large. An inverse of such a large matrix costs a lot of memory and computational power.

2.3.2 Successive substitution method

The Successive substitution method start with all variables initialized to be zero, except for the boundary conditions. Then the problem for the first element is solved, the data that comes out of this first calculation is used to calculate the next element. The calculation sweeps through all the elements, when it reaches the last element it starts once again at the first element. This will continue until convergence is achieved.

2.3.3 q eliminated

Another way of solving this problem is to rewrite the set of equations to eliminate the variable q and solve for temperature alone. The element calculations are similar to the successive substitution method. This can be applied since $\vec{q}$ is an intermediate variable. This method will decrease the calculations required however it will increase the bandwidth of the elemantal matrix which will make it more complex (Li, 2006)

2.3.4 Convergence

All the described methods can solve the system iteratively, which has been used in this study. This solving will go on until convergence has been reached. It is tested whether convergence has been reached by taking the L^2-norm, when this is below a certain threshold ϵ the iterations end and it is regarded that steady state is reached.

The L^2-norms for the error of T and q are defined as:

$$\left(\frac{\sum_{k=1}^{N}(T_{k,i+1}^{-} - T_{k,i}^{-})^2 + (T_{k,i+1}^{+} - T_{k,i}^{+})^2}{\sum_{k=1}^{N} {T_{k,i+1}^{-}}^2 + {T_{k,i+1}^{+}}^2}\right)^{1/2} \leq \epsilon$$

$$\left(\frac{\sum_{k=1}^{N}(q_{k,i+1}^{-} - q_{k,i}^{-})^2 + (q_{k,i+1}^{+} - q_{k,i}^{+})^2}{\sum_{k=1}^{N} {q_{k,i+1}^{-}}^2 + {q_{k,i+1}^{+}}^2}\right)^{1/2} \leq \epsilon$$

2.4 Time dependence

To account for the time dependence of the transient heat conduction problem and the pure advection problem a time integrator for the numerical solution is required. In this study the Runge-Kutta Time integration is used, more specifically the first order RK method (Euler forward method):

$$T_k^{l+1} = T_k^l + \Delta t \dot{T}_k^l$$

Where l denotes the l^{th} timestep. The timestep, Δt, has to be below a certain critical value. The chosen time step dt used for time integration is chosen to conform to the Courant-Friedrichs-Lewy (CFL) condition Anderson and Wendt (1995). When the timestep is larger than the CFL criterion, the simulation will produce numerical instabilities. The CFL condition is defined as:

$$dt = C \min\left(\frac{h}{u}, \frac{h^2}{\kappa}\right) \tag{12}$$

Where h is the element size, u the velocity, $\kappa = k/\rho C_p$ and C is the Courant Number chosen to be 0.1.

2.5 Multidimensions

When expanding the dG-FEM to multiple dimensions the number of DoFs increase quadratically, each node is split up in such a way that it entails 4 separate values for the temperature and the flux in x- and y-direction. This is also depicted in Figure 4. The steps to derive the weak form in 2D are similar to those made to derive the 1D weak form. One important decision is to chose appropriate numerical fluxes, these are the approximations of the heat flux (q) and the temperature on the boundary of the element. Several studies have studied these numerical fluxes for the DG. In this study the Local Discontinuous Galerkin method (LDG) introduced by Cockburn and Shu (1998) is used. When expanding from 1D to 2D we also have to regard the mesh generation. Here we tessellate the domain with polygons, such as, right-angled triangles and squares.

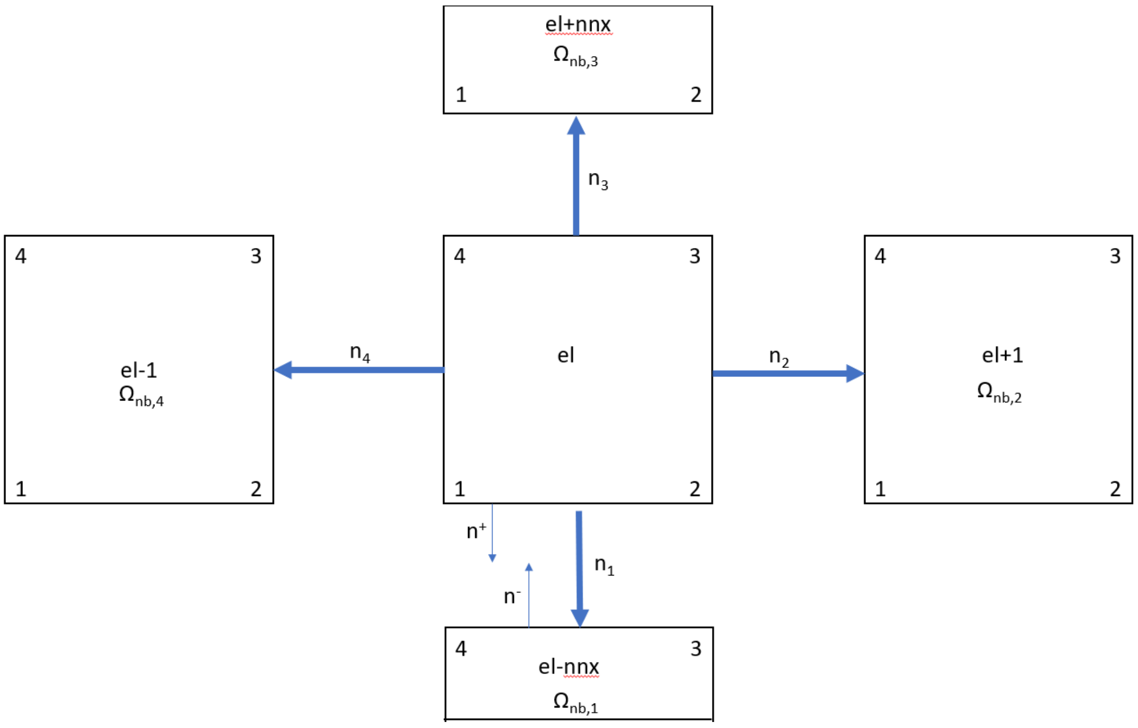


Figure 4: Geometric relation between element el and its neighbours. The elements are separated to illustrate inter element relations

3 Results

3.1 1D steady state diffusion

3.1.1 Derivation

For a 1-D steady state case we start with the problem:

$$\frac{d^2T}{dx^2} = 0 \qquad T(x=0) = 0 \qquad T(x=1) = 1 \qquad \text{on} \quad x \in [0,1] \tag{13}$$

A detailed derivation of the discontinuous Galerkin for a one-dimensional steady state case can be found in Appendix A. For this steady state case there is no time dependency, so it doesn't show the change over time it only depicts how the system in steady state will be. This problem is used to test which solver converges fastest, and which then can be used to tackle more difficult problems. The analytical solution to this relatively easy problem is $T(x) = x$.

3.1.2 Stabilization parameters

In order to determine which stabilization parameters best fit the DG method, the region of interest is split into 3 equally spaced elements. To determine what the best choice of the stabilization constants $\mathcal{E}$ and $\mathcal{C}$ is, several combinations have been tried. The results of this test can be found in Table 1:

			Node 1	Node 2	Node 3	Node 4
Analytical		T	0	$\frac{1}{3}$	$\frac{2}{3}$	1
		q	1	1	1	1
$\mathcal{E} = 1$	$\mathcal{C} = -\frac{1}{2}$	T	0.	0.33333353	0.66666696	1.
		q	0.99999752	1.00000017	1.00000014	0.99999912
	$\mathcal{C} = 0$	T	-2.28242410e-05	3.33380071e-01	6.66703614e-01	1.00000537e+00
		q	0.99987347	0.99999302	0.99991302	0.99995804
	$\mathcal{C} = \frac{1}{2}$	T	2.77555710e-18	3.33333320e-01	6.66667212e-01	1.00000000e+00
		q	0.99999752	1.00000082	1.00000002	1.00000085
$\mathcal{E} = 4$	$\mathcal{C} = -\frac{1}{2}$	T	0.	0.33333331	0.66666665	1.
		q	0.99999987	0.99999995	1.	1.00000004
	$\mathcal{C} = 0$	T	-5.17980307e-10	3.33333333e-01	6.66666667e-01	1.00000000e+00
		q	0.99999998	1.	1.	1.
	$\mathcal{C} = \frac{1}{2}$	T	2.13504162e-18	3.33333307e-01	6.66666650e-01	1.00000000e+00
		q	0.99999987	0.99999997	1.00000004	1.00000009
$\mathcal{E} = 10$	$\mathcal{C} = -\frac{1}{2}$	T	0.	0.33220434	0.66569776	1.
		q	0.9954013	0.99781218	1.00099547	1.00290672
	$\mathcal{C} = 0$	T	-1.65153371e-05	3.32755255e-01	6.66179248e-01	9.99988319e-01
		q	0.99766986	0.99917319	1.00047587	1.00164804
	$\mathcal{C} = \frac{1}{2}$	T	1.01345619e-17	3.32033029e-01	6.65550742e-01	1.00000000e+00
		q	0.9954013	0.99832611	1.00195046	1.00394662

Table 1: Successive substitution

From this table it becomes evident that the best combination of the stabilization parameters is $C = 0$ and $\mathcal{E} = 4$.

3.1.3 Error calculation

It is also important to determine how fast convergence is reached for these parameters, this can be done by calculating the convergence tolerance and how many iterations are required to get to that convergence. We start by examining a range of 32 elements and how the parameters have an effect on the number of iterations required. The problem is the same as in section 3.1.2, with a convergence tolerance of 10^{-8}.

$\mathcal{E}$	$\mathcal{C}$	# iterations
1	0	772
2	0	414
3	-1/2	**104**
3	0	284
3	1/2	129
4	-1/2	196
4	0	220
4	+1/2	196
5	-1/2	256
5	0	195
5	+1/2	256
6	0	246
10	0	486

Table 2: required number of iterations for various stabilization parameter configurations

From Table 2 it becomes evident that the least amount of iterations are required when $\mathcal{E}$ and $\mathcal{C}$ are 3 and -0.5 respectively. So, these are the values we use for the error calculation and for determining how the temperature and heat flux evolve for each iteration.

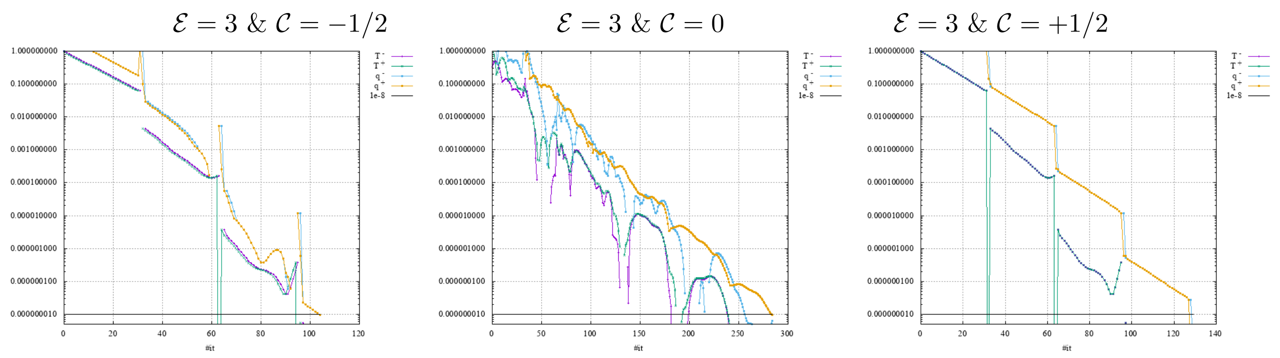


Figure 5: Convergence for the 3 cases

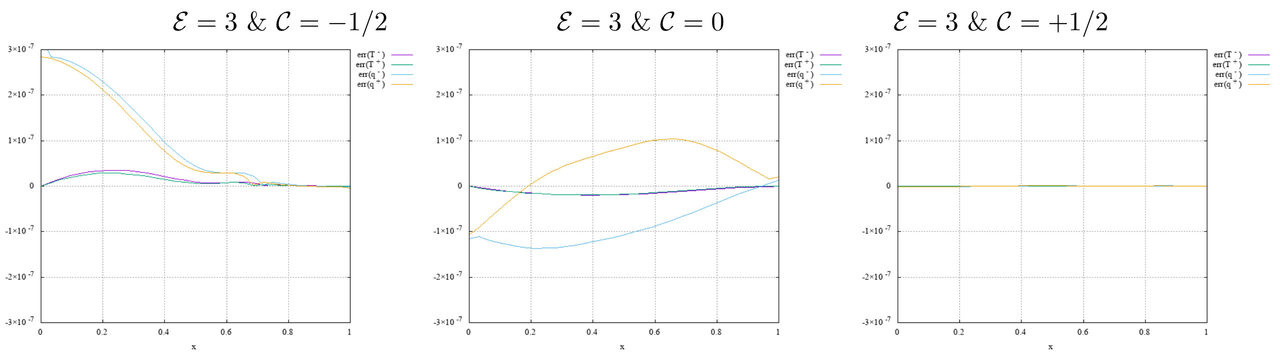


Figure 6: Error between computed and analytical solution as a function of x

From Figures 5, 6 and 7 we can see that the error depends on several factors. The convergence for the case where $\mathcal{C} = 0$ converges quite erratically. From the outcomes of these parameters the $\mathcal{E} = 3, \mathcal{C} = -1/2$ reaches convergence the fastest however the $\mathcal{E} = 3, \mathcal{C} = 1/2$ has the lowest error. For the sake of speed $\mathcal{E} = 3, \mathcal{C} = -1/2$ has been chosen. These parameters have been used to determine the number of iterations required to determine the effect of resolution. Table 3 shows that the relation between the resolution and the number of iterations is linear.

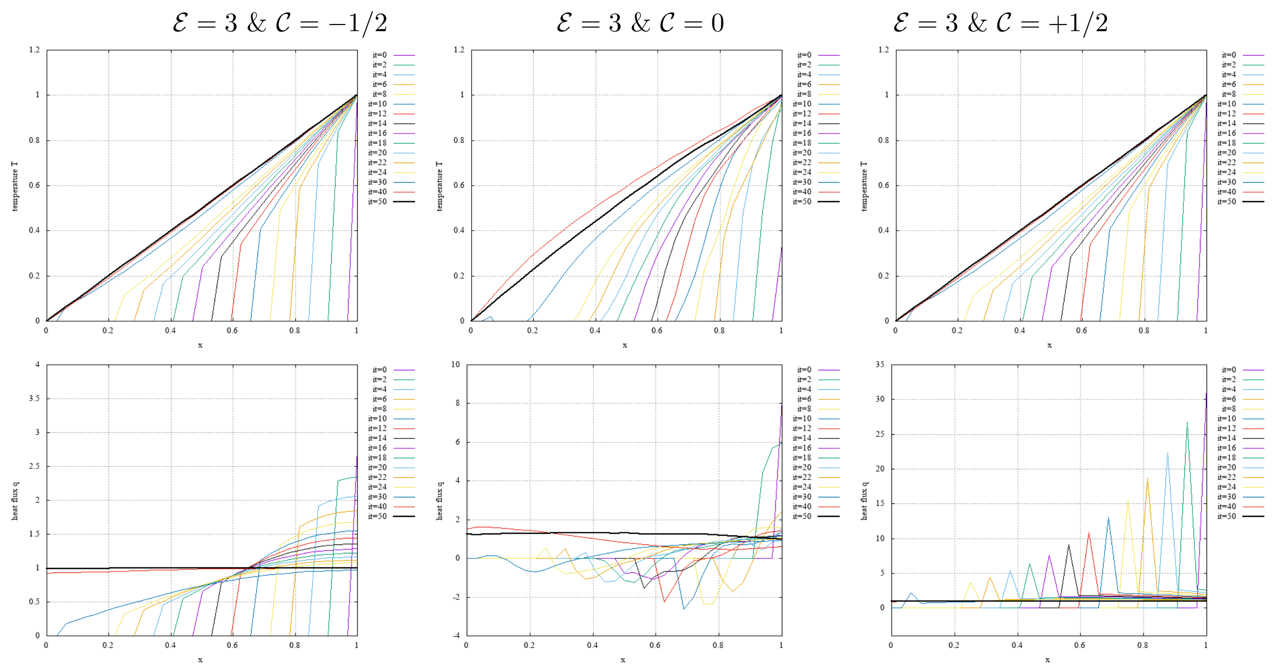


Figure 7: The path to convergence for the temperature and heat flux

nelx	# iterations
08	60
16	116
32	220
64	436
128	857

Table 3: Number of iterations required for several resolutions

3.1.4 Solve method

The stabilization parameters determined in the previous section have been used to compare the Successive substitution method with the outcome of the global matrix and for the calculations where q is first eliminated. The comparison between these methods can be found in Table 4.

With the same tolerance, the successive substitution method requires fewer iterations then the global matrix method. So this method is used in all further codes. Another method of solving the 1D steady state is with eliminating the variable q and only solving for the temperature. According to Cockburn and Shu (1998), this method requires the stabilization parameter $\mathcal{E}$ to meet some requirements to ensure stability.

3.2 1D transient diffusion

For a time dependent case we look at a similar problem as the steady state

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \qquad T(x=0)=0 \qquad T(x=1)=1 \qquad \text{on} \quad x \in [0,1] \tag{14}$$

Besides the steady state heat conduction this method can also be applied to transient heat conduction problems (Appendix B). The problem is similar to the problem for the 1D steady state case so, the same steps are taken as with the steady state case where q is eliminated. Now it is possible to solve for the temperature alone, to include the time dependence the problem has to be integrated over time. This is done by using the first order Runge-Kutta time integration. Which can then be filled into the calculation for the temperature. To ensure that there is no numerical instability the time step has to be below the CFL criterion.

	Iteration		Node 1	Node 2	Node 3	Node 4
Analytical		T	0	$\frac{1}{3}$	$\frac{2}{3}$	1
		q	1	1	1	1
Succesive	1	T	0.	0.	0.13780919	0.95229682
		q	0.	0.	1.48409894	3.1024735
	10	T	-2.50359839e-06	3.33219494e-01	6.66589019e-01	9.99991403e-01
		q	0.99984178	0.99990088	1.00003426	1.00028923
	20	T	-5.17980307e-10	3.33333333e-01	6.66666667e-01	1.00000e+00
		q	0.99999998	1.	1.	1.
Assembled	1	T	0.	0.	0.13780919	0.95229682
		q	0.	0.	1.48409894	3.1024735
	10	T	2.16328639e-04	3.39310626e-01	6.72874806e-01	1.0003942e+00
		q	1.02062084	1.00517102	0.99311122	0.97669299
	20	T	-8.37239436e-06	3.33288420e-01	6.66622580e-01	9.99991403e-01
		q	0.99979916	0.9999304	1.00007229	1.00019856
q eliminated	1	T	0.	0.	0.1875	1.
		q	0.	0.	1.5	3.
	10	T	0.	0.32963851	0.66297184	1.
		q	0.98635758	0.99317879	1.00682121	1.01364242
	20	T	0.	0.33329973	0.66663306	1.
		q	0.99987592	0.99993796	1.00006204	1.00012408

Table 4: Different solve methods and their precision for several iterations.

3.3 1D Pure advection

The derivation of the 1D pure advection is similar to that of the transient diffusion problem and can be found in Appendix C again the first order Runge-Kutta time discretisation is used. We use the advection part from the heat transport equation:

$$\frac{\partial T}{\partial t} + \vec{u}\frac{\partial T}{\partial x} = 0 \tag{15}$$

3.3.1 Experiment 1

For this test we define the system as:

$$T(x,0) = \begin{cases} \sin(10\pi x) & \text{for } x < 0.1 \\ 0 & \text{for } x \geq 0.1 \end{cases} \qquad \text{on} \quad x \in [0,1] \tag{16}$$

The velocity is $u = 0.1$, it consists of 100 elements, so an element size of 1/100. The timestep is $\delta t = 10^{-4}$, the model takes 8000 steps so the time runs to $t = 8$. The CFL-number can be calculated by:

$$C = \frac{\partial t \cdot u}{h} = 0.001 \tag{17}$$

The result of this experiment can be found in 8, here we see the starting position of the sine wave, the analytical solution and the numerical solution. There is a small difference between the two solutions, most notably the little indents before and after the wave.

