Utrecht University

Improving the numerical solution of the SSA equation in an ice flow model

Bachelor Thesis

Wiskunde
Aardwetenschappen

Author:
Kyra Brakkee

Supervisors:
Dr. Paul Zegeling
Prof. Dr. Roderik van de Wal
Dr. Tijn Berends

12 February 2021

Contents

List of used symbols **2**

1 Introduction **1**

2 Discretisation scheme on an irregular mesh. **3**
2.1 Approximating partial derivatives on a regular mesh 3
2.2 Approximations on an irregular mesh. 4
2.2.1 Notation 4
2.2.2 First partial derivative on an irregular mesh 4
2.2.3 Second partial derivative on an irregular mesh 7

3 Iteration methods **11**
3.1 Jacobi method 11
3.2 Gauss-Seidel method 11
3.3 Successive Overrelaxation (SOR) 12
3.4 Convergence of iterative methods 12

4 The Shallow Shelf Approximation (SSA) **15**

5 Demo model **18**
5.1 Demo Laplace-equation 18
5.1.1 Finding the solution using iterative methods 20
5.1.2 Convergence of different iteration methods 21
5.1.3 Adjusting ω on some vertices 22
5.2 Comparing the results with some convergence theory 23

6 Conclusion and further research **26**

List of used symbols

In this thesis, many symbols and notations are used to express some variables in the equations that are described. Since this can be a bit confusing, an overview of those symbols is given here.

N_x^i	Depends on the coordinates of v_i and its neighbours. Every vertex v_i has its own N_x^i and $N_x^i \in \mathbb{R}$.
$N_x^{i,s}$	Depends on the coordinates of v_i and its neighbours. For every neighbour of v_i, there is a $N_x^{i,s}$ and $N_x^{i,s} \in \mathbb{R}$.
N_{xx}^i	Depends on the coordinates of v_i and its neighbours. Every vertex v_i has it's own N_{xx}^i and $N_{xx}^i \in \mathbb{R}$.
$N_{xx}^{i,s}$	Depends on the coordinates of v_i and its neighbours. For every neighbour of v_i, there is a $N_{xx}^{i,s}$ and $N_{xx}^{i,s} \in \mathbb{R}$.
$\bar{v}$	The viscosity (unit: $pa.s$) is a measure for resistance against deformation. It is given by equation (4.8).
$\boldsymbol{f}^*$	The exact solution of the linear system $A\boldsymbol{f} = b$.
$\boldsymbol{f}^k$	The approximation of the solution of the linear system $A\boldsymbol{f} = b$ after k iterations.
$\boldsymbol{v}$	Vertically integrated velocity $\boldsymbol{v} = (u, v)$.
δ_{ij}	$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$.
τ	Index for the time step that is used in the model.
f_i	The function value of f on vertex i.
f_x^i	First partial derivative with respect to x on vertex v_i.
$f_{x,tri}^t$	Partial derivative with respect to x triangle t.
$f_{xx,sub}^s$	Second partial derivative with respect to x on subtriangle s.
f_{xx}^i	Second partial derivative with respect to x on vertex v_i.
k	Index for the iteration step.
s	Index for the subtriangles and w_s surrounding vertex v_i in the mesh.
t	Index for the triangles and vertices v_{nt} surrounding vertex v_i in the mesh.
u	The first component of the vertically integrated velocity $\boldsymbol{v} = (u, v)$.
v	The second component of the vertically integrated velocity $\boldsymbol{v} = (u, v)$.
$\|\|x\|\|_\infty$	Let $\boldsymbol{x}$ be a vector in $\mathbb{R}^n$. The infinity norm or maximal norm is given by $\max(\|x_1\|, \|x_2\|, ..., \|x_n\|)$.

1 Introduction

Ice sheets are continental glaciers with an area of more than 50.000 km^2. Currently, there are only two ice sheets which can be found on Greenland and Antarctica. Together, they contain more than 99% of the permanent ice volume [4], so melting of those ice sheets can contribute a lot to sea level rise. For example: melting of the whole Antarctic ice sheet would lead to a sea level rise of almost 58 metres [4]. A recent study [12] used satellite data to estimate the mass loss of the Greenland ice sheet between 1992 and 2018. They showed that the total loss has led to a sea level rise of approximately ten millimetres. In general, the ice loss was increasing over time.
However, the evolution of an ice sheet is very slow compared to the human timescale. Therefore, these kind of observations do not represent the long term evolution of ice sheets. In [13], it is stated that the response of ice sheets to the warming climate is still a large uncertainty in sea level predictions. Therefore, several ice sheet models have already been made to simulate the evolution of ice sheets.