3.3.2 Experiment 2

This experiment is based on the advection benchmark which originates from Donea and Huerta (2003), the temperature profile at the start is a step function and defined as:

$$T(x,0) = \begin{cases} 1 & \text{for } x < 0.25 \\ 0 & \text{for } x \geq 0.25 \end{cases} \qquad \text{on} \quad x \in [0,1] \tag{18}$$

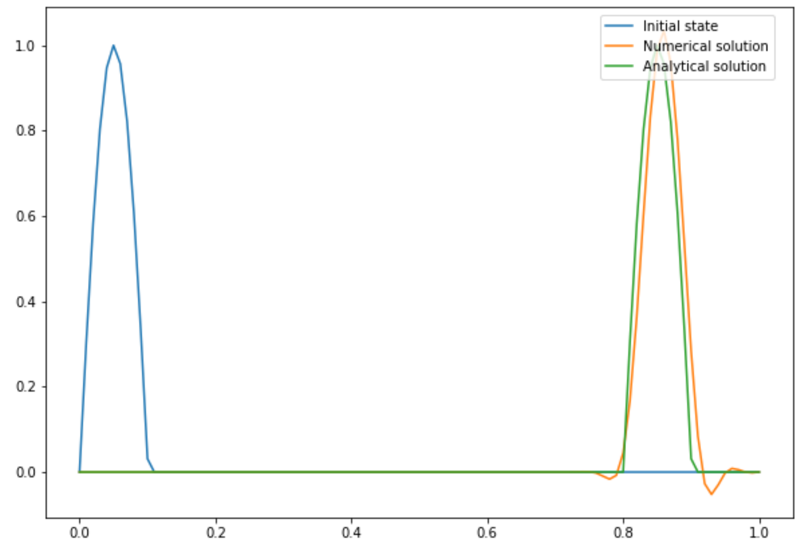


Figure 8: 1D pure advection for experiment 1 based on Li (2006).

For this experiment the velocity is set to be $u = 1$, the domain consists of 50 elements and the CFL number is 0.1. So, $\delta t = 0.002$ and the number of steps is 250. We expect the box function to reach the position where $x = \frac{3}{4}L_x$.

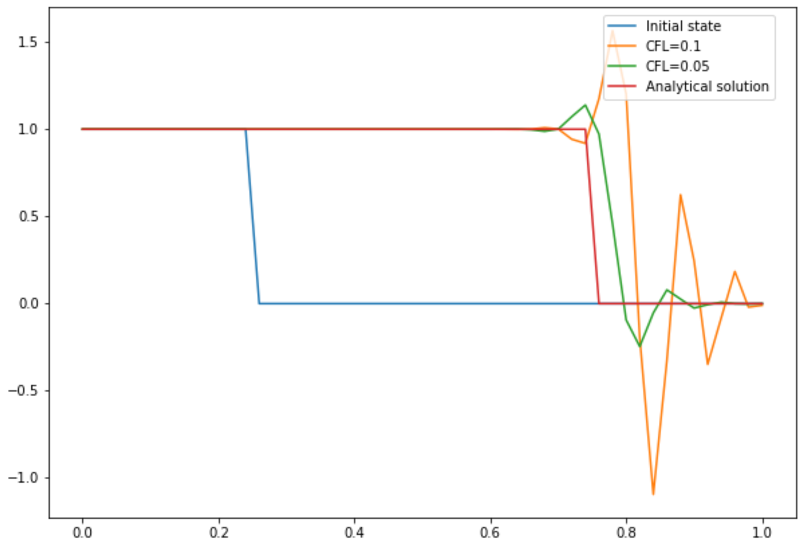


Figure 9: 1D pure advection for experiment 2 based on Donea and Huerta (2003).

Figure 9 shows the result of this experiment. Unfortunately we see that the results are worse than expected, due to the steep front we see non-negligible oscillations for this benchmark. Lowering the CFL-criterion drastically lowers these oscillations, however this will cause the computation to take longer since more time steps are required.

3.4 1D Advection-diffusion

To test this method an advection-diffusion problem is solved:

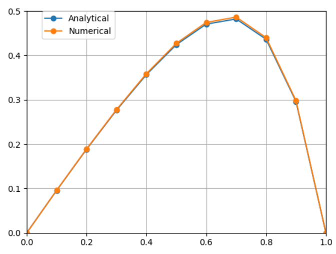
$$\rho C_p u \frac{\partial T}{\partial x} - k\frac{\partial^2 T}{\partial x^2} = H \quad T(x=0)=0 \quad T(x=L_x) \quad x \in [0.L_x] \tag{19}$$

Here C_p is the heat capacity, ρ the density, H the heat production and u is the velocity. This domain is characterised by Lx=1, nelx=10, H=1,ρ=1, C_p=1 and u=1. For this problem we use the Péclet number to determine the different advection-diffusion ratios:

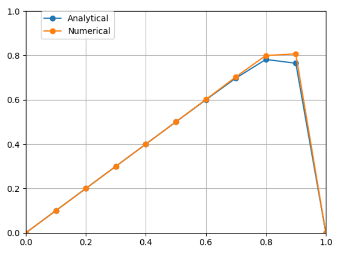
$$Pe = \frac{uh\rho C_p}{2k} \tag{20}$$

where h is the size of one element and k is the thermal conductivity. This test based on Donea and Huerta (2003) will use a Péclet number of 0.25, 0.9 and 5, since all other variables are set only the thermal conductivity varies. So, the inputs for k are 0.2, 0.055 and 0.01. We can compare these results with the analytical solution which is given by:

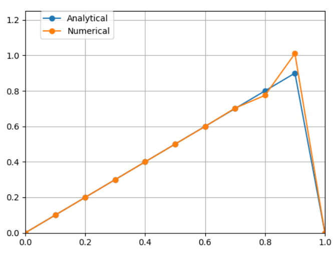
$$T(x) = \frac{H}{u}\left(x - \frac{1-e^{\frac{2Pex}{h}}}{1-e^{\frac{2Pe}{h}}}\right) \tag{21}$$



(a) Pe=0.25



(b) Pe=0.9



(c) Pe=5

Figure 10: Numerical and analytical results for the advection-diffusion problem with various Péclet numbers.

As can be seen in figure 10 there is a very solid overlap between the analytical solution and the numerical solution. Especially for the cases where the Péclet number is higher, thus where the diffusion component is more important.

3.5 2D steady state diffusion

When we expand the 1D cases to 2D the mesh also changes. This study uses both triangles and squares on a structured grid. We start from the 2D steady state heat diffusion equation:

$$\nabla \cdot (k\nabla T) + H = 0 \tag{22}$$

How this equation is implemented for the model can be found in appendix D. To be able to apply the DG method on this equation we take very similar steps as the 1D case. However the resulting system is a lot larger and more complex. To test this model we set up a simple model

$$T(x=0)=0 \qquad T(x=1)=1 \qquad \text{on} \quad x \in [0,1] \tag{23}$$

When solving for this problem we can also obtain the result of the heat flux $\vec{q}$, which is used each iteration to calculate the new temperature.

3.6 2D pure advection

For a pure advection case we start with the fairly simple equation:

$$\frac{dT}{dt} + \vec{u}\nabla T = 0 \tag{24}$$

This equation is converted to a DG finite element formulation in Appendix E, the resulting set of equations is then used to solve the problem for a bending beam as in stone 43 of the fieldstone manual by Cedric Thieulot (Thieulot, 2019). The setup of this experiment is in Figure 11a. the domain is 1000×1000 km, which has been put on a grid of 50×50 elements. $T = 0$ at all the boundaries except the left boundary where $T = 1$ if $200km < y < 800km$ and 0 elsewhere. The initial temperature field is a square of dimensions $800kmx600km$, the velocity is 0 cm/year at the left wall and increases to be 1 cm/year on the right wall. The results for this test can be found in figure 11b.

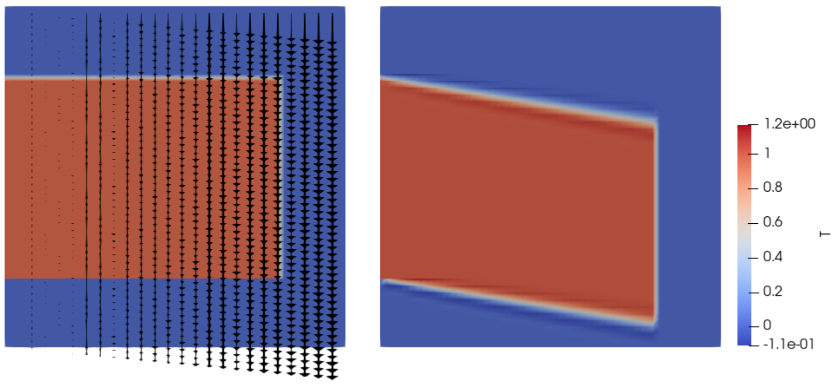


Figure 11: a 2-D DG-FEM pure advection case a. initial state with velocity vectors b. Final state

There is still a small wiggle in this advection problem (Figure 11b) but very small compared to Figure 12 which uses a normal continuous Galerkin FEM and one where the SUPG algorithm has been applied to. Which is noteworthy since the 1D test for a pure convection model had a larger instability.

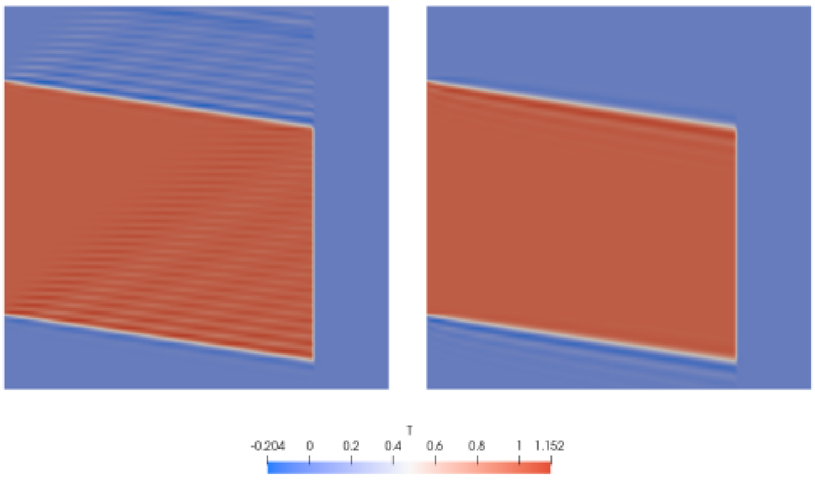


Figure 12: 2-D pure advection case a. Without the use of the SUPG algorithm b. With the SUPG algorithm. (Thieulot, 2019)

3.7 2D Advection-Diffusion

On it's own both the advection and diffusion seem to be working, so the next logical step is combining these two methods. The derivation of this combination of the energy equation can be found in F, for both when solving only for temperature or with an iterative solver in the form of $A\vec{x} = \vec{b}$. This is then applied to various problems, similar to stone 65 in the Fieldstone guide (Thieulot, 2019).

3.7.1 Experiment 1

This experiment is based on Donea and Huerta (2003), the problem is stated in 13, the mesh size is 10×10, the flow is unidirectional and constant, $|u| = 1$, and the Péclet number is 10^4. The boundary conditions are:

- Bottom: $T = 0$
- Left: $T = 0$ if $y < 0.2$, $T = 1$ if $y \geq 0.2$
- Top: $\partial_y T = 0$
- Right: $\partial_x T = 0$

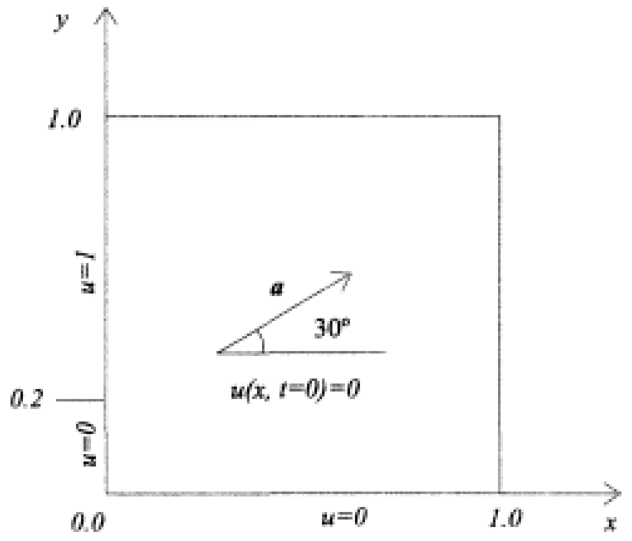


Figure 13: Experiment setup as done by Donea and Huerta (2003)

Figure 14a shows the results for the DG method with square elements, it shows a slight negative part where the system should be zero. Figure 14b shows the same test but with triangular elements. The results are very similar however the square elements seems to be a little bit more stable. The results for both these methods show negative values where there shouldn't be any. The Péclet number is so high that this system can almost be regarded as a full convective system. In the previous section we have shown pretty consistent result for a pure convection case. So, this instability is most likely due to the angle in the system. since the discontinuity is not in line with the elements.

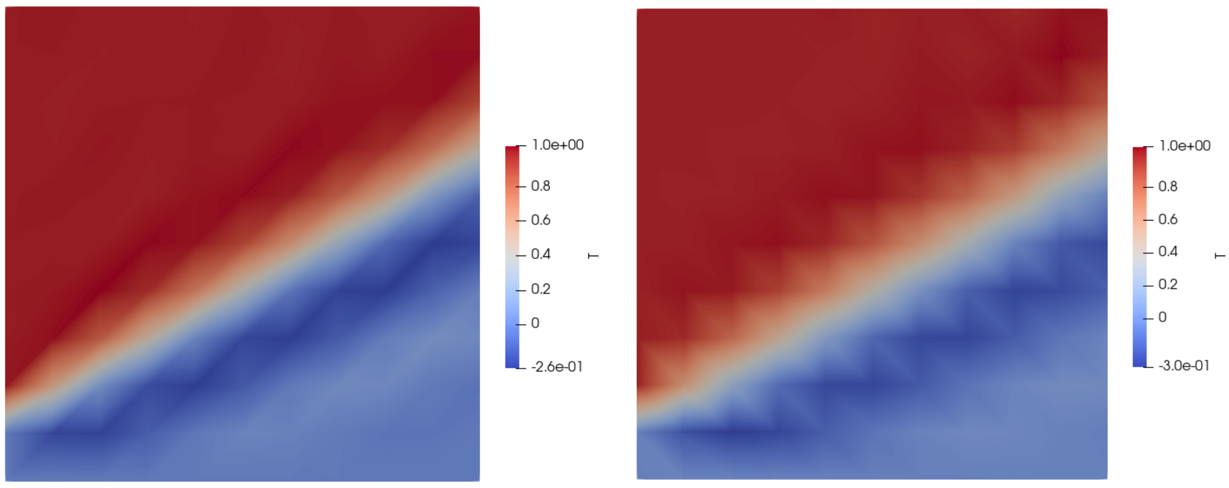


Figure 14: a) Result for the convection-diffusion for the discontinuous galerkin method with square elements b) Result for the convection-diffusion for the discontinuous galerkin method with triangular elements

3.7.2 Experiment 2

We can also apply this derivation to the problem stated by Li (2006) in Section 5.2.3. This problem states a 1x1 square with a grid of nelx $\times$ nely $= 32 \times 32$. The boundary conditions are as follows:

- Bottom: $T = 0$
- Left: $T = 0$ if $y > 0.5$, $T = 1$ if $y \leq 0.5$
- Top: $\partial_y T = 0$
- Right: $T = 0$

This experiment should be run with the advection-diffusion ratios $\eta = (\mu/\kappa) = 1, 10, 100$, this influences the Péclet number by:

$$Pe = \frac{h\eta}{2} \tag{25}$$

These various Péclet numbers are 0.15625,1.5625 and 15.625 and have been plotted in figures 15 and 16. In figure 15, the outcome of this thesis can been compared to that of Li (2006), which also uses a DG-FEM. Even though the same technique has been used to determine the heat profiles, there are some glaring differences. Especially near the boundaries of the system. For a system where diffusion is the dominating drive, so a low Péclet number, the heat profile shows some gaps.

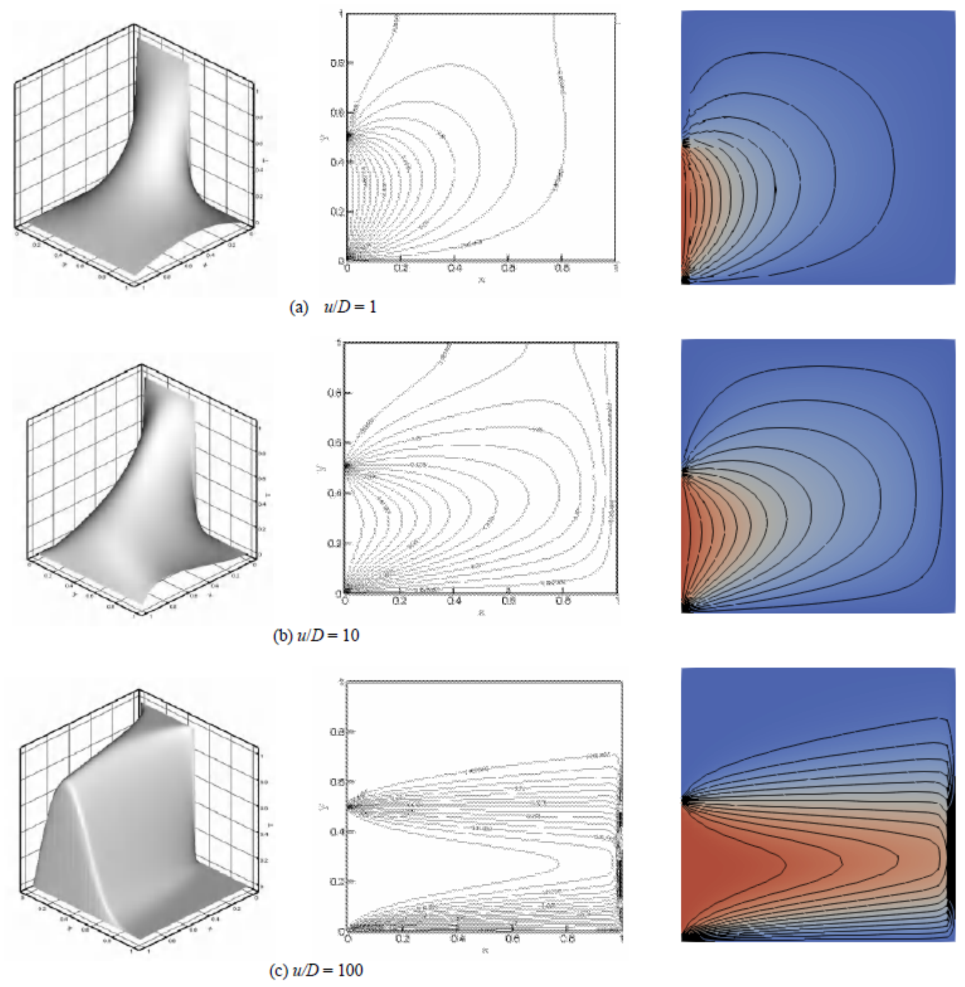
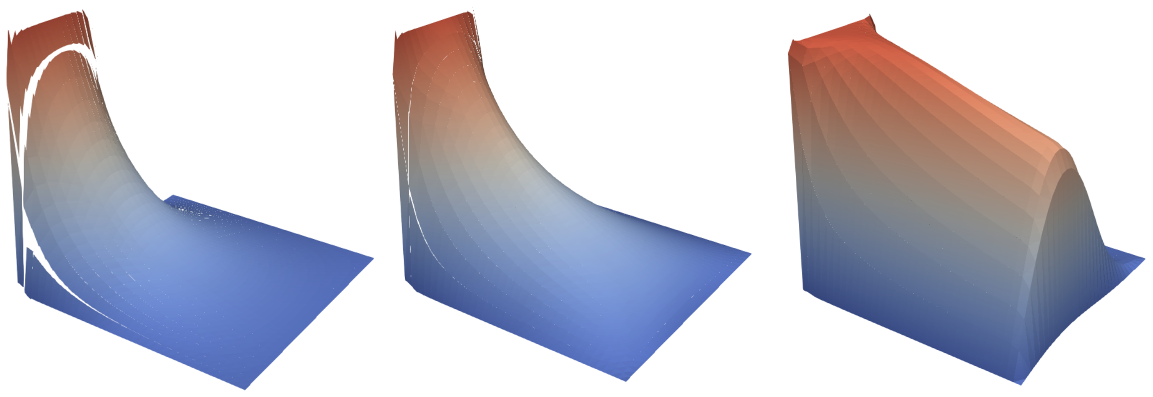


Figure 15: Experiment set up by Li (Li, 2006) on the left and results of this study on the right.