The Utrecht Finite Volume Ice-Sheet Model (UFEMISM) is one of those models and is made by [2]. The model is different from other Ice-sheet models because it solves the equations on an unstructured triangular mesh. Figure 2 shows an example of such a mesh.
With this approach, some parts of the glacier can be modelled with high resolution, while other parts can be modelled with a low resolution.
At the boundary of the ice sheet, the ice starts to float on the ocean water. This part of the ice sheet is called an ice shelf. The location where the ice sheet starts to float is called the grounding line. According to [2], a high resolution at the grounding line of a glacier is needed to solve the physical processes accurately. However, if the whole ice sheet would be modelled with a high resolution, the computation time would be very long.
On other parts, where there is less movement in the glacier, there is no need for such a high resolution. With this unstructured triangular mesh, the resolution can be high only at the places where it is needed to get an accurate solution. The triangular mesh that is used in the model is based on the geometry of the ice sheet. The mesh refinement algorithm that is used to find the triangles of the mesh is based on an extended version of Rupperts algorithm [8, 10]. Several refinement conditions are added based on the geometry of the ice sheet.

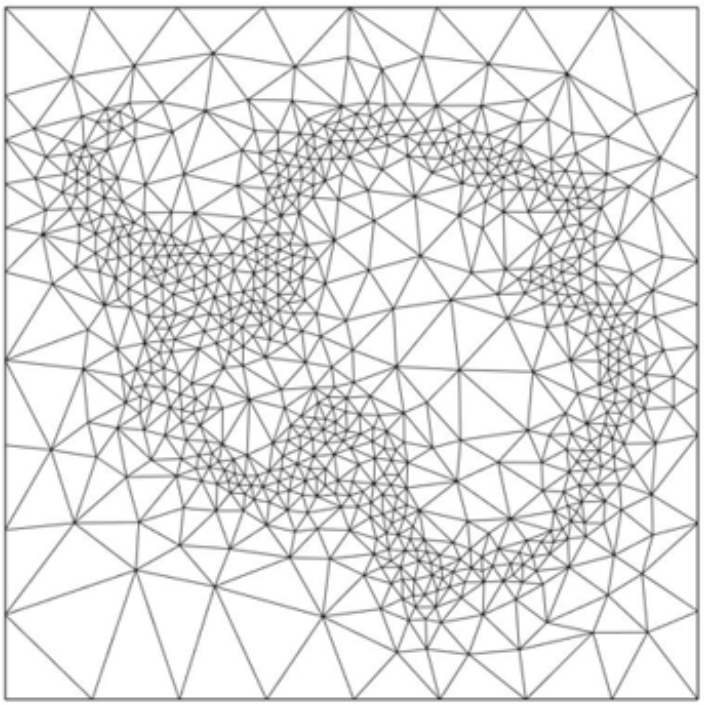


Figure 1: *An example of an irregular mesh of the Antarctic ice sheet. At the grounding line, there is a higher resolution.*

The model uses two approximations of the Navier-Stokes equations to calculate the depth-averaged ice velocities for each timestep. For grounded ice, the shallow ice approximation (SIA) is used [9]. At places where the traction at the base of the ice sheet is negligible, the shallow shelf approximation (SSA) is used [2, 3]. At these places, there is almost no friction at the base of the ice sheet, so the movement of the ice mainly consists of sliding. The SSA can also be used for floating ice, where the movement of the ice is caused by longitudinal stretching.
In the model, the SSA equations are solved using Succesive Overrelaxation (SOR). This needs to be done for each timestep. When calculating the velocities on timestep $\tau + 1$, the velocities from timestep τ can be

used as a first guess in the SOR method. In general, the velocities will not change considerably between two timesteps, so this first guess already gives a good approximation of the velocities on timestep $\tau + 1$. For this reason, an iterative solving method is faster than a direct solving method.
Despite this, solving the equations in the SSA with SOR takes a lot of time. In article [2] it is shown that the iterative method that is used for solving the equations in the SSA requires as much, or even more computation time than all the other model components combined. If this can be done more efficiently, the whole model will become faster.

The goal of this study is to investigate if there are ways to increase the convergence rate of the SOR method on an irregular mesh. When an improved scheme is used for solving the SSA equations, they can be solved more efficiently and therefore, they will make the whole model faster. Eventually, this can help to make better predictions of the response of ice sheets to the warming climate and can help to make predictions for sea level rise.

To get a better understanding of how the SSA equations can be solved on an irregular mesh, a description of the methods used in UFEMISM is given. The method that is used to discretize derivatives on an irregular mesh is explained in section 2. After that, some iterative methods that are used for solving the SSA equations are explained in section 3. In section 4, a short explanation of the SSA equations is given.
To investigate if there are ways to increase the convergence rate of the iteration scheme on an irregular mesh, a demo model is used. This demo model can solve differential equations on an irregular mesh with the same methods as in UFEMISM.
In the demo model, the Laplace equation is solved using different iteration schemes on an irregular mesh. This equation is a bit more easy than the equations in the SSA. Therefore, it is more easy to understand how different iteration methods work for this equation. The results of different methods that are tested on the Laplace equation are given in section 5. Eventually, these results can be tested on the SSA equations in UFEMISM. If they also improve the convergence rate of the SSA equations, they can be implemented in UFEMISM to reduce the computation time of the whole model.