(a) Pe=0.15625 **(b)** Pe=1.56250 **(c)** Pe=15.62500

Figure 16: Numerical and analytical results for the advection-diffusion problem with various Péclet numbers.

4 Methods for the geophysical inversion

4.1 Geophysical inversion

As stated earlier this thesis not only looks at the discontinuous Galerkin but also implements the programming language Julia in order to speed up calculations. Here Julia is used to solve a geophysical inversion problem, geophysical inversion can be used to recover information about how a physical property is distributed in the subsurface. This is done by minimizing the difference between real-earth data and computed data. For this study we use a similar set up as Baumann et al. (2014). The model setup consists of a 2-D space with a solid disk which has a different density and viscosity than the surrounding material (Figure 17). For this thesis we consider the disk to be in the center of the 2D domain with dimensions 1000km in the x-direction and 500km in the z-direction.

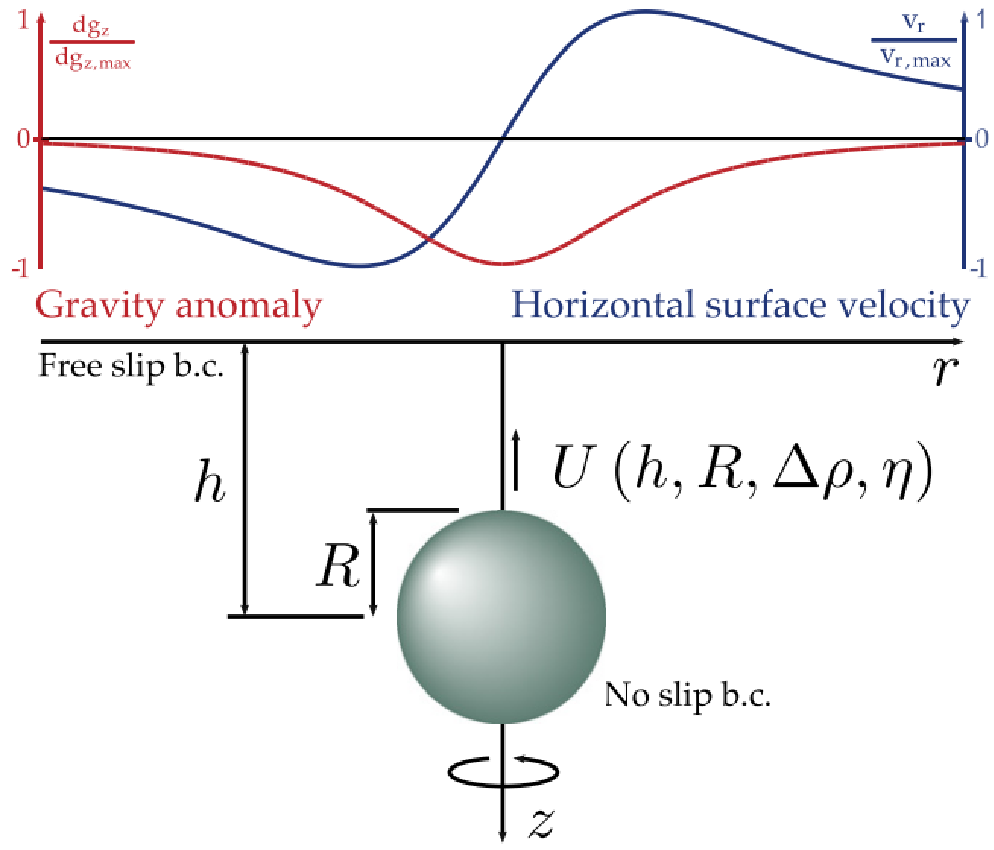


Figure 17: Model setup taken from Baumann et al. (2014)

4.1.1 Forward model: gravity

If we consider a 2D Cartesian domain, the gravity disk can be seen as an infinitely long cylinder. This is done so that we can regard the domain as being in 3D and thus compute the gravity anomaly with the help of integrating the gravity field for a point mass (Turcotte and Schubert (2002) and Jacoby and Smilde (2009)):

$$\delta\vec{g} = \int\int\int \delta g = \iiint_V G\frac{\Delta\rho}{r^2}dV \tag{26}$$

So, $\delta\vec{g}$ is the vector of the gravity anomaly, G is the gravitational constant $G \simeq 6.67 \times 10^{-11}\mathrm{N\,m^2\,kg^{-2}}$, $\Delta\rho$ is the uniform density anomaly between the inclusion and its surroundings and $r = \sqrt{x^2 + y^2 + z^2}$ or the distance of the reference point to the middle of the inclusion. Then δg_z or the vertical component of the gravity anomaly, at an angle of θ becomes:

$$\delta g_z = \delta g cos(\theta) = \iiint_V G\frac{\Delta\rho z}{r^3}dV \tag{27}$$

When regarding this over the infinitely long cylinder in the y-direction:

$$\delta g_z = \pi R^2 \Delta\rho G z \int_{\infty}^{\infty} \frac{1}{r^3}dy \tag{28}$$

Which can be reduced to:

$$\delta g = \frac{2\pi G R^2 \Delta\rho z}{(x^2 + z^2)} \tag{29}$$

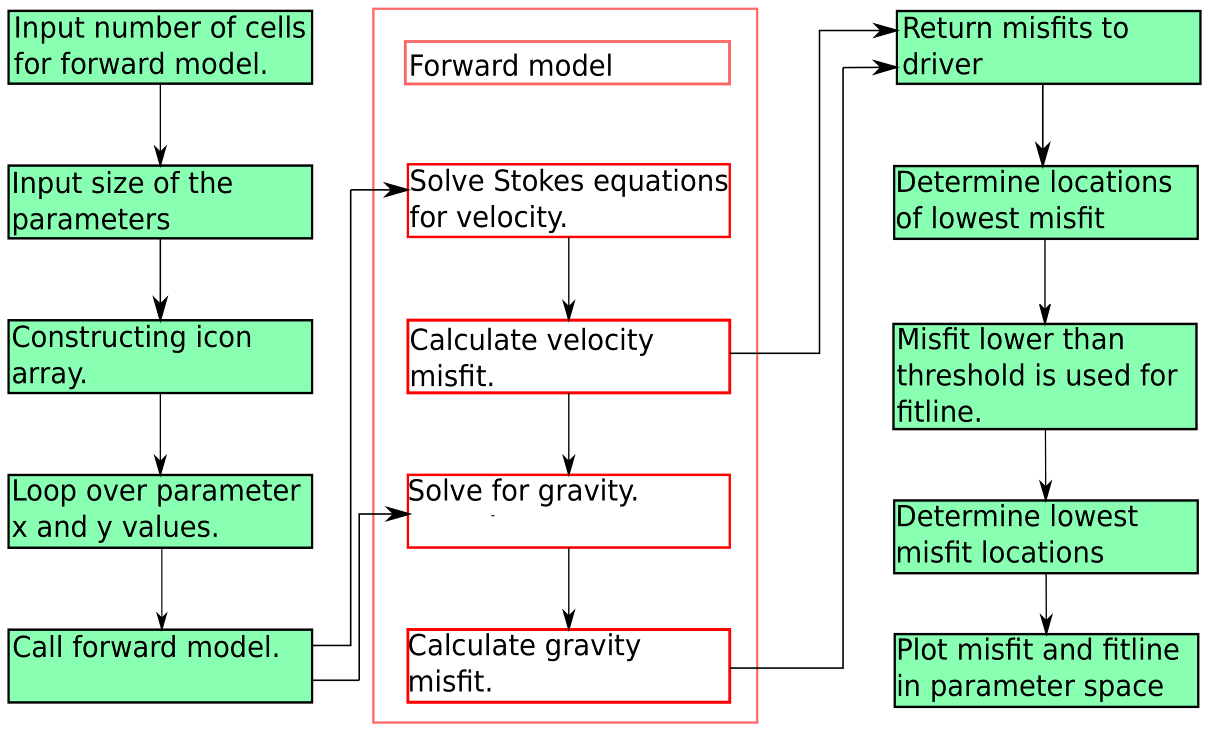
Here R is the radius of the circular inclusion. The inversion in this study will be applied to multiple variables from this formula, the radius, density difference and at which depth it is located. The depth can also be determined with the means of the half-width. The half-width is the distance between the points where the magnitude of the gravity anomaly is half the magnitude of the maximum gravity anomaly (Jacoby and Smilde, 2009).

4.1.2 Forward model: Stokes

When the sphere has a lower density than its surroundings it moves up which creates a divergent flow at the surface. The velocity throughout the whole domain is calculated with the help of a reference code which computes the velocity field at a 150×150 resolution. This velocity field is calculated by means of a simple finite elements Stokes model, based on a Python stokes model produced by Thieulot (2019). This model assumes that the fluid in the domain to be an incompressible Newtonian Fluid with a free-slip boundary along the surface. Unlike the gravity case the velocity is also dependent on the viscosity of the fluid surrounding the gravity inclusion. So, the viscosity will be another parameter for which the velocity field is solved. From the velocity field the root mean squared velocity through the whole domain, the v_{rms} at the surface and the average absolute velocity at the surface are calculated.

4.1.3 Inversion

This study uses a very simple inversion technique, there is no use of refinement or an optimization algorithm. The outline of how the code works can be found in figure 18, it starts by defining the precision of the forward model and that of the parameters put into the driver. The driver is what varies the parameters used for the forward model. The driver can use a variety of parameters, the depth, the radius or the density difference between the inclusion and its surroundings. Only 2 different parameters can be used to construct a grid for which at each node the misfits are determined. Besides these parameters the driver can also solve for various viscosity's. When the parameters are passed onto the forward model, the velocity is solved first. This is done by solving the incompressible flow Stokes equations, which gives a velocity field. This can be used to determine the root mean squared velocity for the whole domain, the root mean squared velocity for the surface and the average velocity at the surface. Which is then compared to reference code and a misfit is then determined. Then the gravity field is solved, this is then compared to the analytical solution for the gravity, with the real values of the anomaly as input. All the misfits are returned to the driver code which plots all the misfit values in a parameter space. Then 4th order polynomial fit lines for the points in these parameter spaces with the lowest misfit are determined and where these fit lines intersect is the most likely point for the true values of the gravity anomaly.



Input number of cells for forward model.
Input size of the parameters
Constructing icon array.
Loop over parameter x and y values.
Call forward model.
Forward model
Solve Stokes equations for velocity.
Calculate velocity misfit.
Solve for gravity.
Calculate gravity misfit.
Return misfits to driver
Determine locations of lowest misfit
Misfit lower than threshold is used for fitline.
Determine lowest misfit locations
Plot misfit and fitline in parameter space

Figure 18: Flowchart for the inversion model

4.2 Julia programming language

The Julia programming language is designed to be easy to use and interactive. Julia offers help with the trade-off between fast executing and fast coding, this is done with the help of the just-in-time (JIT) compilers. A JIT compiler is part of the run-time interpreter, which is a way of compiling that occurs during the executing of the program. Whereas a normal compiler converts the source code in its entirety to machine code. Julia is a relatively new programming language so it has a less extensive library than for example Python, but Julia libraries are written in the Julia language, which makes it less necessary to learn other programming languages. In order to familiarize with this programming language a tutorial was made (Appendix G). To determine how well Julia holds up compared to python, the inversion described in the previous section is put to the test. We use various mesh sizes to determine which calculations of the codes take up the most of the computation time, both the relative and the absolute time these calculations take.

5 Results for geophysical inversion

5.1 Inversion parameters

In this study several several parameters are compared starting with the radius and the density difference between the anomaly and its surroundings (Figure 19). The true values for the depth, radius, density difference and viscosity can be found in Table 5

In figure 19a we see the true location of the anomaly, in Figure 19b represent the misfit of the gravity, this gravity profile does not have a unique solution. In Figures 19d,e,f we see the misfit for the root mean square

Radius	50 km
Depth	250 km
Density difference	300 $\frac{kg}{m^3}$
viscosity	10^{21} Pa s

Table 5: Parameters for true location of the inclusion

velocity for the whole system, the root mean squared velocity at the surface and the average velocity at the surface respectively. These all show a very similar misfit distribution, which also doesn't have a unique solution. In Figure 19c, the misfit fit line of both the gravity and that of the v_{rms} have been plotted. These fit lines have a very similar trend however they are slightly different so they intersect. The point where these two lines intersect is corresponds with the radius and density difference of the true anomaly. Since the fit lines are so similar the point where the lines intersect has a high uncertainty. In figure 20, the radius and the depth of the anomaly is varied. This results once again in a non-unique gravity and velocity misfit. However for this case the fit lines for the misfits are very distinct. This causes that there are two points where the lines intersect so there is still not an unique solution for these two parameters. The inversion for the depth of the inclusion and the density difference is in figure 21, here we once again see the best fit lines for the lowest misfit. There are once again two intersections between these fit lines so we don't end up with a true unique solution. However in figure 21 b, the gravity seems to be more focused towards the center which means it tends to have a unique solution.

radius (m)	49754	density difference	-303
depth (m)	149445	density difference	-366
	248106		-301
depth (m)	166197	radius (m)	53878
	242293		50066

Table 6: Intersection points for the various parameter comparisons.

The location of the overlap for all the parameters in there separate configurations can be found in Table 6. The program also determines the point where the misfit is lowest for both the gravity and velocity, table 7, ideally this point will be located at the true parameter values from table 5. From table 7 we can deduce that the lowest misfit point does not always correlate with the point where the inclusion should be located. This seems to be the case for all the v_{rms} cases and for the radius-density difference inversion, which does correlate with the fact that these inversions are non-unique. The results for the other gravity inversions seem to be pretty close to the true values of the anomaly.

	gravity minimum		v_{rms} minimum	
radius-density difference	84900	-104	40805	-440
depth - density difference	248819	-302	159055	-351
depth - radius	252153	52283	187156	54903

Table 7: Intersection points for the various parameter comparisons.

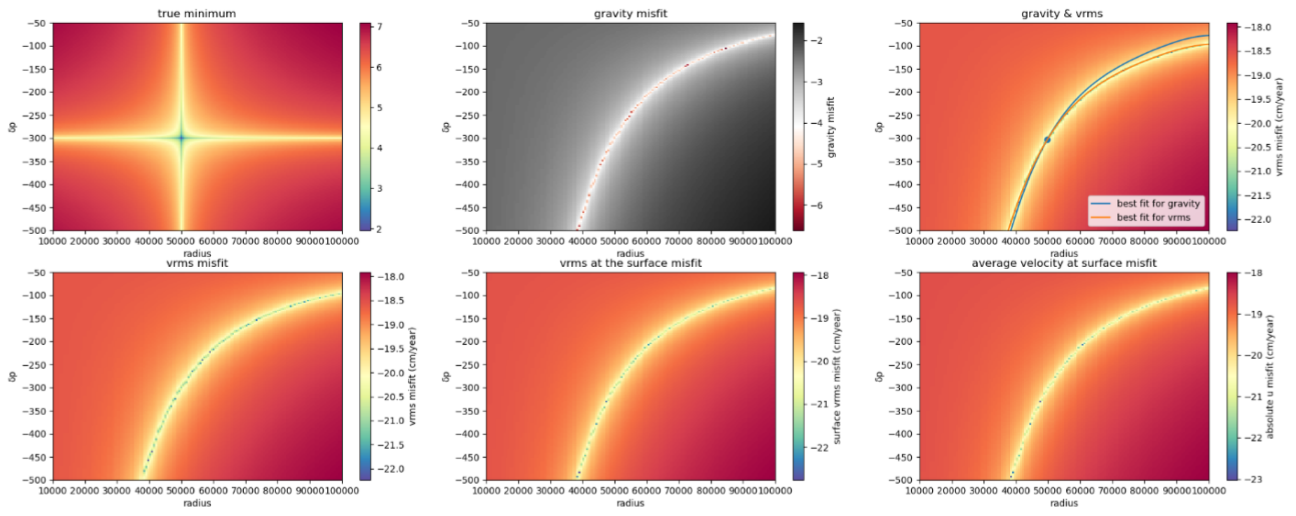


Figure 19: Results for several inversion parameters for the radius and the density difference with a resolution of 150x150. a. The true location of the gravity inclusion $\delta\rho = -300$ and the radius is 50km. b. misfit for the gravity inversion. c.misfit for the v_{rms} for the whole domain and the best fit lines for gravity and the v_{rms} d. v_{rms} for the whole domain. e. misfit for the v_{rms} at the surface. f. misfit for the average velocity at the surface.

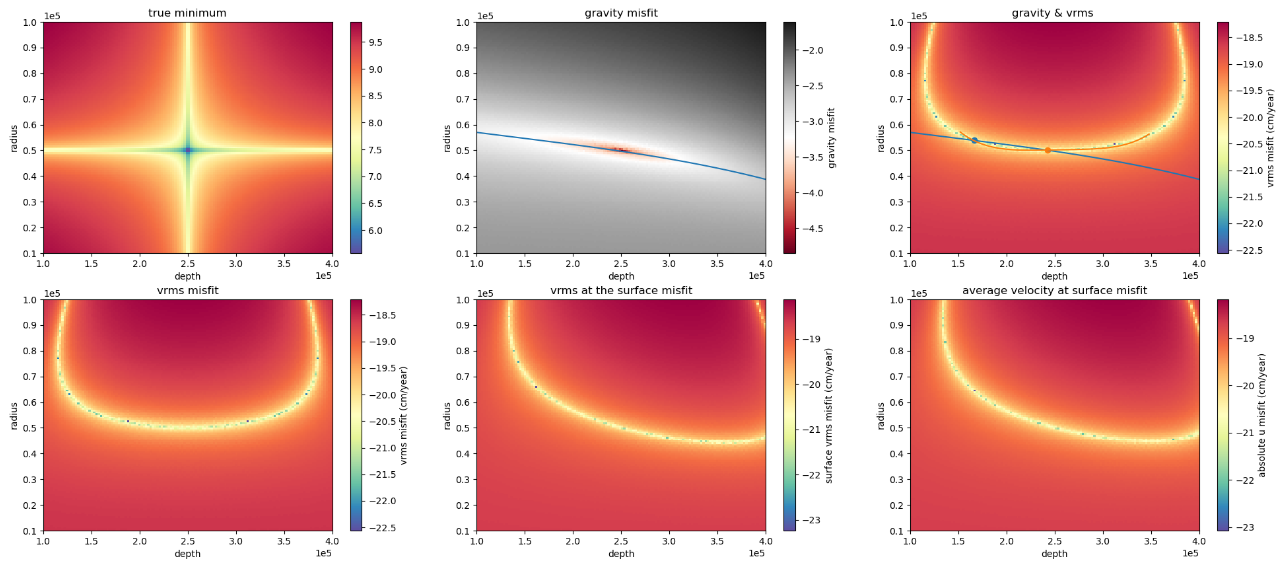


Figure 20: Results for several inversion parameters for the radius and the depth of the inclusion with a resolution of 128x128. a. The true location of the gravity inclusion is at a depth at 250km and the radius is 50km. b. misfit for the gravity inversion. c.misfit for the v_{rms} for the whole domain and the best fit lines for gravity and the v_{rms} d. v_{rms} for the whole domain. e. misfit for the v_{rms} at the surface. f. misfit for the average velocity at the surface.

5.2 Julia vs Python

In order to compare Julia to python, each forward calculation has been split up in parts. The mesh setup, which constructs a mesh of size nnx×nny. The construction of the connectivity matrix, or how the local numbering is related to the whole system. Than the boundary conditions have to be defined. After defining the b.c. all the arrays are created which will be filled with values later on, when dealing with large matrices n>100, some arrays have to be sparse in order to save memory space. Then the Finite Element matrix is build, here the entire matrix A and vector b are filled in with values. These are than made sparse in order to preserve RAM storage for the solving of the system. The solving of the system $\mathbf{A}\vec{x} = \vec{b}$, there are several methods to solve such as system, in this case we use a direct solver. From this solve a horizontal and vertical velocity is obtained which

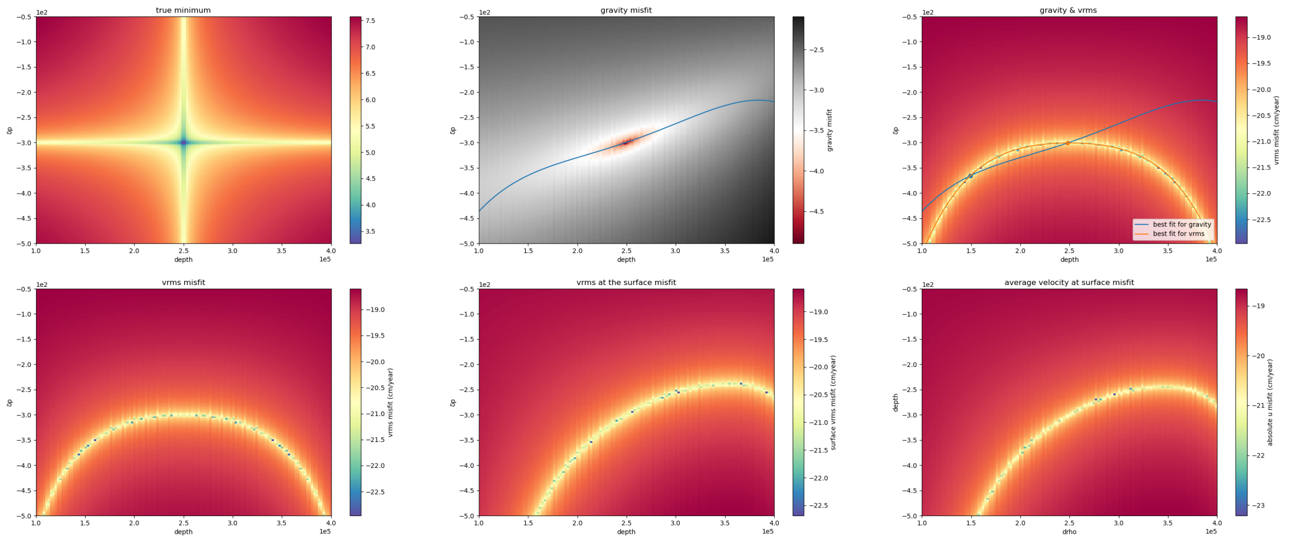


Figure 21: Results for several inversion parameters for the depth of the inclusion and the density difference with a resolution of 128x128. a. The true density of the gravity inclusion $\delta\rho = -300$ and the depth is 250km. b. misfit for the gravity inversion. c.misfit for the v_{rms} for the whole domain and the best fit lines for gravity and the v_{rms} d. v_{rms} for the whole domain. e. misfit for the v_{rms} at the surface. f. misfit for the average velocity at the surface.

than can be used to determine the v_{rms} for the whole system, the v_{rms} at the surface and the average velocity at the surface, this is than compared to a reference to determine the misfit. Finally the gravity is computed which is also compared to reference in order to determine the misfit. The relative time difference for each of these processes is shown in figure 22, this is done for both Julia and Python with varying grid sizes. From this figure we can deduce that especially the v_{rms} calculation is relatively a lot faster for the Julia programming language. The solving of the system is relatively faster for python, both programming languages are consistent in the relative time differences for each process.

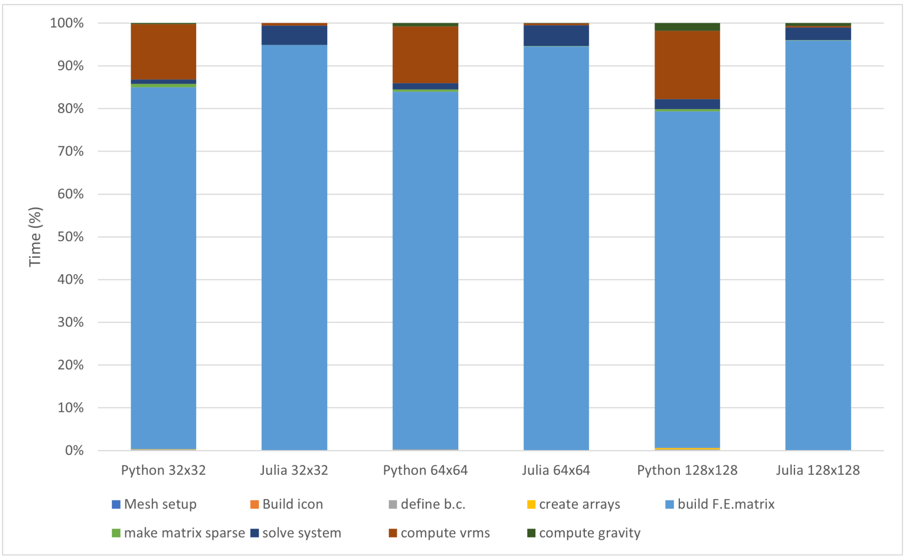


Figure 22: The relative main contributors to the calculation of a FEM calculation for both Julia and Python for various grid sizes.

When comparing Figure 22 with the true calculation times for a single FEM calculation Figure 23, it becomes even more evident that the v_{rms} calculation takes a lot of time for the python calculation. Whereas the Julia calculation mainly consists of the building of the Finite element matrix and the v_{rms} doesn't really show up in the Julia calculation.

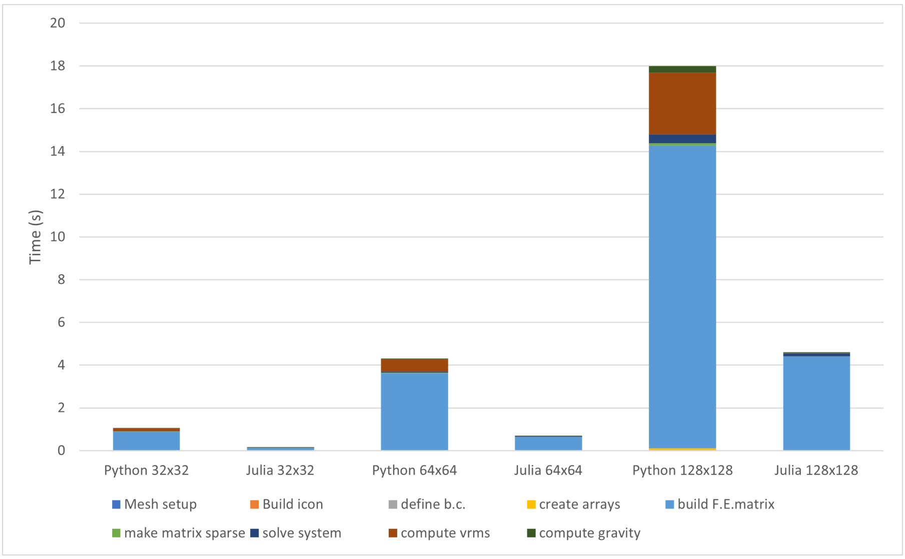


Figure 23: The main contributors to the calculation of a FEM calculation for both Julia and Python for various grid sizes.

When just looking at the total time each calculation takes for various grid sizes, Figure 24, it is clear that the python code increases linearly. Whereas the Julia code has a more quadratic code. This would indicate that there is a point where the python code becomes faster than the Julia code. By using the equations which prescribe this fitline we are able to calculate this crossover point. For Julia the fitline is $y = 9 \times 10^{-9}x^2 + 0.0002x - 0.3453$ and for python $y = 2 \times 10^{-9}x^2 + 0.0011x - 0.5079$, this means that the points where these lines cross are at a number of cells of 181 and 128391. So a nelx×nely of approximately 13×13 and 358×358.

In order to compare a Julia code with a python code some of the stones from fieldstone have been translated from Python to Julia, an important part of these codes is the construction of the connectivity matrix this can be found in G.10

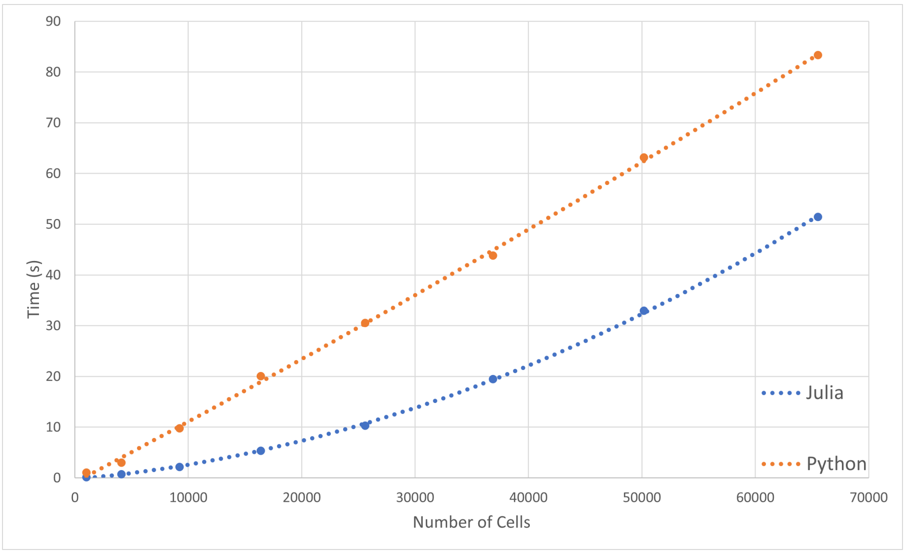


Figure 24: Time it takes for one iteration of the forward model, for various numbers of cells, nelx × nely is 256 × 256 is the maximum.

6 Discussion

6.1 Discontinuous Galerkin Method

The decision to improve the way discontinuities are dealt for the heat equation, is a trade off between precision and computation time. The degrees of freedom are double that of the CG-FEM, however when we eliminate q and solve only for the variable T, the calculation certainly speeds up. Unfortunately, stability has to be ensured. According to Renaud et al. (2020), the stabilization parameter $\mathcal{E}$ has to be larger than $\frac{\textbf{\# of elements}}{\textbf{size of 1 element}}$, otherwise the calculation will become numerically unstable.

6.1.1 Numerical flux

Another important decision to be made with regard to the discontinuous Galerkin is the numerical flux. There have been multiple studies, such as Arnold et al. (2000), to determine which numerical fluxes, maintain consistency and stability. In this study the flux proposed by Cockburn and Shu (1998) has been used, this is one of the fluxes which has proven to be suitable for the numerical solution of steady state heat conduction problems (Arnold et al., 2002). Multiple studies (e.g. Li (2006) and Cockburn et al. (2000) have stated that in order for the fluxes to be stable we have the impose weak boundary conditions.

6.1.2 Boundary conditions

From all cases it becomes evident that the boundary conditions play a vital role in the numerical calculation. We assumed Dirichlet and Neumann boundary conditions. For the Dirichlet boundary conditions we specify the value that the new temperature distribution should have along the boundary. For the Neumann boundary condition, the values of the heat flow across the boundary of the new temperature is specified. Implementing this will lead to a different weak form of the diffusion equation, because we assumed the heat flow to be zero in equation 3. If there would be no heat flow across the boundary we would expect that the energy within the system would remain constant, this is however not the case because we implement boundaries from the analytic solution. In the 2-D diffusion (Figure 4) case we see that there is a small difference between the analytic solution and the numerical solution. This difference most likely comes from numerical instability and how we deal with heat flux across the boundary.

6.1.3 The DG FEM or the CG FEM

As stated before this study wasn't able to construct a successful DG method, the 1-D cases worked but the 2D cases did not show the expected results and took quite longer than with a CG method. This is mainly due to the added degrees of freedom, the fact that we have multiple unknowns when solving for second order terms, such as diffusion and the method of solving. As can be seen in figure 3 the DOFs for large 2D grids are nearly 4 times as much for the DG method. For pure conduction problems the DG method requires it to be split up in two unknowns, as is done for temperature and the heat flux in Appendix A. So when solving this we have more unknowns and thus this effects the execution speed. As stated before, this study used iterative solvers to solve the DG method, a direct solver might be more succesful however this would in turn require a lot more RAM. Comparing the outcome of the two for a pure convection case as in Figures 9 and 25, we see that where the numerical instability for the CG case is mainly located after the flow front, whereas for the DG case the instability is further than the front, which shouldn't be possible. In 25, the instability is counteracted by the introduction of the alpha parameter which introduces some numerical diffusion. This alpha parameter is a factor between fully explicit, where the state of a system at a later time is calculated from the state of the system at the current time, $\alpha = 0$ Or fully implicit, $\alpha = 1$.

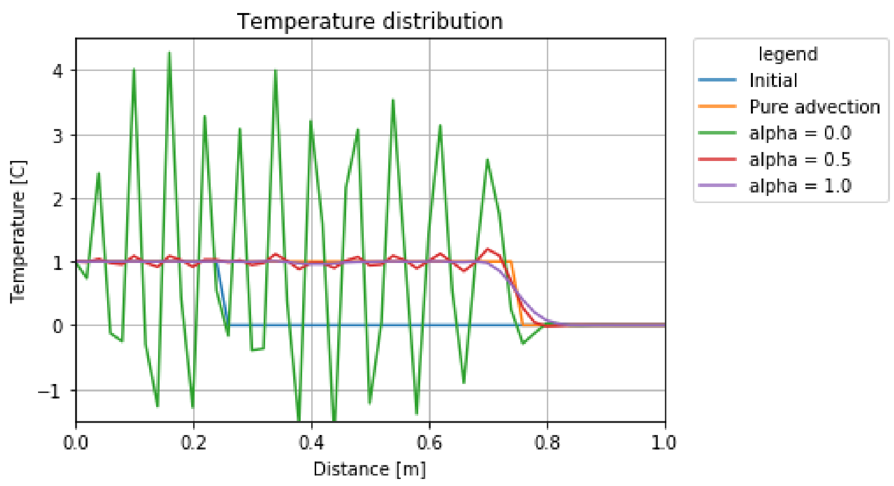


Figure 25: a cG-FEM pure advection case as in 9, and how the numerical instability can be surpressed with the α parameter.

6.1.4 Thoughts about the DG-FEM

In this study we attempted to successfully implement a 2D advection-diffusion model which was capable of solving the heat transport equation. However when comparing this model with benchmarks designed by the community, the model did poorly. This method has shown some promise for a 1D steady state advection-diffusion case, the results for the Donea and Huerta (2003) benchmark of section 3.4 were very positive and affirmed the method for such the DG method. However the increased iterations required for solving the system of equations., grew significantly when expanding from 1D to 2D. Not only the time for the computations grew but also expanding the system of equations from 1D to 2D proved difficult. The original plan was expanding the 2D advection-diffusion code from simple benchmarks to a subducting slab system, however since it has shown to be more difficult than expected this plan was dropped.

6.2 Inversion

In order to put a newly constructed Julia code to the test a geophysical inversion model was constructed, based on a study by Baumann et al. (2014). This fairly simple model is a joint inversion which means that we integrate multiple datasets in this case the velocity and the gravity. By determining these datasets with the help a gravity and Stokes model and varying the parameters of the depth, radius and density difference of the gravity anomaly to compare it with the true values of this anomaly. A joint inversion was necessary since solving for temperature alone will give a non-unique solution. This model was pretty successful in retrieving the values of the anomaly, which was to be expected since it's a fairly simple model. Even though this model was easy to solve, increasing the mesh of the forward model or the mesh in the parameter space caused it to take progressively longer. This can be a problem for inverse models since they become computationally expansive very fast, to counteract this one might try to use an optimization algorithm (Klessens, W., 2018). However these relative high computational costs are ideal for testing the difference between two programming languages.

6.3 Julia vs Python

The comparison between Julia showed some glaring results. Julia seems to be a very promising programming language, it is indeed a lot faster than python approximately a factor of 4 faster. However this seems to be decreasing when increasing the number of cells. So, in computation time Julia definitely gets a plus. It is also fairly easy to implement certainly when you have a background in FORTRAN90 and python. Since it is very similar to these two languages and almost feels like a combination of the two. In Appendix G.10, a comparison between a Python and a Julia code has been made. The most glaring differences are the numbering, that for python we have to import numpy and the fact that we have to declare a variable within the for loop scope for Julia. This comes from the fact that Julia tries to prevent variable naming conflicts between separate scopes. Another benefit is the ease-of-use for Julia, implementing a code feels very natural and it remains readable. Since Julia has a smaller database than python, its packages are also less extensive. However by using PyCall one can use python packages directly in Julia, this comes at a cost of it's speed but is ideal for plotting for example. Julia's JIT compiler is very promising, since all the code compiles just before it is executed. Julia can also be helpful for students new to programming, since it has a more readable syntax. Unfortunately, this also means that when changing the code there is a fixed compile cost, which makes iterative development less productive.

7 Conclusion

The discontinuous galerkin FEM didn't show the promise that was expected. It showed some promising results for the 1D advection-diffusion equation where it did not have any of the instability problems. However when expanding to 2D, the derivation becomes very large and more difficult to implement and the DoF's in a 2D case are almost 4 times as many as for the CG-FEM. The 2D advection-diffusion test showed different results than expected and for diffusion dominated problems showed such large discontinuities between the elements which are not desirable for these tests. We expected that the DG would be capable of dealing with sharp boundaries but both in 1D and in 2D the model became numerically unstable which is not desirable. A non-optimized inversion can take a lot of time so using a programming language which is 3-4 times as fast is very beneficial. So, I would say that Julia shows a lot of promise as a tool for Geosciences students, mainly due to it's ease-of-use and faster computations than python. Tools wise the two languages are very similar and translating an python code to Julia can be a good first step to finite element modelling.

References

Anderson, J. D. and Wendt, J. (1995). *Computational fluid dynamics*, volume 206. Springer.

Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, D. (2000). Discontinuous galerkin methods for elliptic problems. In *Discontinuous Galerkin Methods*, pages 89–101. Springer.

Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D. (2002). Unified analysis of discontinuous galerkin methods for elliptic problems. *SIAM journal on numerical analysis*, 39(5):1749–1779.

Bathe, K.-J. (2001). *Computational fluid and solid mechanics*. Elsevier.

Baumann, T. S., Kaus, B. J., and Popov, A. A. (2014). Constraining effective rheology through parallel joint geodynamic inversion. *Tectonophysics*, 631:197–211.

Bezanson, J., Chen, J., Chung, B., Karpinski, S., Shah, V. B., Vitek, J., and Zoubritzky, L. (2018). Julia: Dynamism and performance reconciled by design. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–23.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

Bijwaard, H., Spakman, W., and Engdahl, E. R. (1998). Closing the gap between regional and global travel time tomography. *Journal of Geophysical Research: Solid Earth*, 103(B12):30055–30078.

Brooks, A. N. and Hughes, T. J. (1982). Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Computer methods in applied mechanics and engineering*, 32(1-3):199–259.

Cercato, M. (2009). Addressing non-uniqueness in linearized multichannel surface wave inversion. *Geophysical Prospecting*, 57(1):27–47.

Cockburn, B., Karniadakis, G., and Shu, C.-W. (2000). *Discontinuous Galerkin Methods. Theory, Computation and Applications*. Springer.

Cockburn, B., Karniadakis, G. E., and Shu, C.-W. (2012). *Discontinuous Galerkin methods: theory, computation and applications*, volume 11. Springer Science & Business Media.

Cockburn, B. and Shu, C.-W. (1998). The local discontinuous galerkin method for time-dependent convection-diffusion systems. *SIAM Journal on Numerical Analysis*, 35(6):2440–2463.

Donea, J. and Huerta, A. (2003). *Finite Element Methods for Flow Problems*. John Wiley & Sons.

Hager, B. H. and O'Connell, R. J. (1979). Kinematic models of large-scale flow in the earth's mantle. *Journal of Geophysical Research: Solid Earth*, 84(B3):1031–1048.

He, Y., Puckett, E. G., and Billen, M. I. (2017). A discontinuous galerkin method with a bound preserving limiter for the advection of non-diffusive fields in solid earth geodynamics. *Physics of the Earth and Planetary Interiors*, 263:23–37.