2 Discretisation scheme on an irregular mesh.

To solve the equations of the SSA, which are described in section 4, some second partial derivatives of the velocity are needed. To find the numerical solution, those partial derivatives need to be discretized. On a square grid, there are several ways for approximating partial derivatives of a function f. Every derivative of a function f on vertex v_i can be approximated as a linear combination of the function-values on vertex v_i and its surrounding vertices. Because UFEMISM is running on a irregular triangular mesh, these approximations can not be used. In UFEMISM, they use a similar approach and show that the first- and second-order spatial derivatives of f on vertex v_i can also be approximated as linear combinations of the function-values on vertex v_i and its surrounding vertices. Here we are going into detail how this works. The whole method, and some exceptions, are also described in the original article about UFEMISM [2].

2.1 Approximating partial derivatives on a regular mesh

In figure 2, a regular mesh is shown. In a regular mesh, the horizontal distance Δx between the vertices is equal to the vertical distance Δy. There are several ways to approximate partial derivatives on a regular mesh. One of these approximations is shown here. Let f be a continuous function defined on every vertex of the mesh and $f(x, y)$ the value on vertex (x, y).
The second partial derivatives on a vertex with coordinates (x, y) can then be approximated as a linear combination of the values of f on the surrounding vertices. To see this, the approximation of the second partial derivative with respect to x is derived here. Let f be a differentiable function given on every vertex (x, y). Now, the first partial derivatives with respect to x can be approximated on the red points in figure 2. The red points p_1 and p_2 are added half way between vertices in such a way that p_1 has coordinates $(x + \frac{1}{2}\Delta x, y)$ and p_2 has coordinates $(x - \frac{1}{2}\Delta x, y)$. The distance between p_1 and p_2 is then also given by Δx. With this, the first partial derivatives with respect to x on the points p_1 and p_2 are given by:

$$\left.\frac{\partial}{\partial x} f\right|_{p_1} \approx \frac{f(x, y) - f(x - \Delta x, y)}{\Delta x}, \tag{2.1}$$

$$\left.\frac{\partial}{\partial x} f\right|_{p_2} \approx \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}. \tag{2.2}$$

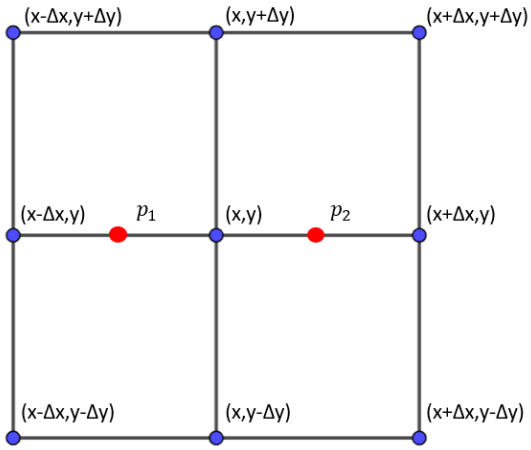


Figure 2: *Regular mesh*

These approximations can be used to approximate the second partial derivative with respect to x on vertex (x, y). Using the same approximation as above, the second partial derivative can be approximated as a linear combination of the approximated derivatives on the points p_1 and p_2:

$$\left.\frac{\partial^2}{\partial x^2} f\right|_{(x,y)} \approx \frac{\left.\frac{\partial}{\partial x} f\right|_{p_1} - \left.\frac{\partial}{\partial x} f\right|_{p_2}}{\Delta x}. \tag{2.3}$$

Now, equations (2.2) and (2.1) can be substituted into equation (2.3). This leads to the approximation of the second partial derivative with respect to x:

$$\left.\frac{\partial^2}{\partial x^2} f\right|_{(x,y)} \approx \frac{f(x + \Delta x, y) - 2f(x, y) + f(x - \Delta x, y)}{(\Delta x)^2}. \tag{2.4}$$

The approximation for the second partial derivative with respect to y can be derived in the same way and is given by:

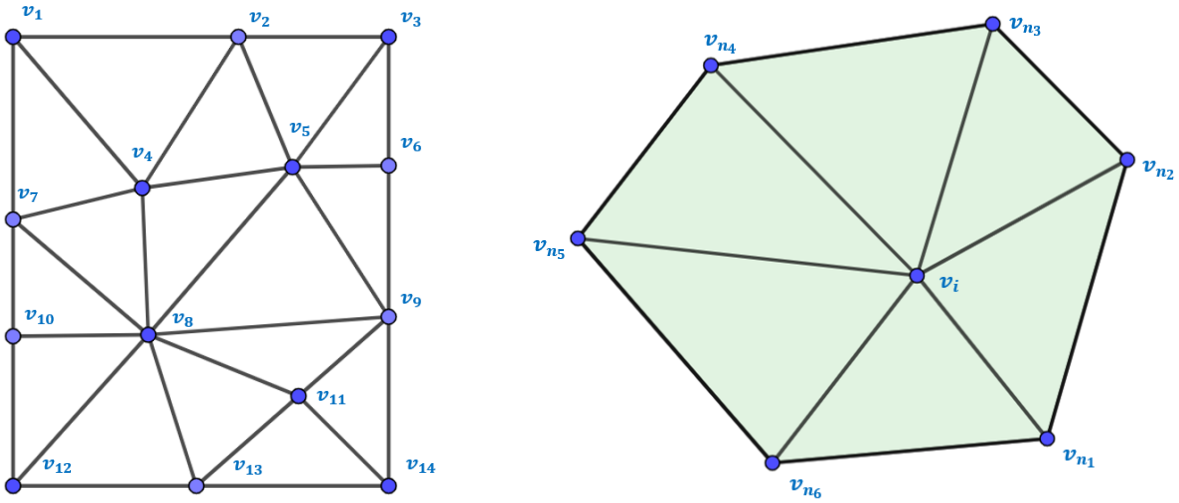
$$\left.\frac{\partial^2}{\partial y^2} f\right|_{(x,y)} \approx \frac{f(x, y + \Delta y) - 2f(x, y) + f(x, y - \Delta y)}{(\Delta y)^2}. \tag{2.5}$$

In this way, the partial derivatives in vertex (x, y) can be approximated as a linear combination of the surrounding vertices.

2.2 Approximations on an irregular mesh.

2.2.1 Notation

To get an approximation of the partial derivative on an irregular mesh, all vertices in the mesh are denoted with a unique number (the 'vertex-index'). In the mesh, vertex v_i is connected with some other vertices. If v_i and v_j are connected, they are called neighbours. If a vertex v_i has N neighbours, the neighbours of v_i are indicated with v_{n_t} with $t \in [1, N]$ (the 'neighbour-index'). The neighbours are numbered counterclockwise. To illustrate this, a mini-mesh is given in figure 3a. Here, vertex v_4 has five neighbours where $v_{n_1} = v_8$, $v_{n_2} = v_5$, $v_{n_3} = v_2$, $v_{n_4} = v_1$ and $v_{n_5} = v_7$.



(a) *A mini-mesh consisting of* 14 *vertices.* (b) *Vertex* v_i *with its surrounding vertices and triangles.*

Figure 3

Figure 3b gives an illustration of vertex v_i with coordinates (x_i, y_i) and its neighbouring vertices v_{n_1} to v_{n_6}. When talking about neighbours, an arbitrary neighbour of v_i is sometimes denoted with v_t in stead of v_{n_t}. From the context, it should be clear if the neighbour-index or the vertex-index is meant.
For the approximation, it sometimes is necessary to refer to the the next neighbour v_{t+1}. In general, we can refer to any other neighbour than v_t by v_{t+k} for $k \in \mathbb{N}$. Because there are only N neighbours, $t+k$ should be taken mod(N). In this way, a neighbour that is k steps away from neighbour v_t is denoted with $v_{(t+k) \bmod (N)}$. From now on, the notation $t + k^*$ will be used in stead of $(t + k) \bmod (n)$.
In figure 3b the triangles surrounding vertex v_i are shown in green. They are also numbered counterclockwise. For every neighbour v_t, there is a triangle between vertices v_i, v_t and v_{t+1^*}. If v_i is not at the boundary, the amount of neighbours is equal to the amount of triangles.

2.2.2 First partial derivative on an irregular mesh

Let $f : \mathbb{R}^2 \to \mathbb{R}^n$ be a function defined on every vertex $v_i = (x_i, y_i)$ of the mesh and let $f(v_i) = f_i$. Then, for every vertex v_i, a vector $\boldsymbol{v}_i$ is defined:

$$\boldsymbol{v}_i = [x_i, y_i, f_i] \in \mathbb{R}^3.$$

Now, for every vertex v_i with neighbours v_t and v_{t+1^*}, there is a triangle $\mathcal{T}_t \in \mathbb{R}^3$ which is spanned by $\boldsymbol{v}_i$, $\boldsymbol{v}_t$ and $\boldsymbol{v}_{t+1^*}$. On every triangle $\mathcal{T}_t$, the partial derivatives can be approximated. The derivative f_x^i on vertex v_i is then approximated by taking the average of the derivatives on the N surrounding triangles. In the example below, only the partial derivative f_x is derived. The approximation of f_y can be derived in the same way.