Hoffman, N. and McKenzie, D. (1985). The destruction of geochemical heterogeneities by differential fluid motions during mantle convection. *Geophysical Journal International*, 82(2):163–206.

Incropera, F. P., DeWitt, D. P., Bergman, T. L., Lavine, A. S., et al. (1996). *Fundamentals of heat and mass transfer*, volume 6. Wiley New York.

Jackson, D. D. (1979). The use of a priori data to resolve non-uniqueness in linear inversion. *Geophysical Journal International*, 57(1):137–157.

Jacoby, W. and Smilde, P. L. (2009). *Gravity interpretation: Fundamentals and application of gravity inversion and geological interpretation*. Springer Science & Business Media.

Kaus, B., Berlie, N., Churavy, V., Cosarinsky, M., Duretz, T., Kiss, D., Kozdon, J., de Montserrat, A., Moser, L., Medinger, N., et al. (2022). How composable software tools in julia help developing multi-physics codes in geodynamics. Technical report, Copernicus Meetings.

Klessens, W. (2018). Joint Gravity and Stokes inversion of a 2D circular anomaly: a Delaunay triangulation of parameter space. Bachelor thesis.

Kronbichler, M., Heister, T., and Bangerth, W. (2012). High accuracy mantle convection simulation through modern numerical methods. *Geophysical Journal International*, 191:12–29.

Li, B. Q. (2006). *Discontinuous finite elements in fluid dynamics and heat transfer*. Springer Science & Business Media.

Mulyukova, E., Steinberger, B., Dabrowski, M., and Sobolev, S. V. (2015). Survival of llsvps for billions of years in a vigorously convecting mantle: Replenishment and destruction of chemical anomaly. *Journal of Geophysical Research: Solid Earth*, 120(5):3824–3847.

Nagar, S., Nagar, and Anglin (2017). *Beginning Julia Programming*. Springer.

Puckett, E. G., Turcotte, D. L., He, Y., Lokavarapu, H., Robey, J. M., and Kellogg, L. H. (2018). New numerical approaches for modeling thermochemical convection in a compositionally stratified fluid. *Physics of the Earth and Planetary Interiors*, 276:10–35.

Reddy, J. N. (1993). An introduction to the finite element method. *New York*.

Reddy, J. N. (2010). *Principles of continuum mechanics: A study of conservation principles with applications*. Cambridge University Press.

Reddy, J. N. and Gartling, D. K. (2010). *The finite element method in heat transfer and fluid dynamics*. CRC press.

Renaud, A., Heuzé, T., and Stainier, L. (2020). Stability properties of the discontinuous galerkin material point method for hyperbolic problems in one and two space dimensions. *International Journal for Numerical Methods in Engineering*, 121(4):664–689.

Ricard, Y. and Bercovici, D. (2009). A continuum theory of grain size evolution and damage. *Journal of Geophysical Research: Solid Earth*, 114(B1).

Schubert, G. (1992). Numerical models of mantle convection. *Annual review of fluid mechanics*, 24:359–394.

Schubert, G., Turcotte, D. L., and Olson, P. (2001). *Mantle convection in the Earth and planets*. Cambridge University Press.

Tackley, P. and Xie, S. (2003). Stag3d: a code for modeling thermo-chemical multiphase convection in earth's mantle. In *Computational Fluid and Solid Mechanics 2003*, pages 1524–1527. Elsevier.

Thieulot, C. (2019). Fieldstone: The Finite Element Method in Computational Geodynamics.

Tikhonov, A. N. (1963). On the solution of ill-posed problems and the method of regularization. In *Doklady Akademii Nauk*, volume 151, pages 501–504. Russian Academy of Sciences.

Turcotte, D. L. and Schubert, G. (2002). *Geodynamics*. Cambridge university press.

Vozoff, K. and Jupp, D. (1975). Joint inversion of geophysical data. *Geophysical Journal International*, 42(3):977–991.

Wilcox, L. C., Stadler, G., Burstedde, C., and Ghattas, O. (2010). A high-order discontinuous galerkin method for wave propagation through coupled elastic–acoustic media. *Journal of Computational Physics*, 229(24):9373–9396.

Witte, P. A., Louboutin, M., Kukreja, N., Luporini, F., Lange, M., Gorman, G. J., and Herrmann, F. J. (2019). A large-scale framework for symbolic implementations of seismic inversion algorithms in julia. *Geophysics*, 84(3):F57–F71.

Appendices

A 1D Steady State

Starting with the 1D steady state heat conduction problem, expressed by the following equation:

$$\frac{d^2T}{dx^2} = 0 \tag{30}$$

This equation is usually solved with its second-order derivative, but we can also write it in its mixed form, by introducing the heat flux q (a scalar in 1D):

$$-\frac{dq}{dx} = 0 \qquad q - \frac{dT}{dx} = 0 \qquad x \in [0,1]$$

The standard approach to establish the weak forms is applied for these two first-order ODEs, this is done for an element e bound by nodes k and $k+1$ with coordinates x_k and x_{k+1}

$$-\int_{x_k}^{x_{k+1}} \frac{dq}{dx}\tilde{f}(x)dx = -[q\tilde{f}]_{x_k}^{x_{k+1}} + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}q(x)dx = 0$$

$$\int_{x_k}^{x_{k+1}} \left(q - \frac{dT}{dx}\right)\overline{f}(x)dx = \int_{x_k}^{x_{k+1}} q(x)\overline{f}(x)dx - [T\overline{f}]_{x_k}^{x_{k+1}} + \int_{x_k}^{x_{k+1}} \frac{d\overline{f}}{dx}T(x)dx = 0$$

where $\tilde{f}$ and $\overline{f}$ are test functions. The terms between the square brackets have to be examined in more detail. Inside the element, the test functions $\tilde{f}$ and $\overline{f}$ are well defined polynomials so we can compose them as:

$$\tilde{f}_k^+ = \tilde{f}(x_k^+) \qquad \tilde{f}_{k+1}^- = \tilde{f}(x_{k+1}^-) \qquad\qquad \overline{f}_k^+ = \overline{f}(x_k^+) \qquad \overline{f}_{k+1}^- = \overline{f}(x_{k+1}^-)$$

Concerning q and T, they will be the values $\hat{q}_k$ and $\hat{T}_k$ at node k and $\hat{q}_{k+1}$ and $\hat{T}_{k+1}$ at node $k+1$, and we will specify these hat quantities as follows:

$$\hat{T}_k = \begin{cases} T_k^- & k=1 \\ \frac{1}{2}(T_k^- + T_k^+) + \mathcal{C}(T_k^- - T_k^+) & k = 2,...N-1 \\ T_k^+ & k=N \end{cases}$$

$$\hat{q}_k = \begin{cases} q_k^+ - \mathcal{E}(T_k^- - T_k^+) & k=1 \\ \frac{1}{2}(q_k^+ + q_k^-) - \mathcal{E}(T_k^- - T_k^+) - \mathcal{C}(q_k^- - q_k^+) & k = 2,...N-1 \\ q_k^- - \mathcal{E}(T_k^- - T_k^+) & k=N \end{cases} \tag{31}$$

where N is the number of nodes and where $\mathcal{C}$ and $\mathcal{E}$ are two constants.

Inside an element bounded by nodes k and $k+1$, the temperature T and heat flux q are interpolated over an isoparametric linear element:

$$T_h(x) = N_k(x)T_k^+ + N_{k+1}(x)T_{k+1}^-$$

$$q_h(x) = N_k(x)q_k^+ + N_{k+1}(x)q_{k+1}^-$$

The test functions are taken to be the same as the shape functions resulting in:

$$
\begin{aligned}
0 &= -\hat{q}_{k+1}\tilde{f}(x_{k+1}^-) + \hat{q}_k\tilde{f}(x_k^+) + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx} q_h(x)dx \\
&= -\hat{q}_{k+1}\tilde{f}_{k+1}^- + \hat{q}_k\tilde{f}_k^+ + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}(N_k(x)q_k^+ + N_{k+1}(x)q_{k+1}^-)dx \\
&= -\hat{q}_{k+1}\tilde{f}_{k+1}^- + \hat{q}_k\tilde{f}_k^+ + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}N_k(x)dx \cdot q_k^+ + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}N_{k+1}(x)dx \cdot q_{k+1}^- \\
&= -\left(\frac{1}{2}(q_{k+1}^+ + q_{k+1}^-) - \mathcal{E}(T_{k+1}^- - T_{k+1}^+) - \mathcal{C}(q_{k+1}^- - q_{k+1}^+)\right)\tilde{f}_{k+1}^- \\
&\quad + \left(\frac{1}{2}(q_k^+ + q_k^-) - \mathcal{E}(T_k^- - T_k^+) - \mathcal{C}(q_k^- - q_k^+)\right)\tilde{f}_k^+ \\
&\quad + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}N_k(x)dx \cdot q_k^+ + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}N_{k+1}(x)dx \cdot q_{k+1}^- \\
&= \left[-(0.5+\mathcal{C})\tilde{f}_{k+1}^- + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}N_{k+1}(x)dx\right] q_{k+1}^- - \mathcal{E}\tilde{f}_{k+1}^- T_{k+1}^- \\
&\quad + \left[(0.5+\mathcal{C})\tilde{f}_k^+ + \int_{x_k}^{x_{k+1}} \frac{d\tilde{f}}{dx}N_{k+1}(x)dx\right] q_k^+ + \mathcal{E}\tilde{f}_k^+ T_k^+ \\
&\quad - \left(\frac{1}{2}q_{k+1}^+ + \mathcal{E}T_{k+1}^+ + \mathcal{C}q_{k+1}^+)\right)\tilde{f}_{k+1}^- + \left(\frac{1}{2}q_k^- - \mathcal{E}T_k^- - \mathcal{C}q_k^-\right)\tilde{f}_k^+
\end{aligned}
$$

$$
\begin{aligned}
&\int_{x_k}^{x_{k+1}} \begin{pmatrix} \frac{dN_k}{dx}N_k & \frac{dN_k}{dx}N_{k+1} \\ \frac{dN_{k+1}}{dx}N_k & \frac{dN_{k+1}}{dx}N_{k+1} \end{pmatrix} dx \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^+ \\ (\mathcal{C}-\frac{1}{2})q_{k+1}^- \end{pmatrix} + \begin{pmatrix} \mathcal{E}T_k^+ \\ \mathcal{E}T_{k+1}^- \end{pmatrix} \\
&= \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^- \\ (\mathcal{C}+\frac{1}{2})q_{k+1}^+ \end{pmatrix} + \begin{pmatrix} \mathcal{E}T_k^- \\ \mathcal{E}T_{k+1}^+ \end{pmatrix} \qquad (32)
\end{aligned}
$$

$$
\begin{aligned}
0 &= -[T\overline{f}]_{x_k}^{x_{k+1}} + \int_{x_k}^{x_{k+1}} q_h(x)\overline{f}(x)dx + \int_{x_k}^{x_{k+1}} \frac{d\overline{f}}{dx}T_h(x)dx \\
&= -\hat{T}_{k+1}\overline{f}(x_{k+1}^-) + \hat{T}_k\overline{f}(x_k^+) + \int_{x_k}^{x_{k+1}} q_h(x)\overline{f}(x)dx + \int_{x_k}^{x_{k+1}} \frac{d\overline{f}}{dx}T_h(x)dx \\
&= -\left(\frac{1}{2}(T_{k+1}^- + T_{k+1}^+) + \mathcal{C}(T_{k+1}^- - T_{k+1}^+)\right)\overline{f}_{k+1} + \left(\frac{1}{2}(T_k^- + T_k^+) + \mathcal{C}(T_k^- - T_k^+)\right)\overline{f}_k^+ \\
&\quad + \int_{x_k}^{x_{k+1}} (N_k(x)q_k^+ + N_{k+1}(x)q_{k+1}^-)\overline{f}(x)dx + \int_{x_k}^{x_{k+1}} \frac{d\overline{f}}{dx}(N_k(x)T_k^+ + N_{k+1}(x)T_{k+1}^-)dx
\end{aligned}
$$

$$
\begin{aligned}
&\int_{x_k}^{x_{k+1}} \begin{pmatrix} N_kN_k & N_kN_{k+1} \\ N_{k+1}N_k & N_{k+1}N_{k+1} \end{pmatrix} dx \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \int_{x_k}^{x_{k+1}} \begin{pmatrix} \frac{dN_k}{dx}N_k & \frac{dN_k}{dx}N_{k+1} \\ \frac{dN_{k+1}}{dx}N_k & \frac{dN_{k+1}}{dx}N_{k+1} \end{pmatrix} dx \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} \\
&+ \begin{pmatrix} (\frac{1}{2}-\mathcal{C})T_k^+ \\ -(\mathcal{C}+\frac{1}{2})T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\mathcal{C}+\frac{1}{2})T_k^- \\ (\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix} \qquad (33)
\end{aligned}
$$

rewriting equations 32 and 33, by taking $\tilde{f} = \{N_k, N_{k+1}\}$ and $\overline{f} = \{N_k, N_{k+1}\}$ then we obtain four equations For each element with the four unknowns q_k^+, q_{k+1}^-, T_k^+ and T_{k+1}^-. All other q and T quantities in the above equations will need to find their way to the rhs.

We will also use the results obtained in Appendix A:

$$
\boldsymbol{M}^e = \int_{\Omega_e} \vec{N}^T\vec{N}dV = \int_{\Omega_e} \begin{pmatrix} N_kN_k & N_kN_{k+1} \\ N_{k+1}N_k & N_{k+1}N_{k+1} \end{pmatrix} dV = \frac{h}{2}\frac{1}{3}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \frac{h}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}
$$

and also

$$\boldsymbol{K}^e = \int_{\Omega_e} \begin{pmatrix} \frac{dN_k}{dx} N_k & \frac{dN_k}{dx} N_{k+1} \\ \frac{dN_{k+1}}{dx} N_k & \frac{dN_{k+1}}{dx} N_{k+1} \end{pmatrix} dV = \frac{1}{2} \begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}$$

Filling this into equations 32 and 33, gives

$$\boldsymbol{K}^e \cdot \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^+ \\ (\mathcal{C}-\frac{1}{2})q_{k+1}^- \end{pmatrix} + \begin{pmatrix} \mathcal{E}T_k^+ \\ \mathcal{E}T_{k+1}^- \end{pmatrix} = \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^- \\ (\mathcal{C}+\frac{1}{2})q_{k+1}^+ \end{pmatrix} + \begin{pmatrix} \mathcal{E}T_k^- \\ \mathcal{E}T_{k+1}^+ \end{pmatrix}$$

$$\boldsymbol{M}^e \cdot \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \boldsymbol{K}^e \cdot \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\frac{1}{2}-\mathcal{C})T_k^+ \\ -(\mathcal{C}+\frac{1}{2})T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\mathcal{C}+\frac{1}{2})T_k^- \\ (\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix}$$

Which becomes

$$\begin{pmatrix} \mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & \mathcal{C} \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \begin{pmatrix} \mathcal{E} & 0 \\ 0 & \mathcal{E} \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\frac{1}{2}-\mathcal{C})q_k^- + \mathcal{E}T_k^- \\ (\frac{1}{2}+\mathcal{C})q_{k+1}^+ + \mathcal{E}T_{k+1}^+ \end{pmatrix} \tag{34}$$

and

$$\frac{h}{6} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \begin{pmatrix} -\mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & -\mathcal{C} \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\frac{1}{2}+\mathcal{C})T_k^- \\ (\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix} \tag{35}$$

Combining these equations gives the numerical implementation:

$$\begin{pmatrix} \frac{h}{3} & \frac{h}{6} & -\mathcal{C} & -\frac{1}{2} \\ \frac{h}{6} & \frac{h}{3} & \frac{1}{2} & -\mathcal{C} \\ \mathcal{C} & -\frac{1}{2} & \mathcal{E} & 0 \\ \frac{1}{2} & \mathcal{C} & 0 & \mathcal{E} \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \\ T_k^+ \\ T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\frac{1}{2}+\mathcal{C})T_k^- \\ (\frac{1}{2}-\mathcal{C})T_{k+1}^+ \\ -(\frac{1}{2}-\mathcal{C})q_k^- + \mathcal{E}T_k^- \\ (\frac{1}{2}+\mathcal{C})q_{k+1}^+ + \mathcal{E}T_{k+1}^+ \end{pmatrix}$$

At the boundaries of the investigated region, the boundary conditions have to be incorporated. This gives the numerical implementation for the first element:

$$\begin{pmatrix} \frac{h}{3} & \frac{h}{6} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{h}{6} & \frac{h}{3} & \frac{1}{2} & -\mathcal{C} \\ \frac{1}{2} & -\frac{1}{2} & \mathcal{E} & 0 \\ \frac{1}{2} & \mathcal{C} & 0 & \mathcal{E} \end{pmatrix} \begin{pmatrix} q_1^+ \\ q_2^- \\ T_1^+ \\ T_2^- \end{pmatrix} = \begin{pmatrix} -T_1^- \\ (\frac{1}{2}-\mathcal{C})T_2^+ \\ \mathcal{E}T_1^- \\ (\frac{1}{2}+\mathcal{C})q_2^+ + \mathcal{E}T_2^+ \end{pmatrix}$$

and for the last element:

$$\begin{pmatrix} \frac{h}{3} & \frac{h}{6} & -\mathcal{C} & -\frac{1}{2} \\ \frac{h}{6} & \frac{h}{3} & \frac{1}{2} & \frac{1}{2} \\ \mathcal{C} & -\frac{1}{2} & \mathcal{E} & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & \mathcal{E} \end{pmatrix} \begin{pmatrix} q_N^+ \\ q_{N+1}^- \\ T_N^+ \\ T_{N+1}^- \end{pmatrix} = \begin{pmatrix} -(\frac{1}{2}+\mathcal{C})T_N^- \\ T_{N+1}^+ \\ -(\frac{1}{2}-\mathcal{C})q_N^- + \mathcal{E}T_N^- \\ \mathcal{E}T_{N+1}^+ \end{pmatrix}$$

For the solver where the variable q is eliminated, part of these sets of equations can be rewritten to first get $q =$ which can then be plugged in to the bottom set of equations. Using equation 35 and the inverse of the Mass matrix:

$$M^{-1} = -\frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix}$$

$$\begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} = -\frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \begin{pmatrix} -\mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & -\mathcal{C} \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} - \frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \begin{pmatrix} (\mathcal{C}+\frac{1}{2})T_k^- \\ -(\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix}$$

Which results in

$$\begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} = -\frac{1}{h}\begin{pmatrix} -4\mathcal{C}-1 & -2+2\mathcal{C} \\ 2+2\mathcal{C} & 1-4\mathcal{C} \end{pmatrix}\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} - \frac{1}{h}\begin{pmatrix} 4(\frac{1}{2}+\mathcal{C})T_k^- - 2(-\frac{1}{2}+\mathcal{C})T_{k+1}^+ \\ -2(\frac{1}{2}+\mathcal{C})T_k^- + 4(-\frac{1}{2}+\mathcal{C})T_{k+1}^+ \end{pmatrix}$$

Equation 34 can also be rewritten to calculate to solve for the temperature. To achieve this the inverse of:

$$\begin{pmatrix} \mathcal{E} & 0 \\ 0 & \mathcal{E} \end{pmatrix} \rightarrow \frac{1}{\mathcal{E}}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} = -\frac{1}{\mathcal{E}}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & \mathcal{C} \end{pmatrix}\begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \frac{1}{\mathcal{E}}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} -(\frac{1}{2}-\mathcal{C})q_k^- + \mathcal{E}T_k^- \\ (\frac{1}{2}+\mathcal{C})q_{k+1}^+ + \mathcal{E}T_{k+1}^+ \end{pmatrix}$$

$$\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} = -\frac{1}{\mathcal{E}}\begin{pmatrix} \mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & \mathcal{C} \end{pmatrix}\begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \frac{1}{\mathcal{E}}\begin{pmatrix} -(\frac{1}{2}-\mathcal{C})q_k^- + \mathcal{E}T_k^- \\ (\frac{1}{2}+\mathcal{C})q_{k+1}^+ + \mathcal{E}T_{k+1}^+ \end{pmatrix}$$