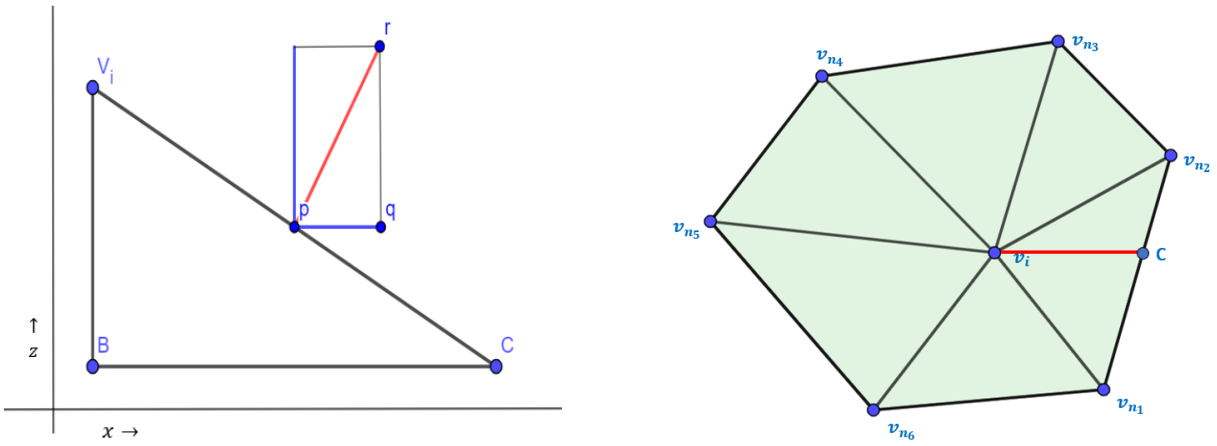
An approximation for f_x on triangle $\mathcal{T}_t$ can be made by looking at the slope of triangle $\mathcal{T}_t$. The slope of this triangle can be calculated using the upward normal vector $\boldsymbol{n}^t$ on that triangle, which can be given by the cross product of the two edges of the triangle. Since triangle $\mathcal{T}_t$ is spanned by $\boldsymbol{v}_i$, $\boldsymbol{v}_t$ and $\boldsymbol{v}_{t+1^*}$, this cross product is given by:

$$\boldsymbol{n}^t = (\boldsymbol{v}_t^* - \boldsymbol{v}_i) \times (\boldsymbol{v}_{t+1^*} - \boldsymbol{v}_i) = \begin{bmatrix} (y_{t^*} - y_i)(f_{t+1^*} - f_i) - (f_{t^*} - f_i)(y_{t+1^*} - y_i) \\ (f_{t^*} - f_i)(x_{t+1^*} - x_i) - (x_{t^*} - x_i)(f_{t+1^*} - f_i) \\ (x_{t^*} - x_i)(y_{t+1^*} - y_i) - (y_{t^*} - y_i)(x_{t+1^*} - x_i) \end{bmatrix}.$$

This can be rewritten as:

$$\boldsymbol{n}^t = \begin{bmatrix} f_i(y_{t+1^*} - y_{t_*}) + f_{t^*}(y_i - y_{t+1^*}) + f_{t+1^*}(y_{t^*} - y_i) \\ f_i(x_{t^*} - x_{t+1^*}) + f_{t^*}(x_{t+1^*} - x_i) + f_{t+1^*}(x_i - x_{t^*}) \\ (x_{t^*} - x_i)(y_{t+1^*} - y_i) - (y_{t^*} - y_i)(x_{t+1^*} - x_i) \end{bmatrix} = \begin{bmatrix} n_x^t \\ n_y^t \\ n_z^t \end{bmatrix}.$$

Now, the partial derivative on triangle $\mathcal{T}_t$ is given by $\frac{-n_x^t}{n_z^t}$. To see why this is true, an illustration is made in figure 4a. Because the partial derivative with respect to x is calculated, a cross section of triangle $\mathcal{T}_t$ in the xz plane is shown. The upper normal vector $\boldsymbol{n}^t$ is given in red. Since triangle V_iBC and pqr are similar, the slope can be calculated by $\frac{-n_x^t}{n_z^t}$. The notation $f_{x,tri}^t$ is used for the approximation of the partial derivative on triangle $\mathcal{T}_t$:



(a) *Cross section of triangle $\mathcal{T}_t$ in the xz plane. The normal vector is shown in red.*

(b) *The location of the cross section of figure 4a is given in red. Vertex C is also indicated here.*

Figure 4

$$f_{x,tri}^t = \frac{-n_x^t}{n_z^t} = \frac{-(f_i(y_{t+1^*} - y_{t^*}) + f_{t^*}(y_i - y_{t+1^*}) + f_{t+1^*}(y_{t^*} - y_i))}{(x_{t^*} - x_i)(y_{t+1^*} - y_i) - (y_{t^*} - y_i)(f_{t^*} - f_i)}.$$

Or, after rewriting the upper part:

$$f_{x,tri}^t = \frac{f_i(y_{t^*} - y_{t+1^*}) + f_{t^*}(y_{t+1^*} - y_i) + f_{t+1^*}(y_i - y_{t^*})}{(x_{t^*} - x_i)(y_{t+1^*} - y_i) - (y_{t^*} - y_i)(f_{t^*} - f_i)}. \tag{2.6}$$

In equation (2.6) the terms with x and y only depend on the properties of the mesh, so when the mesh is created, they can be calculated in advance. In [2], all those terms are isolated and collected in one function. This kind of functions are called neighbourfunctions. Neighbourfunctions are linear coefficients that can be used to express the partial derivative on a triangle or vertex as a linear combination of the function values on the neighbouring vertices. For every triangle in the mesh, the neighbourfunction $\boldsymbol{N}_{x,tri}^t$ is defined as:

$$\boldsymbol{N}_{x,tri}^t = \frac{1}{n_z^t}\left[(y_{t^*} - y_{t+1^*}), \quad (y_{t+1^*} - y_i), \quad (y_i - y_{t^*})\right]. \tag{2.7}$$

Let $\boldsymbol{F}_{tri} = [f_i, f_{t^*}, f_{t+1^*}]$. Now, equation (2.6) can also be given by taking the dot product of $\boldsymbol{F}_{tri}$ and $\boldsymbol{N}^t_{x,tri}$. So:

$$f^t_{x,tri} = \langle \boldsymbol{F}_{tri}, \boldsymbol{N}^t_{x,tri} \rangle. \tag{2.8}$$

In this way, the partial derivatives can be calculated on every triangle surrounding vertex v_i using the properties of the the vertices of that triangle and the values of f on those vertices. The approximation for the partial derivative on vertex v_i is now given by taking the average of the derivatives on all surrounding triangles $\mathcal{T}_t$. For this approximation, the notation f^i_x is used. For N surrounding triangles, this approximation is given by the equation below:

$$f^i_x = \frac{1}{N} \sum_{t=1}^{N} f^t_{x,tri}. \tag{2.9}$$

This can be written in another way with new neighbourfunctions. Let $N^t_{x,tri}(j)$ be the j'th column of $\boldsymbol{N}^t_{x,tri}$. Then equation (2.9) becomes:

$$\begin{aligned} f^i_x = \frac{1}{N} \sum_{t=1}^{N} f^t_{x,tri} &= \frac{1}{N} \sum_{t=1}^{N} \left[f_i N^t_{x,tri}(1) + f_{t^*} N^t_{x,tri}(2) + f_{t+1^*} N^t_{x,tri}(3) \right] \\ &= \frac{1}{N} \sum_{t=1}^{N} f_i N^t_{x,tri}(1) + \frac{1}{N} \sum_{t=1}^{N} f_{t^*} N^t_{x,tri}(2) + \frac{1}{N} \sum_{t=1}^{N} f_{t+1^*} N^t_{x,tri}(3). \end{aligned}$$

Because the sum is taken over all the triangles surrounding v_i and $t+1^*$ is taken $mod(N)$, it does not matter in which order the summation is done. Therefore,

$$\frac{1}{N} \sum_{t=1}^{N} f_{t+1^*} N^t_{x,tri}(3)) = \frac{1}{N} \sum_{t=1}^{N} f_{t^*} N^{t-1}_{x,tri}(3).$$

Applying this, The last two sums can be put together. For the left sum, only the function values on vertex v_i are needed, so the term f_i does not have to be inside the summation. This gives:

$$\begin{aligned} f^i_x &= \frac{1}{N} \sum_{t=1}^{N} f_i N^t_{x,tri}(1) + \frac{1}{N} \sum_{t=1}^{N} f_{t^*} N^t_{x,tri}(2) + \frac{1}{N} \sum_{t=1}^{N} f_{t^*} N^{t-1}_{x,tri}(3) \\ &= f^i \frac{1}{N} \sum_{t=1}^{N} N^t_{x,tri}(1) + \frac{1}{N} \sum_{t=1}^{N} f_{t^*} (N^t_{x,tri}(2) + N^{t-1}_{x,tri}(3). \end{aligned} \tag{2.10}$$