The same method is applied to the boundary

B 1D transient

Starting from a simple transient 1-D heat conduction problem which is similar to the Steady state heat conduction problem only with added time dependence:

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} \tag{36}$$

Once again we split this system into two seperate first order equations:

$$\frac{\partial T}{\partial t} - \frac{\partial q}{\partial x} = 0 \quad ; \quad \frac{\partial T}{\partial x} - q = 0 \tag{37}$$

The standard approach for establishing the weak form is applied to these two first order ODEs, for an element bound by nodes k and k+1.

$$\int_{x_k}^{x_{k+1}} \left(\frac{\partial T}{\partial t} - \frac{\partial q}{\partial x} \right) \tilde{f}(x) dx = \int_{x_k}^{x_{k+1}} \tilde{f}(x) \frac{\partial T}{\partial t} dx - [q\tilde{f}]_{x_k}^{x_{k+1}} + \int_{x_k}^{x_{k+1}} \frac{\partial \tilde{f}}{\partial x} q dx = 0$$

$$\int_{x_k}^{x_{k+1}} \left(-q + \frac{dT}{dx} \right) \overline{f}(x) dx = - \int_{x_k}^{x_{k+1}} q(x) \overline{f}(x) dx + [T\overline{f}]_{x_k}^{x_{k+1}} - \int_{x_k}^{x_{k+1}} \frac{d\overline{f}}{dx} T(x) dx = 0$$

Here $\tilde{f}$ and $\overline{f}$ are test functions, inside the element these test functions are well defined polynomials.

$$\tilde{f}_k^+ = \tilde{f}(x_k^+) \qquad \tilde{f}_{k+1}^- = \tilde{f}(x_{k+1}^-) \qquad \qquad \overline{f}_k^+ = \overline{f}(x_k^+) \qquad \overline{f}_{k+1}^- = \overline{f}(x_{k+1}^-)$$

Filling this in on the element boundaries results in:

$$\int_{x_k}^{x_{k+1}} \left(\tilde{f}(x) \frac{\partial T}{\partial t} + \frac{\partial \tilde{f}}{\partial x} q \right) dx - \hat{q}_{k+1} \tilde{f}(x_{k+1}^-) + \hat{q}_k \tilde{f}(x_k^+) = 0 \tag{38}$$

$$\int_{x_k}^{x_{k+1}} \left(q(x) \overline{f}(x) + \frac{\partial \overline{f}}{\partial x} T \right) dx - \hat{T}(x_{k+1}^-) \overline{f}(x_{k+1}^-) + \hat{T}(x_k^+) \overline{f}(x_k^+) = 0$$

Now the numerical fluxes on the nodes can be specified as:

$$\hat{T}_k = \begin{cases} T_k^- & k = 1 \\ \frac{1}{2}(T_k^- + T_k^+) + \mathcal{C}(T_k^- - T_k^+) & k = 2, ...N-1 \\ T_k^+ & k = N \end{cases}$$

$$\hat{q}_k = \begin{cases} q_k^+ & k = 1 \\ \frac{1}{2}(q_k^+ + q_k^-) - \mathcal{C}(q_k^- - q_k^+) & k = 2, ...N-1 \\ q_k^- & k = N \end{cases}$$

Once again the temperature and the heat flux are interpolated over an isoparametric linear element. The test functions are taken to be the same as the shape functions leading to:

$$\int_{x_k}^{x_{k+1}} \begin{pmatrix} \frac{dN_k}{dx} N_k & \frac{dN_k}{dx} N_{k+1} \\ \frac{dN_{k+1}}{dx} N_k & \frac{dN_{k+1}}{dx} N_{k+1} \end{pmatrix} dx \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \int_{x_k}^{x_{k+1}} \begin{pmatrix} N_k N_k & N_k N_{k+1} \\ N_{k+1} N_k & N_{k+1} N_{k+1} \end{pmatrix} dx \begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\mathcal{C} + \frac{1}{2}) q_k^+ \\ (\mathcal{C} - \frac{1}{2}) q_{k+1}^- \end{pmatrix} = \begin{pmatrix} (\mathcal{C} + \frac{1}{2}) q_k^- \\ (\mathcal{C} + \frac{1}{2}) q_{k+1}^+ \end{pmatrix} \tag{39}$$

$$\int_{x_k}^{x_{k+1}} \begin{pmatrix} N_k N_k & N_k N_{k+1} \\ N_{k+1} N_k & N_{k+1} N_{k+1} \end{pmatrix} dx \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \int_{x_k}^{x_{k+1}} \begin{pmatrix} \frac{dN_k}{dx} N_k & \frac{dN_k}{dx} N_{k+1} \\ \frac{dN_{k+1}}{dx} N_k & \frac{dN_{k+1}}{dx} N_{k+1} \end{pmatrix} dx \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\frac{1}{2} - \mathcal{C}) T_k^+ \\ -(\mathcal{C} + \frac{1}{2}) T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\mathcal{C} + \frac{1}{2}) T_k^- \\ (\frac{1}{2} - \mathcal{C}) T_{k+1}^+ \end{pmatrix} \tag{40}$$

Using the results obtained in Appendix B, which have also been used in the previous section and filling into equations

39 and 40, gives

$$\boldsymbol{K}^e \cdot \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \boldsymbol{M}^e \cdot \begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^+ \\ (\mathcal{C}-\frac{1}{2})q_{k+1}^- \end{pmatrix} = \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^- \\ (\mathcal{C}+\frac{1}{2})q_{k+1}^+ \end{pmatrix}$$

$$\boldsymbol{M}^e \cdot \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \boldsymbol{K}^e \cdot \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \begin{pmatrix} (\frac{1}{2}-\mathcal{C})T_k^+ \\ -(\mathcal{C}+\frac{1}{2})T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\mathcal{C}+\frac{1}{2})T_k^- \\ (\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix}$$

Which becomes:

$$\begin{pmatrix} \mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & \mathcal{C} \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \boldsymbol{M}^e \cdot \begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} = \begin{pmatrix} (\mathcal{C}+\frac{1}{2})q_k^- \\ (\mathcal{C}+\frac{1}{2})q_{k+1}^+ \end{pmatrix} \tag{41}$$

$$\boldsymbol{M}^e \cdot \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} + \begin{pmatrix} -\mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & -\mathcal{C} \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} = \begin{pmatrix} -(\mathcal{C}+\frac{1}{2})T_k^- \\ (\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix} \tag{42}$$

To solve this time dependent problem we can first solve the heat flux values (q) using the Temperature of the previous timestep. To do so we take the inverse of $\boldsymbol{M}^e$ and multiply this to all other parts of equation 42:

$$\begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} = -\frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \begin{pmatrix} -\mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & -\mathcal{C} \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} - \frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \begin{pmatrix} (\mathcal{C}+\frac{1}{2})T_k^- \\ -(\frac{1}{2}-\mathcal{C})T_{k+1}^+ \end{pmatrix}$$

Which results in:

$$\begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} = -\frac{1}{h} \begin{pmatrix} -4\mathcal{C}-1 & -2+2\mathcal{C} \\ 2+2\mathcal{C} & 1-4\mathcal{C} \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} - \frac{1}{h} \begin{pmatrix} 4(\frac{1}{2}+\mathcal{C})T_k^- - 2(-\frac{1}{2}+\mathcal{C})T_{k+1}^+ \\ -2(\frac{1}{2}+\mathcal{C})T_k^- + 4(-\frac{1}{2}+\mathcal{C})T_{k+1}^+ \end{pmatrix} \tag{43}$$

The same method can be used to determine the temperatures:

$$\begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} = -\frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \begin{pmatrix} \mathcal{C} & -\frac{1}{2} \\ \frac{1}{2} & \mathcal{C} \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} - \frac{1}{h} \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \begin{pmatrix} (\frac{1}{2}-\mathcal{C})q_k^- \\ -(\frac{1}{2}+\mathcal{C})q_{k+1}^+ \end{pmatrix}$$

$$\begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} = -\frac{1}{h} \begin{pmatrix} 4\mathcal{C}-1 & -2-2\mathcal{C} \\ 2-2\mathcal{C} & 1+4\mathcal{C} \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} - \frac{1}{h} \begin{pmatrix} 4(\frac{1}{2}-\mathcal{C})q_k^- + 2(\frac{1}{2}+\mathcal{C})q_{k+1}^+ \\ -2(\frac{1}{2}-\mathcal{C})q_k^- - 4(\frac{1}{2}+\mathcal{C})q_{k+1}^+ \end{pmatrix}$$

To obtain a numerical solution for htis equation a time integrator is required. In this case we can use thefirst order Runge-Kutta scheme for an explicit time integration.

$$T_k^{l+1} = T_k^l + (\Delta t)\dot{T}_k^l$$

where l denotes the lth timestep.

$$\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix}^{l+1} = \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix}^{l} - \frac{\Delta t}{h} \begin{pmatrix} 4\mathcal{C}-1 & -2-2\mathcal{C} \\ 2-2\mathcal{C} & 1+4\mathcal{C} \end{pmatrix} \begin{pmatrix} q_k^+ \\ q_{k+1}^- \end{pmatrix} - \frac{\Delta t}{h} \begin{pmatrix} 4(\frac{1}{2}-\mathcal{C})q_k^- + 2(\frac{1}{2}+\mathcal{C})q_{k+1}^+ \\ -2(\frac{1}{2}-\mathcal{C})q_k^- - 4(\frac{1}{2}+\mathcal{C})q_{k+1}^+ \end{pmatrix} \tag{44}$$

C 1D Convection

Starting from the pure 1-D convection equation:

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = 0 \tag{45}$$

As shown before we start by discretizing the domain into a collection of elements. Then the above equation can be integrated over the element which is bounded by nodes x_k and x_{k+1}.

$$\int_{x_k}^{x_{k+1}} \left(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x}\right)\tilde{f}(x)dx = \int_{x_k}^{x_{k+1}} \tilde{f}(x)\frac{\partial T}{\partial t}dx + [uT\tilde{f}]_{x_k}^{x_{k+1}} - \int_{x_k}^{x_{k+1}} \frac{\partial \tilde{f}}{\partial x}uTdx = 0$$

with the testfunction $\tilde{f}$, inside the elements the test functions are defined by well defined polynomials.

$$\tilde{f}_k^+ = \tilde{f}(x_k^+) \qquad \tilde{f}_{k+1}^- = \tilde{f}(x_{k+1}^-)$$

$$\int_{x_k}^{x_{k+1}} \left(\tilde{f}(x)\frac{\partial T_h}{\partial t} - \frac{\partial \tilde{f}}{\partial x}uT_h\right)dx + \tilde{f}(x_{k+1})uT(T_{k+1}^-, T_{k+1}^+) - \tilde{f}(x_k)uT(T_k^-, T_k^+) = 0 \tag{46}$$

Then we can choose the numerical flux, in this case the Lax-Friedrichs flux:

$$uT(a,b) = u\frac{(a+b}{2}) - |u|\frac{(b-a)}{2}$$

when regarding $u > 0$ then this flux becomes:

$$uT(a,b) = ua$$

Filling this into equation 46 gives:

$$\int_{x_k}^{x_{k+1}} \left(\tilde{f}(x)\frac{\partial T_h}{\partial t} - \frac{\partial \tilde{f}}{\partial x}uT_h\right)dx + \tilde{f}_{k+1}^- uT_{k+1}^- - \tilde{f}_k^- uT_k^- = 0$$

Using the same shape functions as in the previous sections, this results in:

$$M^e \begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} - uK^e \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \begin{pmatrix} 0 \\ uT_{k+1}^- \end{pmatrix} - \begin{pmatrix} uT_k^- \\ 0 \end{pmatrix} = 0 \tag{47}$$

Using the same method as for the 1-D transient diffusion, we can get the elemantal temperature values for the time derivative:

$$\begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} = \frac{u}{2}M^{-1} \cdot \begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + uM^{-1} \cdot \begin{pmatrix} T_k^- \\ 0 \end{pmatrix} = 0$$

$$M^{-1} = \frac{1}{h}\begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \tag{48}$$

$$\begin{pmatrix} \dot{T}_k^+ \\ \dot{T}_{k+1}^- \end{pmatrix} = \frac{u}{h}\begin{pmatrix} -3 & -1 \\ 3 & -1 \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \frac{u}{h}\begin{pmatrix} 4 \\ -2 \end{pmatrix} T_k^- = 0$$

Using the same first order Runge-Kutta method as in the previous section,

$$T_k^{l+1} = T_k^l + (\Delta t)\dot{T}_k^l$$

Leads to the element-wise numerical implementation.

$$\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix}^{l+1} = \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix}^{l} + \frac{u\Delta t}{h}\begin{pmatrix} -3 & -1 \\ 3 & -1 \end{pmatrix} \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \frac{u\Delta t}{h}\begin{pmatrix} 4 \\ -2 \end{pmatrix} T_k^- = 0$$

When the velocity is negative the flux becomes:

$$uT(a,b) = ub$$

following the same steps as earlier this results in the following numerical implementation.

$$\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix}^{l+1} = \begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix}^{l} + \frac{u\Delta t}{h}\begin{pmatrix} 1 & -3 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} T_k^+ \\ T_{k+1}^- \end{pmatrix} + \frac{u\Delta t}{h}\begin{pmatrix} -2 \\ 4 \end{pmatrix} T_{k+1}^+ = 0$$

This problem can be solved starting from the left boundary and sweeping through all the elements. The updated values of adjacent elements are used in the calculation of the next element as soon as this element becomes available.

D 2D Steady State diffusion

Starting from the 2D steady state heat diffusion equation:

$$\nabla \cdot (k\nabla T) + H = 0 \tag{49}$$

Similar to the 1D case this equation is split up into 2 separate equations and the heat flux q is introduced.

$$\vec{\nabla} \cdot \vec{q} = -H \qquad \vec{q} = k\vec{\nabla}T \tag{50}$$

The standard approach to establish the weak form of these first-order ODEs is used. Which starts by multiplying these equations with the test functions and taking the integral over the domain Ω.

$$\int_\Omega \vec{v}\ \nabla \cdot \vec{q} dV = \int_\Omega \vec{\nabla} \cdot (\vec{v}\vec{q}) dV - \int_\Omega \vec{q}\vec{\nabla}\vec{v} dV = -\int_\Omega \vec{v} H dV \tag{51}$$

Using the divergence theorem on the first term of this equation results in:

$$\int_{\partial\Omega,j} \vec{v}\hat{q} \cdot n_j dS - \int_\Omega \vec{q}\vec{\nabla}\vec{v} dV = -\int_\Omega \vec{v} H dV \tag{52}$$

Where n_j is the outward vector to $\partial\Omega$ and $\hat{q}$ is the approximation of q on the boundary of the element.
After multiplying with the test function and taking the integral the second term of equation 50 becomes:

$$\int_\Omega \vec{q} \cdot \vec{w} dV = \int_\Omega k\vec{\nabla}T \cdot \vec{w} dV \tag{53}$$

Applying integration by parts and the divergence theorem on the right hand side of this equation leads to:

$$\int_\Omega \vec{q} \cdot \vec{w} dV = -\int_\Omega T\vec{\nabla}(k\vec{w}) dV + \int_{\partial\Omega,j} k\hat{T} n_j \cdot \vec{w} dS \tag{54}$$

Equation 52 and 54 have a $\hat{q}$ and $\hat{T}$ term, these terms denote the numerical flux which are approximations of q and T on the boundary of element j. To do so the numerical fluxes have to be selected to render the dG stable. Here the Local discontinuous Galerkin (LDG) is chosen (Cockburn and Shu, 1998).

$$\hat{q} = (q) - \mathcal{E}[T] - \mathcal{C}[q]$$
$$\hat{T} = (T) + \mathcal{C}[T]$$

Here the square brackets denote the jump operator:

$$[q] = q^{el} \cdot n^+ + q^{nb} \cdot n^-$$
$$[T] = T^{el} \cdot n^+ + T^{nb} \cdot n^-$$

The round brackets indicate the average operator

$$(q) = \frac{1}{2}(q^{el} + q^{nb})$$
$$(T) = \frac{1}{2}(T^{el} + T^{nb})$$

Here the el superscript indicates that it belongs to the element of interest and the nb denotes the values belonging to the neighbouring element. With the LDG method we can put the numerical fluxes into equation 52 and 54, this results in the final integral representation:

$$\int_\Omega q\vec{\nabla}\vec{v} dV - \int_{d\Omega,j} \vec{v} \cdot ((q) - \mathcal{E}[T] - \mathcal{C}[q]) \cdot n_j dS = \int_\Omega \vec{v} H dV \tag{55}$$

$$\int_\Omega q \cdot \vec{w} dV + \int_\Omega T\vec{\nabla}(k\vec{w}) dV - \int_{\partial\Omega,j} k((T) + \mathcal{C}[T] \cdot n_j \vec{w} dS = 0 \tag{56}$$

Inside the element T and q vary according to the shape functions so that:

$$T = \sum_{k=1}^{N_e} N_k(r) T_k \tag{57}$$

$$q_i = \sum_{k=1}^{N_e} N_k(r) q_{k,i} \tag{58}$$

The shape functions v and w are chosen to be the same and the following relations based on figure 4 are used:

$$\begin{aligned}
q^{el} &= q^{el} \cdot n^+ \\
n^+ &= -n^- \\
q^{nb} = q^{nb} \cdot n^+ &= -q^{nb} \cdot n^- \\
\mathcal{C} = \mathcal{C} \cdot n^+ &= -\mathcal{C} \cdot n^- \\
n^+ &= n_j
\end{aligned}$$

this gives the following expressions:

$$\begin{aligned}
\int_\Omega \vec{q}\vec{\nabla}\vec{v}dV - \int_{d\Omega,j} \vec{v} \cdot (\frac{1}{2}(q^{el} + q^{nb}) - \mathcal{E}(T^{el} \cdot n^+ + T^{nb} \cdot n^-) - \mathcal{C}(q^{el} \cdot n^+ + q^{nb} \cdot n^-)) \cdot n_j dS &= \int_\Omega \vec{v} H dV \\
\int_\Omega \vec{q}\vec{\nabla}\vec{v}dV - \int_{d\Omega,j} \vec{v} \cdot ((\frac{1}{2} - \mathcal{C})q^{el} + (\frac{1}{2} + \mathcal{C})q^{nb} - \mathcal{E}T^{el} + \mathcal{E}T^{nb}) \cdot n_j dS &= \int_\Omega \vec{v} H dV \\
\int_\Omega \vec{q}\vec{\nabla}\vec{v}dV - \int_{d\Omega,j} (\frac{1}{2} - \mathcal{C}) \cdot n_j N_k N_k dS \cdot \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \end{pmatrix}_{el,j} - \int_{d\Omega,j} (\frac{1}{2} + \mathcal{C}) \cdot n_j N_k N_k dS \cdot \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \end{pmatrix}_{nb,j} & \\
+ \int_{d\Omega,j} \mathcal{E} n_j N_k N_k dS \cdot \left(\vec{T}\right)_{el,j} - \int_{d\Omega,j} \mathcal{E} n_j N_k N_k dS \cdot \left(\vec{T}\right)_{nb,j} &= \int_\Omega \vec{v} H dV \\
\left(\int_\Omega (\vec{\nabla}\overline{N})\overline{N}^T dV\right) \cdot \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \end{pmatrix}_{el} - \left(\int_\Omega (\frac{1}{2} - \mathcal{C})\overline{NN}^T n_j dS\right) \cdot \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \end{pmatrix}_{el} & \\
- \left(\int_\Omega (\frac{1}{2} + \mathcal{C})\overline{NN}^T n_j dS\right) \cdot \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \end{pmatrix}_{nb} + \left(\int_{\partial\Omega} \mathcal{E}\overline{NN}^T dS\right) \cdot (\vec{T})_{el} & \\
- \left(\int_{\partial\Omega} \mathcal{E}\overline{NN}^T dS\right) \cdot (\vec{T})_{nb} &= \int_\Omega \overline{N} H dV
\end{aligned} \tag{59}$$

$$\begin{aligned}
\int_\Omega q \cdot \vec{w} dV + \int_\Omega T\vec{\nabla}(k\vec{w})dV - \int_{\partial\Omega,j} k(\frac{1}{2}(T^{el} + T^{nb}) + \mathcal{C}(T^{el} \cdot n^+ + T^{nb} \cdot n^-)) \cdot n_j \vec{w} dS &= 0 \\
\int_\Omega q \cdot \vec{w} dV + \int_\Omega T\vec{\nabla}(k\vec{w})dV - \int_{\partial\Omega,j} k(\frac{1}{2} + \mathcal{C})T^{el} + \frac{1}{2} - \mathcal{C})T^{nb})) \cdot n_j \vec{w} dS &= 0 \\
\int_\Omega q \cdot \vec{w} dV + \int_\Omega T\vec{\nabla}(k\vec{w})dV - \int_{\partial\Omega,j} k(\frac{1}{2} + \mathcal{C}) n_j N_k N_k dS \cdot \left(\vec{T}\right)_{el} - \int_{\partial\Omega,j} k(\frac{1}{2} - \mathcal{C}) n_j N_k N_k dS \cdot \left(\vec{T}\right)_{nb} & \\
\left(\int_\Omega \overline{NN}^T dV\right) \cdot \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \end{pmatrix} + \left(\int_\Omega (\vec{\nabla}(k\overline{N})) - \overline{N}^T dV\right) \cdot (\vec{T})_{el} & \\
- \left(\int_\Omega k(\frac{1}{2} + \mathcal{C})\overline{NN}^T n_j dS\right) \cdot (\vec{T})_{el} - \left(\int_\Omega k(\frac{1}{2} - \mathcal{C})\overline{NN}^T n_j dS\right) \cdot (\vec{T})_{nb} &= 0
\end{aligned} \tag{60}$$

The unknowns $\vec{q_x}, \vec{q_y}$ and $\vec{T}$ consist of all the unknown values on each of the nodes. Combining these two equations into a matrix equation results in:

$$\begin{aligned}
&\begin{pmatrix} \int_\Omega \overline{N}\overline{N}^T dV & 0 & \int_\Omega \frac{(\partial k\overline{N})}{\partial x}\overline{N}dV \\ 0 & \int_\Omega \overline{N}\overline{N}^T dV & \int_\Omega \frac{(\partial k\overline{N})}{\partial y}\overline{N}dV \\ \int_\Omega \frac{\partial \overline{N}}{\partial x}\overline{N}dV & \int_\Omega \frac{\partial \overline{N}}{\partial y}\overline{N}dV & 0 \end{pmatrix} \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \\ \vec{T} \end{pmatrix}_{el} \\
&+\sum_{i=1}^{NS} \begin{pmatrix} 0 & 0 & -(\frac{1}{2}+\mathcal{C})k\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_x dS \\ 0 & 0 & -(\frac{1}{2}+\mathcal{C})k\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_y dS \\ -(\frac{1}{2}-\mathcal{C})\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_x dS & -(\frac{1}{2}-\mathcal{C})\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_y dS & \mathcal{C}\int_{\partial\Omega_i} \overline{N}\overline{N}^T dS \end{pmatrix} \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \\ \vec{T} \end{pmatrix}_{el} \\
&+\sum_{i=1}^{NS} \begin{pmatrix} 0 & 0 & -(\frac{1}{2}-\mathcal{C})k\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_x dS \\ 0 & 0 & -(\frac{1}{2}-\mathcal{C})k\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_y dS \\ -(\frac{1}{2}+\mathcal{C})\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_x dS & -(\frac{1}{2}+\mathcal{C})\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_y dS & -\mathcal{C}\int_{\partial\Omega_i} \overline{N}\overline{N}^T dS \end{pmatrix} \begin{pmatrix} \vec{q_x} \\ \vec{q_y} \\ \vec{T} \end{pmatrix}_{nb,j} \\
&= \begin{pmatrix} 0 \\ 0 \\ \int_\Omega \overline{N}HdV \end{pmatrix}
\end{aligned} \tag{61}$$

Using the 2D bi-linear shape functions for a quadrilateral:

$$T(x,y) = N_1(x,y) + N_2(x,y) + N_3(x,y) + N_4(x,y) \tag{62}$$

with the change of variables this becomes:

$$\begin{aligned}
N_1(x,y) &= \left(\frac{x_3-x}{h_x}\right)\left(\frac{y_3-y}{h_y}\right) \rightarrow N_1(r,s) = \frac{1}{4}(1-r)(1-s) \\
N_2(x,y) &= \left(\frac{x-x_1}{h_x}\right)\left(\frac{y_3-y}{h_y}\right) \rightarrow N_2(r,s) = \frac{1}{4}(1+r)(1-s) \\
N_3(x,y) &= \left(\frac{x-x_1}{h_x}\right)\left(\frac{y-y_1}{h_y}\right) \rightarrow N_3(r,s) = \frac{1}{4}(1+r)(1+s) \\
N_1(x,y) &= \left(\frac{x_3-x}{h_x}\right)\left(\frac{y-y_3}{h_y}\right) \rightarrow N_4(r,s) = \frac{1}{4}(1-r)(1+s)
\end{aligned} \tag{63}$$

For the gradient matrix B $\left(\frac{\partial N}{\partial x,y}\right)$this leads to:

$$B(r,s) = \begin{pmatrix} -\frac{1}{2h_x}(1-s) & \frac{1}{2h_x}(1-s) & \frac{1}{2h_x}(1+s) & -\frac{1}{2h_x}(1+s) \\ -\frac{1}{2h_y}(1-r) & -\frac{1}{2h_y}(1+r) & \frac{1}{2h_y}(1+r) & \frac{1}{2h_y}(1-r) \end{pmatrix} \tag{64}$$

The change of variables should also be applied to the integration where:

$$\begin{aligned}
\int_\Omega ...dV &= \frac{h_x h_y}{4}\int_{-1}^{1}\int_{-1}^{1} ...drds \\
\int_{\partial\Omega} ..n_x dS &= \frac{h_y}{2}\int_{-1}^{1} ..ds \\
\int_{\partial\Omega} ..n_y dS &= \frac{h_x}{2}\int_{-1}^{1} ..dr
\end{aligned}$$

Applying these relations to the different parts of the matrix equation yields:

$$\left(\int_\Omega \overline{N}\overline{N}^T dV\right) = M^e = \frac{h_x h_y}{9}\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{2} & 1 \end{pmatrix}$$

$$\int_\Omega \frac{\partial \overline{N}}{\partial x}\overline{N} dV = G_x = \frac{h_y}{6}\begin{pmatrix} -2 & -2 & -1 & -1 \\ 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ -1 & -1 & -2 & -2 \end{pmatrix}$$

$$\int_\Omega \frac{\partial \overline{N}}{\partial x}\overline{N} dV = G_y = \frac{kh_x}{6}\begin{pmatrix} -2 & -1 & -1 & -2 \\ -1 & -2 & -2 & -1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 \end{pmatrix}$$

$$\int_{\partial\Omega_i} \overline{N}\overline{N}^T n_j dS = J_{j,i} = n_j C_i \tag{65}$$

where

$$C_1 = \frac{h_j}{6}\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, C_2 = \frac{h_j}{6}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \end{pmatrix}, C_3 = \frac{h_j}{6}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}, C_4 = \frac{h_j}{6}\begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The final structure of the system is thus:

$$\begin{aligned} &\begin{pmatrix} M^e & 0 & G_x \\ 0 & M^e & G_y \\ G_x & G_y & 0 \end{pmatrix}\begin{pmatrix} \vec{q_x} \\ \vec{q_y} \\ \vec{T} \end{pmatrix}_{el} + \sum_{i=1}^{NS}\begin{pmatrix} 0 & 0 & -k(\frac{1}{2}+\mathcal{C})J_{x,i} \\ 0 & 0 & -k(\frac{1}{2}+\mathcal{C})J_{y,i} \\ -(\frac{1}{2}-\mathcal{C})J_{x,i} & -(\frac{1}{2}-\mathcal{C})J_{y,i} & \mathcal{C}C_i \end{pmatrix}\begin{pmatrix} \vec{q_x} \\ \vec{q_y} \\ \vec{T} \end{pmatrix}_{el} \\ &= -\begin{pmatrix} 0 & 0 & -k(\frac{1}{2}-\mathcal{C})J_{x,i} \\ 0 & 0 & -k(\frac{1}{2}-\mathcal{C})J_{y,i} \\ -(\frac{1}{2}+\mathcal{C})J_{x,i} & -(\frac{1}{2}+\mathcal{C})J_{y,i} & -\mathcal{C}C_i \end{pmatrix}\begin{pmatrix} \vec{q_x} \\ \vec{q_y} \\ \vec{T} \end{pmatrix}_{nb,i} + \begin{pmatrix} 0 \\ 0 \\ \int_\Omega \overline{N}H dV \end{pmatrix} \end{aligned} \tag{66}$$

E 2D pure advection

For the 2D pure advection case we can start with the advection equation:

$$\frac{dT}{dt} + \vec{u}\nabla T = 0 \tag{67}$$

Similar to the 2-D diffusion equation, the domain can be discretised into a collection of elements. The above equation can be integrated over an element which results in the weak form:

$$\int_\Omega N_i^\theta \dot{T} dV + \underbrace{\int_\Omega N_i^\theta \vec{u}\nabla T dV}_{} = 0 \tag{68}$$

$$\int_{\partial\Omega_j} N_i^\theta [\vec{u}T] \cdot n_j dS - \int_\Omega \vec{u}T \cdot \nabla N_i^\theta dV \tag{69}$$

$$\underbrace{\int_\Omega N_i^\theta \dot{T} dV}_{A} - \underbrace{\int_\Omega \vec{u}T \cdot \nabla N_i^\theta dV}_{B} + \underbrace{\int_{\partial\Omega_j} N_i^\theta [\vec{u}T] \cdot n_j dS}_{C} = 0 \tag{70}$$

where $[\vec{u}T]$ is the numerical flux,in this case we choose this flux to be the Lax-Friedrichs flux:

$$[\vec{u}T] = \vec{u}\{T\} + C_u \cdot [T] \tag{71}$$

$$\{T\} = \frac{1}{2}(T^{el} + T^{nb}) \tag{72}$$

$$[T] = (T^{el}n^+ - T^{nb}n^+) \tag{73}$$

$$C_u = \frac{1}{2}|u| \tag{74}$$

The equation can be split in an x and y part:

$$\frac{1}{2}u_x(T^{el} + T^{nb}) + \frac{1}{2}|u_x|(T^{el}n_x^+ - T^{nb}n_x^+) \tag{75}$$

$$\frac{1}{2}u_y(T^{el} + T^{nb}) + \frac{1}{2}|u_y|(T^{el}n_y^+ - T^{nb}n_y^+) \tag{76}$$

if we put this in part C of equation 70 we get:

$$\begin{aligned} &\int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}u_x(T^{el} + T^{nb}) + \frac{1}{2}|u_x|(T^{el}n_x^+ - T^{nb}n_x^+)) \cdot n_j dS = \\ &\int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}u_x(T^{el} + T^{nb}) + \frac{1}{2}|u_x|(T^{el}n_x^+ - T^{nb}n_x^+)) \cdot n_x dS + \\ &\quad \int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}u_x(T^{el} + T^{nb}) + \frac{1}{2}|u_x|(T^{el}n_x^+ - T^{nb}n_x^+)) \cdot n_y dS \end{aligned} \tag{77}$$

since $n_x \cdot n_y = 0$ the second part equals 0. The same can be done for the part. This results in:

$$\int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}(u_x + |u_x|)T^{el}) \cdot n_x^+ dS + \int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}(u_x - |u_x|)T^{nb}) \cdot n_x^+ dS + \tag{78}$$

$$\int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}(u_y + |u_y|)T^{el}) \cdot n_y^+ dS + \int_{\partial\Omega_j} N_i^\theta (\frac{1}{2}(u_y - |u_y|)T^{nb}) \cdot n_y^+ dS \tag{79}$$

So from equation 70 we get:

$$\underbrace{\int_{\Omega} N^{\theta} N^{\theta} dV}_{E} \cdot \frac{dT^{el}}{dt} - \underbrace{\int_{\Omega} u_x \frac{dN^{\theta}}{dx} N^{\theta} dV}_{J_x} \cdot T^{el} + \underbrace{\int_{\Omega} u_y \frac{dN^{\theta}}{dy} N^{\theta} dV}_{J_y} \cdot T^{el} +$$
$$\underbrace{\int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} (\frac{1}{2}(u_x + |u_x|)) \cdot n_x^+ dS}_{J_{x,j}} \cdot T^{el} + \underbrace{\int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} (\frac{1}{2}(u_x - |u_x|)) \cdot n_x^+ dS}_{H_{x,j}} \cdot T^{nb} +$$
$$\underbrace{\int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} (\frac{1}{2}(u_y + |u_y|)) \cdot n_y^+ dS}_{J_{y,j}} \cdot T^{el} + \underbrace{\int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} (\frac{1}{2}(u_y - |u_y|)) \cdot n_y^+ dS}_{H_{y,j}} \cdot T^{nb} \tag{80}$$

$$\dot{T} = E^{-1}(J_x - J_y - J_{x,j} - J_{y,j}) \cdot T^{el} + E^{-1}(-H_{x,j} - H_{y,j}) \cdot T^{nb} \tag{81}$$

When we apply the first order Runge-Kutta method:

$$T_k(t + \delta t) = T_k(t) + \delta t \dot{T}_k \tag{82}$$

So,

$$T_k^{i+1} = T_k^i + \delta t(E^{-1}(J_x - J_y - J_{x,j} - J_{y,j}) \cdot T^{el} + E^{-1}(-H_{x,j} - H_{y,j})) \cdot T^{nb} \tag{83}$$

for square elements:

$$E = \left(\int_{\Omega} \overline{NN}^T dV\right) = \frac{h_x h_y}{9} \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{2} & 1 \end{pmatrix}$$

$$J_x = \left(\int_{\Omega} u_x \frac{dN^{\theta}}{dx} N^{\theta} dV\right) = \frac{u_x h_y}{12} \begin{pmatrix} -2 & -2 & -1 & -1 \\ 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ -1 & -1 & -2 & -2 \end{pmatrix} \tag{84}$$

$$J_y = \left(\int_{\Omega} u_y \frac{dN^{\theta}}{dy} N^{\theta} dV\right) = \frac{u_y h_x}{12} \begin{pmatrix} -2 & -1 & -1 & -2 \\ -1 & -2 & -2 & -1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 \end{pmatrix} \tag{85}$$

$$J_{x,j} = \int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} (\frac{1}{2}(u_x + |u_x|)) \cdot n_x^+ dS = \frac{1}{2}(u_x + |u_x|) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} \cdot n_x^+ dS}_{C_i} \tag{86}$$

$$J_{y,j} = \int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} (\frac{1}{2}(u_y + |u_y|)) \cdot n_y^+ dS = \frac{1}{2}(u_y + |u_y|) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^{\theta} N_i^{\theta} \cdot n_y^+ dS}_{C_i} \tag{87}$$

$$H_{x,j} = \int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_x - |u_x|)) \cdot n_x^+ dS = \frac{1}{2}(u_x - |u_x|)) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^\theta N_i^\theta \cdot n_x^+ dS}_{C_i} \tag{88}$$

$$H_{y,j} = \int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_y - |u_y|)) \cdot n_y^+ dS = \frac{1}{2}(u_y - |u_y|) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^\theta N_i^\theta \cdot n_y^+ dS}_{C_i} \tag{89}$$

$$C_1 = \int_{1\to 2} N_i^\theta N_i^\theta dS = \frac{h_x}{6} \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{90}$$

$$C_2 = \int_{2\to 3} N_i^\theta N_i^\theta dS = \frac{h_y}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{91}$$

$$C_3 = \int_{3\to 4} N_i^\theta N_i^\theta dS = \frac{h_x}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} \tag{92}$$

$$C_4 = \int_{4\to 1} N_i^\theta N_i^\theta dS = \frac{h_y}{6} \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix} \tag{93}$$

F 2D advection diffusion

When we regard the advection-diffusion equation:

$$\rho C_p(\frac{\partial T}{\partial t} + \vec{u} \cdot \vec{\nabla} T) = \vec{\nabla} \cdot k \vec{\nabla} T + Q \tag{94}$$

Once again we can split this into 2 first order differential equations:

$$\underbrace{\rho C_p(\frac{\partial T}{\partial t} + \vec{u} \cdot \vec{\nabla} T) + \vec{\nabla} \cdot \vec{q} = Q}_{ODE \quad 1} \quad ; \quad \underbrace{\vec{q} = -k\vec{\nabla} T}_{ODE \quad 2} \tag{95}$$

For the temperature we choose the basis function to be N_i^θ, so the temperature inside an element is given by

$$T_h(\vec{r}) = \sum_{i=1}^{N_e} N_i^\theta(\vec{r}) T_i = \vec{N^\theta} \cdot \vec{T} \tag{96}$$

For the heat fluxes the basis functions are defined by:

$$\underset{x}{q_h}(\vec{r}) = \sum_{i=1}^{N_e} N_i^q(\vec{r}) \underset{x}{q_i} = \vec{N^q} \cdot \underset{x}{\vec{q}} \tag{97}$$

$$\underset{y}{q_h}(\vec{r}) = \sum_{i=1}^{N_e} N_i^q(\vec{r}) \underset{y}{q_i} = \vec{N^q} \cdot \underset{y}{\vec{q}} \tag{98}$$

For ODE 1 this leads to the weak form of:

$$\rho C_p(\int_\Omega N_i^\theta \dot{T} dV) + \rho C_p(\int_\Omega N_i^\theta \vec{u} \nabla T dV) + \int_\Omega N_i^\theta \vec{\nabla} \cdot \vec{q} dV = \int_\Omega N_i^\theta Q dV \tag{99}$$

This shows strong resemblance to the earlier defined weak forms. So, we take similar steps. First the second and third part we apply the product rule which leads to:

$$\underbrace{\rho C_p(\int_\Omega N_i^\theta \dot{T} dV)}_{1} - \underbrace{\rho C_p(\int_\Omega \vec{u} T \cdot \nabla N_i^\theta dV)}_{2} + \underbrace{\rho C_p(\int_{\partial\Omega_j} N_i^\theta [\vec{u} T] \cdot n_j dS)}_{3}$$
$$+ \underbrace{\int_{d\Omega} N_i^\theta \vec{q_h} \cdot \vec{n} \, dS}_{4} - \underbrace{\int_\Omega \vec{\nabla} N_i^\theta \cdot \vec{q_h} \, dV}_{5} = \underbrace{\int_\Omega N_i^\theta Q dV}_{6} \tag{100}$$

The same fluxes as described before are used, which leads to:

$$\begin{aligned}
1 &= \underbrace{\rho C_p(\int_\Omega N^\theta N^\theta dV)}_{E} \cdot \frac{dT^{el}}{dt} \qquad (101) \\
2 &= -\underbrace{\rho C_p(\int_\Omega u_x \frac{dN^\theta}{dx} N^\theta dV)}_{J_x} \cdot T^{el} - \underbrace{\rho C_p(\int_\Omega u_y \frac{dN^\theta}{dy} N^\theta dV)}_{J_y} \cdot T^{el} \\
3 &= \underbrace{\rho C_p(\int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_x + |u_x|)) \cdot n_x^+ dS)}_{J_{x,j}} \cdot T^{el} + \underbrace{\rho C_p(\int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_x - |u_x|)) \cdot n_x^+ dS)}_{H_{x,j}} \cdot T^{nb} + \\
&\quad \underbrace{\rho C_p(\int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_y + |u_y|)) \cdot n_y^+ dS)}_{J_{y,j}} \cdot T^{el} + \underbrace{\rho C_p(\int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_y - |u_y|)) \cdot n_y^+ dS) back}_{H_{y,j}} \cdot T^{nb} \\
4 &= \underbrace{\left(\int_{\partial\Omega_j} \left(\frac{1}{2} - \vec{\mathcal{C}} \cdot \vec{n}^+\right) \vec{N}^T \vec{N} n_x^+ \, dS\right)}_{A_{x,j}} \cdot \vec{q}_x + \underbrace{\left(\int_{\partial\Omega_j} \left(\frac{1}{2} - \vec{\mathcal{C}} \cdot \vec{n}^+\right) \vec{N}^T \vec{N} n_y^+ \, dS\right)}_{A_{y,j}} \cdot \vec{q}_y \\
&\quad - \underbrace{\left(\int_{\partial\Omega_j} \vec{N}^T \vec{N} \mathcal{E} dS\right)}_{B_j} \cdot T^{el} + \underbrace{\left(\int_{\partial\Omega_j} \vec{N}^T \vec{N} \mathcal{E} dS\right)}_{C_j} \cdot T^{nb} \\
&\quad + \underbrace{\left(\int_{\partial\Omega_j} \left(\frac{1}{2} + \vec{\mathcal{C}} \cdot \vec{n}^+\right) \vec{N}^T \vec{N} n_x^+ \, dS\right)}_{D_{x,j}} \cdot \vec{q}_x + \underbrace{\left(\int_{\partial\Omega_j} \left(\frac{1}{2} + \vec{\mathcal{C}} \cdot \vec{n}^+\right) \vec{N}^T \vec{N} n_y^+ \, dS\right)}_{D_{y,j}} \cdot \vec{q}_y \\
5 &= \underbrace{\left(\int_\Omega \partial_x \vec{N}^T \vec{N} \, dV\right)}_{F_x} \cdot \vec{q}_x + \underbrace{\left(\int_\Omega \partial_y \vec{N}^T \vec{N} \, dV\right)}_{F_y} \cdot \vec{q}_y \\
6 &= \underbrace{\left(\int_\Omega NQ dV\right)}_{HS} \qquad (102)
\end{aligned}$$