To make equation (2.10) clearer, new neighbourfunctions are introduced. The neighbourfunctions that are introduced below are N^i_x and $N^{i,t}_x$. For every v_i, there is a N^i_x and for every neighbour of v_i there is a $N^{i,t}_x$. Therefore, the amount of neighbourfunctions $N^{i,t}_x$ is equal to the amount of surrounding triangles. The neighbourfunctions are defined below:

$$N^i_x = \frac{1}{N} \sum_{t=1}^{N} N^t_{x,tri}(1) = \frac{1}{N} \sum_{t=1}^{N} \frac{1}{n^t_z} (y_{t^*} - y_{t+1^*}), \tag{2.11}$$

$$N^{i,t}_x = \frac{1}{N} (N^t_{x,tri}(2) + N^{t-1}_{x,tri}(3)) = \frac{1}{N} (\frac{1}{n^t_z} (y_{t+1^*} - y_i) + \frac{1}{n^t_z} (y_i - y_{t-1^*})). \tag{2.12}$$

Using the neighbourfuncions, the partial derivative with respect to x is given by:

$$f^i_x = f_i N^i_x + \sum_{t=1}^{N} f_{t^*} N^{i,t}_x. \tag{2.13}$$

Because $N^i_x, N^{i,t}_x \in \mathbb{R}$, equation (2.13) shows that the partial derivative f_i on v_i can be approximated as a linear combination of the function-values on v_i and its neighbours. In the same way, it can be derived that:

$$f^i_y = f_i N^i_y + \sum_{t=1}^{N} f_{t^*} N^{i,t}_y, \tag{2.14}$$

which is also a linear combination of the function-values on v_i and its neighbours.

2.2.3 Second partial derivative on an irregular mesh

To get the second partial derivatives on every vertex, a similar approach can be used. Now, the values of the first derivatives on every vertex are needed. But to determine the first derivative on every neighbour-vertex, also the neighbour-vertices of those are needed. So then the second derivative would become a linear combination of the values of f of the surrounding vertices and the vertices surrounding the surrounding vertex. The method described below will give an approximation of the second derivative on vertex v_i in such a way that it is linear combination of the values of f on the vertex v_i and its surrounding vertices.
To do this, some temporary vertices are added in the mesh. For every triangle surrounding vertex v_i, the vertex w_t is added at the geometric centre of that triangle. Together with vertex v_i, these temporary vertices define the subtriangles $\mathcal{T}'_s$. For example, subtriangle $\mathcal{T}'_2$ is defined by v_i, w_2 and w_3. If v_i is not at the boundary of the mesh and has N neighbours, there are also N triangles and also N subtriangles. Let s be the index of a vertex or triangle, then the notation $(s+k)^*$ is used again for $(s+k)(\text{mod}(N))$.

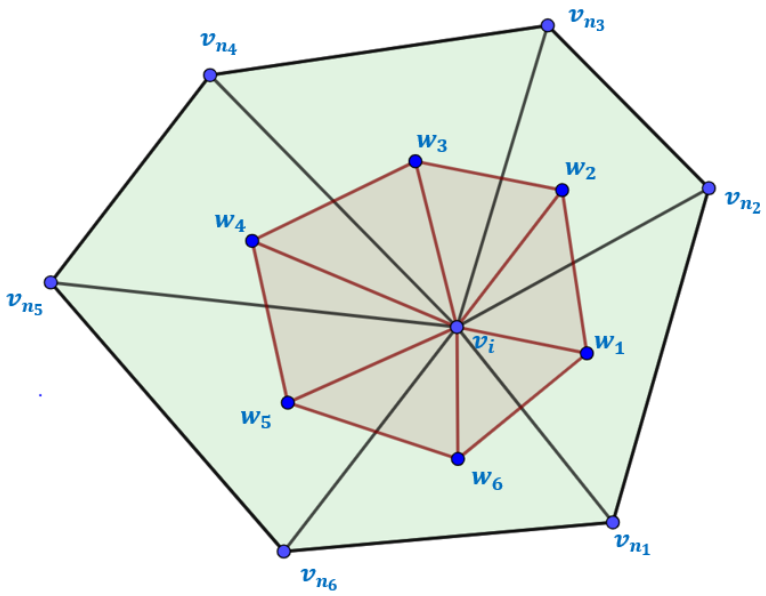


Figure 5: *This figure shows the temporary vertices w_t on every triangle. The temporary vertices define the subtriangles $\mathcal{T}'_s$, which are given in red.*

The partial derivative f_x on w_t can now be approximated with the slope on triangle $\mathcal{T}_t$. In equation (2.8), this is given by $f^t_{x,tri}$. As in 2.2, a vector $\boldsymbol{w}_t$ is created that uses the x and y coordinates of w_t:

$$\boldsymbol{w}_t = \left[\frac{x_i + x_{t^*} + x_{t+1^*}}{3}, \frac{y_i + y_{t^*} + y_{t+1^*}}{3}, f^t_{x,tri}\right]. \tag{2.15}$$