We can do the same for the second ODE, which is the same as for the 2D diffusion case:

$$\underbrace{\int_\Omega N_i \vec{q} \, dV}_{K} + \underbrace{\int_{\partial\Omega_j} k N_i \hat{T} \, \vec{n}^+ dS}_{L} - \underbrace{\int_\Omega \vec{\nabla}(kN_i) T \, dV}_{M} = 0$$

Which can be split in a 2D Cartesian system which leads to:

$$\begin{aligned}
0 &= \underbrace{\int_\Omega N_i q_x \, dV}_{K_x} + \underbrace{\int_{\partial\Omega_j} k N_i \hat{T} \, n_x^+ \, dS}_{L} - \underbrace{\int_\Omega \partial_x(kN_i) \, T \, dV}_{M_x} \qquad (0 = K_x + L - M_x) \\
0 &= \underbrace{\int_\Omega N_i q_y \, dV}_{K_y} + \underbrace{\int_{\partial\Omega_j} k N_i \hat{T} \, n_y^+ \, dS}_{N} - \underbrace{\int_\Omega \partial_y(kN_i) \, T \, dV}_{M_y} \qquad (0 = K_y + N - M_y) \qquad (103)
\end{aligned}$$

$$
\begin{aligned}
K_x &= \left(\int_\Omega \vec{N}^T \vec{N} dV\right) \cdot \underset{x}{\vec{q}} \\
K_y &= \left(\int_\Omega \vec{N}^T \vec{N} dV\right) \cdot \underset{y}{\vec{q}} \\
M_x &= \left(\int_\Omega k\partial_x \vec{N}^T \vec{N}\; dV\right) \cdot \vec{T} \\
M_y &= \left(\int_\Omega k\partial_y \vec{N}^T \vec{N}\; dV\right) \cdot \vec{T} \\
L &= \int_{\partial\Omega_j} kN_i \left[\left(\frac{1}{2} + \vec{\mathcal{C}} \cdot \vec{n}^+\right) T_h + \left(\frac{1}{2} - \vec{\mathcal{C}} \cdot \vec{n}^+\right) T_h - \mathcal{F}(\vec{q}_h - \vec{q}_h) \cdot \vec{n}^+\right] n_x^+ dS \\
&= \underbrace{\left(\int_{\partial\Omega_j} k\left(\frac{1}{2} + \vec{\mathcal{C}} \cdot \vec{n}^+\right) N^T \vec{N} n_x^+ dS\right)}_{L_{1,j}} \cdot \vec{T} + \underbrace{\left(\int_{\partial\Omega_j} k\left(\frac{1}{2} - \vec{\mathcal{C}} \cdot \vec{n}^+\right) N^T \vec{N} n_x^+ dS\right)}_{L_{2,j}} \cdot \vec{T} \\
&\quad \underbrace{-\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_x^+ n_x^+ dS\right)}_{L_{3,x,j}} \cdot \underset{x}{\vec{q}} \underbrace{-\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_y^+ n_x^+ dS\right)}_{L_{3,y,j}} \cdot \underset{y}{\vec{q}} \\
&\quad \underbrace{\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_x^+ n_x^+ dS\right)}_{L_{4,x,j}} \cdot \underset{x}{\vec{q}} + \underbrace{\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_y^+ n_x^+ dS\right)}_{L_{4,y,j}} \cdot \underset{y}{\vec{q}} \\
N &= \int_{\partial\Omega_j} kN_i \left[\left(\frac{1}{2} + \vec{\mathcal{C}} \cdot \vec{n}^+\right) T_h + \left(\frac{1}{2} - \vec{\mathcal{C}} \cdot \vec{n}^+\right) T_h - \mathcal{F}(\vec{q}_h - \vec{q}_h) \cdot \vec{n}^+\right] n_y^+ dS \\
&= \underbrace{\left(\int_{\partial\Omega_j} k\left(\frac{1}{2} + \vec{\mathcal{C}} \cdot \vec{n}^+\right) N^T \vec{N} n_y^+ dS\right)}_{N_{1,j}} \cdot \vec{T} + \underbrace{\left(\int_{\partial\Omega_j} k\left(\frac{1}{2} - \vec{\mathcal{C}} \cdot \vec{n}^+\right) N^T \vec{N} n_y^+ dS\right)}_{N_{2,j}} \cdot \vec{T} \\
&\quad \underbrace{-\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_x^+ n_y^+ dS\right)}_{N_{3,x,j}} \cdot \underset{x}{\vec{q}} \underbrace{-\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_y^+ n_y^+ dS\right)}_{N_{3,y,j}} \cdot \underset{y}{\vec{q}} \\
&\quad \underbrace{\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_x^+ n_y^+ dS\right)}_{N_{4,x,j}} \cdot \underset{x}{\vec{q}} + \underbrace{\left(\int_{\partial\Omega_j} k\vec{N}^T \vec{N} \mathcal{F} n_y^+ n_y^+ dS\right)}_{N_{4,y,j}} \cdot \underset{y}{\vec{q}}
\end{aligned}
\tag{104}
$$

If we put ODE 2 together we get:

$$
\begin{pmatrix} K_x & 0 & M_x \\ 0 & K_y & M_y \end{pmatrix} \cdot \begin{pmatrix} \underset{x}{\vec{q}} \\ \underset{y}{\vec{q}} \\ \vec{T} \end{pmatrix} + \sum_{j=0}^{N^e} \begin{pmatrix} -L_{3,x,j} & -L_{3,y,j} & L_{1,j} \\ -N_{3,x,j} & -N_{3,y,j} & N_{1,j} \end{pmatrix} \cdot \begin{pmatrix} \underset{x}{\vec{q}} \\ \underset{y}{\vec{q}} \\ \vec{T} \end{pmatrix} \tag{105}
$$

$$
+ \sum_{j=0}^{N^e} \begin{pmatrix} L_{4,x,j} & L_{4,y,j} & L_{2,j} \\ N_{4,x,j} & N_{4,y,j} & N_{2,j} \end{pmatrix} \cdot \begin{pmatrix} \underset{y}{\vec{q}} \\ \underset{y}{\vec{q}} \\ \vec{T} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{106}
$$

We can use this to calculate $\underset{x}{q}$ and $\underset{y}{q}$:

$$\begin{pmatrix} \underset{x}{\vec{q}} \\ \underset{y}{\vec{q}} \end{pmatrix} = -\left[\begin{pmatrix} K_x & 0 \\ 0 & K_y \end{pmatrix} + \sum_{j=0}^{N^e} \begin{pmatrix} -L_{3,x,j} & -L_{3,y,j} \\ -N_{3,x,j} & -N_{3,y,j} \end{pmatrix}\right]^{-1} \left[\begin{pmatrix} M_x \\ M_y \end{pmatrix} + \sum_{j=0}^{N^e} \begin{pmatrix} L_{1,j} \\ N_{1,j} \end{pmatrix}\right] \cdot T \tag{107}$$

$$-\left[\begin{pmatrix} K_x & 0 \\ 0 & K_y \end{pmatrix} + \sum_{j=0}^{N^e} \begin{pmatrix} -L_{3,x,j} & -L_{3,y,j} \\ -N_{3,x,j} & -N_{3,y,j} \end{pmatrix}\right]^{-1} \left[\sum_{j=0}^{N^e} \begin{matrix} L_{4,x,j} & L_{4,y,j} & L_{2,j} \\ N_{4,x,j} & N_{4,y,j} & N_{2,j} \end{matrix}\right] \cdot \begin{pmatrix} \underset{y}{\vec{q}} \\ \underset{y}{\vec{q}} \\ \vec{T} \end{pmatrix} \tag{108}$$

These newly found $\underset{x}{q}$ and $\underset{y}{q}$ can then be used in ODE 1 to calculate the temperature for the next timestep:

$$\dot{T} = E^{-1}\left(\left(J_x + J_y + \sum_{j=0}^{N^e}(-J_{x,j} - J_{y,j} + B_j)\right) \cdot T + \left(\sum_{j=0}^{N^e}(-H_{x,j} - H_{y,j} - C_j)\right) \cdot T + HS\right)$$

$$+E^{-1}\left(\left(\sum_{j=0}^{N^e}(-A_{x,j}) - F_x\right) \cdot \underset{x}{q} + \left(\sum_{j=0}^{N^e}(-A_{y,j}) - F_y\right) \cdot \underset{y}{q} + \left(\sum_{j=0}^{N^e}(-D_{x,j})\right) \cdot \underset{x}{q} + \left(\sum_{j=0}^{N^e}(-D_{y,j})\right) \cdot \underset{y}{q}\right) \tag{109}$$

$$T_k^{i+1} = T_k^i + \delta t \dot{T}_k \tag{110}$$

Alternatively we can solve this iteratively, in the form of K U = F.
Each of terms for square elements:

$$K_x = K_y = \left(\int_\Omega \overline{NN}^T dV\right) = \frac{h_x h_y}{9} \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{2} & 1 \end{pmatrix}$$

$$F_x = \left(\int_\Omega \partial_x \vec{N}^T \vec{N}\, dV\right) = \frac{hy}{12} \begin{pmatrix} -2 & -2 & -1 & -1 \\ 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ -1 & -1 & -2 & -2 \end{pmatrix} ; Mx = kF_x \tag{111}$$

$$F_y = \int_\Omega \frac{\partial(\vec{N})}{\partial y} \vec{N} dV = \frac{h_x}{12} \begin{pmatrix} -2 & -1 & -1 & -2 \\ -1 & -2 & -2 & -1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 \end{pmatrix} ; M_y = kF_y$$

$$J_x = \left(\int_\Omega u_x \frac{dN^\theta}{dx} N^\theta dV\right) = \frac{u_x h_y}{12} \begin{pmatrix} -2 & -2 & -1 & -1 \\ 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ -1 & -1 & -2 & -2 \end{pmatrix} \tag{112}$$

$$J_y = \left(\int_\Omega u_y \frac{dN^\theta}{dy} N^\theta dV\right) = \frac{u_y h_x}{12} \begin{pmatrix} -2 & -1 & -1 & -2 \\ -1 & -2 & -2 & -1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 \end{pmatrix} \tag{113}$$

$$J_{xx,j} = \int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_x + |u_x|)) \cdot n_x^+ dS = \frac{1}{2}(u_x + |u_x|) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^\theta N_i^\theta \cdot n_x^+ dS}_{C_i} \tag{114}$$

$$J_{yy,j} = \int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_y + |u_y|)) \cdot n_y^+ dS = \frac{1}{2}(u_y + |u_y|) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^\theta N_i^\theta \cdot n_y^+ dS}_{C_i} \tag{115}$$

$$H_{xx,j} = \int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_x - |u_x|)) \cdot n_x^+ dS = \frac{1}{2}(u_x - |u_x|)) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^\theta N_i^\theta \cdot n_x^+ dS}_{C_i} \tag{116}$$

$$H_{yy,j} = \int_{\partial\Omega_j} N_i^\theta N_i^\theta (\frac{1}{2}(u_y - |u_y|)) \cdot n_y^+ dS = \frac{1}{2}(u_y - |u_y|) \sum_{j=1}^{4} \underbrace{\int_{\partial\Omega_j} N_i^\theta N_i^\theta \cdot n_y^+ dS}_{C_i} \tag{117}$$

$$J_{j,i} = -(\frac{1}{2} - C_{12}) \left(\int_{\partial\Omega_i} \overline{NN}^T n_j dS \right) = (\frac{1}{2} - C_{12}) n_j C_i$$

$$JB_{j,i} = -(\frac{1}{2} + C_{12}) \left(\int_{\partial\Omega_i} \overline{NN}^T n_j dS \right) = (\frac{1}{2} + C_{12}) n_j C_i$$

$$H_{j,i} = k(\frac{1}{2} + C_{12}) \left(\int_{\partial\Omega_i} \overline{NN}^T n_j dS \right) = k(\frac{1}{2} + C_{12}) n_j C_i$$

$$HB_{j,i} = -k(\frac{1}{2} - C_{12}) \left(\int_{\partial\Omega_i} \overline{NN}^T n_j dS \right) = -k(\frac{1}{2} - C_{12}) n_j C_i$$

$$B_i = C_{11} \left(\int_{\partial\Omega_i} \vec{N}\vec{N} dS \right) = C_{11} C_i$$

$$C_i = C_{11} \left(\int_{\partial\Omega_i} \vec{N}\vec{N} dS \right) = C_{11} C_i$$

$$C_{bot} = \frac{h_x}{6} \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, C_{right} = \frac{h_y}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, C_{top} = \frac{h_x}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}, C_{left} = \frac{h_y}{6} \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix}$$

G Julia tutorial

G.1 Introduction

This manual is meant to be an introduction to the programming language for geophysics students with some knowledge with programming in Python. Julia is a relatively straightforward language, similar to Python. This language is designed to be easy to use and interactive. Julia offers help with the trade-off between fast executing and fast coding, this is done with the help of the just-in-time (JIT) compilers. A JIT compiler is part of the run-time interpreter, which is a way of compiling that occurs during the executing of the program. Whereas a normal compiler converts the source code in its entirety to machine code. Julia is a relatively new programming language so it has a less extensive library than for example Python, but Julia libraries are written in the Julia language, which makes it less necessary to learn other programming languages.

G.2 executing

Executing a Julia program can be done in various ways, one can use a script, it can be run interactively in a console and it can be used with an Integrated Develepmont Environment (IDE) such as Jupyter Notebook. When using a script the source text the file name extension is .jl, so if you want to run with the use of a command line you can use:

/path/to/julia script-name.jl

With an interactive session also called a REPL (read-eval-print loop) julia can be used in a console where the commands can be entered. WHen using an IDE a julia script can be opened and run within that editor. As a first example a bit of Julia code might look like:

```
#!/usr/bin/julia
println("hello world")
```

G.3 Packages

Julia uses external packages which provide many functions, to use these packages in your Julia code you have to write using Pkg to use that packages capabilities. Once the package is added it can be used in a script with the using statement. For example:

```
#!/usr/bin/julia
using Plots
pyplot()
plot(rand(4,4))
```

Some use full packages included in Julia are: Printf, Statistics, LinearAlgebra, BenchmarkTools,SparseArrays,Plots and Formatting

G.4 Data types

Julia has a dynamic type system which means that nothing is known about the types until run time, then the values manipulated by the program become available. Julia however takes some of the advantages by allowing to indicate that certain values are of specific types. This is a great advantage for generating an efficient code. When types are omitted in Julia, Julia behaves as if these values can be any type. However it is advised to add annotations into previously untyped code, since this improves human readability and reduces the chance for errors.

G.5 Scalar Types

Julia uses several types of scalars: Int64, Float64, Char, String and Bool. It also allows most string operations split(s) , join([s1,s2], ""), replace(s, "toSearch" => "toReplace") and strip(s). When using string operations extra attention should be paid to the single quotes for Char and double quotes for strings. When one wants to concatenate strings there are several ways: by using the concatenation operator, the function string and with combining string variables using the $ symbol. Julia is also able to handle unicode characters which will allow the use of strings with advanced mathematical symbols.

G.6 Arrays

An array consists of a collection of elements from which each are identified by an array index, the type of the object is contained and it has a number of dimensions e.g. ArrayFloat64,2. Important to note is that Julia's indexing starts at 1 apposed to the 0 which is the case when using the Python programming language. Array's in Julia can be heterogeneous which means that the type is Any and it can be of the form a = [1, "hello", true].However this is much slower than an array of an other type. With two dimensional arrays or Matrices in Julia the first dimension is always interpreted as rows and the second dimension as columns. To differentiate between rows and columns one should use comma's and spacebars between the elements. For example:

- a = [[1,2,3],[4,5,6]] creates a 1-dimensional array with 2-elements (each of those is again a vector);
- a = [[1,2,3] [4,5,6]] creates a 2-dimensional array (a matrix with 2 columns) with three elements (scalars).

G.7 Control Flow

Control flow operators help shape the flow of the program. The standard types of control flow are supported with Julia (while, for, if/else, do). Unlike python one has to end these control flows with an end statement. Julia uses the logical operators && for and, || for or and ! for not. To clean up your code it is possible to use a ternary operator such as ? and : which can replace if and else statements.

G.8 Input Output

The basic method of printing is with print() or println() when a new line after the print statement is needed. To use more advanced printing methods you have to import Printf, which can be used to determine the length of the variables you want to print. For writing or reading a file you first have to open the file of interest and specify how you want to modify it(w for writing, a for appending or r for reading), when you're done with using the file you have to close() it. This can also be done by the use of a do operation, which causes the file to close when the do block ends.

```
open("afile.txt", "w") do f  # "w" for writing
  write(f, "test\n")         # \n for newline
end
```

G.9 functions

A function in Julia is an object that maps argument values to a return value, they can be defined by using the function keyword:

```
function myfunc(x)
    y=x^2
    return y
end
```

Note that indentation is not needed with Julia but it is introduced for visual appeal. The arguments passed to the function can be specified by position, but when the arguments is preceded by a semicolon than it is specified by its name.

G.10 Comparison between Python and Julia

In order to compare Julia to Python, the 2D setup of the connectivity array has been coded for both:

```
import numpy as np
nelx=16
nely=16
nel=nelx*nely
icon =np.zeros((4, nel),dtype=np.int32)
counter = 0
for j in range(0, nely):
    for i in range(0, nelx):
        icon[0, counter] = i + j * (nelx + 1)
        icon[1, counter] = i + 1 + j * (nelx + 1)
        icon[2, counter] = i + 1 + (j + 1) * (nelx + 1)
        icon[3, counter] = i + (j + 1) * (nelx + 1)
        counter += 1
```

Python code for the connectivity array

```
nelx=16
nely=16
nel=nelx*nely
icon = zeros(Int32,4,nel)
counter = 1
for j = 1:nely
    for i = 1:nelx

        icon[1, counter]= (i)+ (j-1) * (nelx +1)
        icon[2, counter] = i + 1 + (j-1) * (nelx + 1)
        icon[3, counter] = i + 1 + (j) * (nelx + 1)
        icon[4, counter] = i + (j ) * (nelx + 1)
        global(counter) += 1
    end
end
```

Julia code for the connectivity array

As stated before, the Julia and Python codes are very similar in design, the most obvious differences are that we have to import numpy for python, the numbering differences and the fact that we have to state within the loops that the counter is a global variable. This is due to the for loop introducing a new scope block. These scope blocks have been introduced to help avoiding variable naming conflicts, but since we have to give a value to the counter before the for loop we have to define it twice.

G.11 Conclusion

As shown in this short tutorial Julia is a fairly easy to learn programming language, which shares a lot of similarities with Python. But the speed Julia can achieve is similar to more difficult programming languages such as C/C++. It is less mature than any of these languages so it has less libraries. it does however have an easy interoperability with other programming languages, which means that it is able to call functions and libraries from these other languages. The Julia community is rapidly increasing and since it is open source the number of packages are rapidly increasing.