On vertex v_i, vector $\boldsymbol{w}_i$ is given by $\boldsymbol{w}_i = [x_i, y_i, f^i_x]$. Together, the vectors $\boldsymbol{w}_i$ and $\boldsymbol{w}_t$ with $t \in [1, N]$ define the subtriangles $\mathcal{T}'_s$ surrounding vertex v_i. The second partial derivative on every subtriangle $\mathcal{T}'_s$ is can now be approximated by taking the normal vector on the plane on from subtriangle $\mathcal{T}'_s$:

$$\boldsymbol{n}^s_x = (\boldsymbol{w}_s - \boldsymbol{w}_i) \times (\boldsymbol{w}_{s+1^*} - \boldsymbol{w}_i). \tag{2.16}$$

This cross product can be calculated and and the values of the first derivatives f^i_x, $f^{s^*}_{x,tri}$ and $f^{s+1^*}_{x,tri}$ can be

isolated. Then the normal vector on subtriangle $\mathcal{T}'_s$ is given by:

$$\boldsymbol{n}^s_x = \frac{1}{3}\begin{bmatrix} f^i_x(y_{s+2^*} - y_{s^*}) + fs^*_{x,tri}(2y_i - y_{s+1^*} - y_{s+2^*}) + f^{s+1^*}_{x,tri}(y_{s^*} + y_{s+1^*} - 2y_i) \\ f^i_x(x_{s*} - x_{s+2^*}) + f^{s^*}_{x,tri}(-2x_i + x_{s+1^*} + x_{s+2^*}) + f^{s+1^*}_{x,tri}(2x_i - x_{s+1^*} - x_{s^*}) \\ \frac{1}{3}\left((x_{s*} + x_{s+1^*} - 2x_i)(y_{s+1^*} + y_{s+2^*} - 2y_i) - (y_{s*} + y_{s+1^*} - 2y_i)(x_{s+1^*} + x_{s+2^*} - 2x_i)\right) \end{bmatrix}.$$

With this, the second order derivatives $f^s_{xx,sub}$ and $f^s_{xy,sub}$ on every subtriangle $\mathcal{T}'_s$ can be approximated by:

$$\begin{aligned} f^s_{xx,sub} &= \frac{-n^s_{x,x}}{n^s_{x,z}}, \\ f^s_{xy,sub} &= \frac{-n^s_{x,y}}{n^s_{x,z}}. \end{aligned} \tag{2.17}$$

This gives:

$$f^s_{xx,sub} = \frac{f^i_x(y_{s^*} - y_{s+2^*}) + f^{s^*}_{x,tri}(y_{s+1^*} + y_{s+2^*} - 2y_i) + f^{s+1^*}_{x,tri}(2y_i - y_{s^*} - y_{s+1^*})}{\frac{1}{3}\left((x_{s^*} + x_{s+1^*} - 2x_i)(y_{s+1^*} + y_{s+2^*} - 2y_i) - (y_{s*} + y_{s+1^*} - 2y_i)(x_{s+1^*} + x_{s+2^*} - 2x_i)\right)}. \tag{2.18}$$

In this equation, new neighbourfunctions are defined on the subtriangles $\mathcal{T}'_s$. These only depend on the subtriangles and can be calculated in advance:

$$\boldsymbol{N}^s_{x,sub} = \frac{1}{n^s_{x,z}}\left[(y_{s^*} - y_{s+2^*}),\qquad (y_{s+1^*} + y_{s+2^*} - 2y_i),\qquad (2y_i - y_{s^*} - y_{s+1^*})\right]. \tag{2.19}$$

Now, the second partial derivative on vertex on subtriangle $\mathcal{T}'_s$ is:

$$f^s_{xx,sub} = f^i_x(N^s_{x,sub}(1)) + f^{s^*}_{x,tri}(N^s_{x,sub}(2)) + f^{s+1^*}_{x,tri}(N^s_{x,sub}(3)).$$

Here, $N^s_{x,sub}(j)$ denotes column j of $\boldsymbol{N}^s_{x,sub}$. Now, the values for f^i_x, f_{s^*} and f_{s+1^*} can be filled in using equations (2.8) and (2.13) from section 2.2:

$$\begin{aligned} f^s_{xx,sub} = N^s_{x,sub}(1)(f_i N^i_x + \sum_{t=1}^{n} f_t N^{i,t}_x) + N^s_{x,sub}(2)(f_i N^s_{x,tri}(1) + f_s N^s_{x,tri}(2) + f_{(s+1)^*} N^s_{x,tri}(3)) \\ + N^s_{x,sub}(3)(f_i N^{(s+1)^*}_{x,tri}(1) + f_{(s+1)^*} N^{(s+1)^*}_{x,tri}(2) + f_{(s+2)^*} N^{(s+1)^*}_{x,tri}(3)). \end{aligned}$$

After rearranging the terms above, the partial derivative of on subtriangle $\mathcal{T}'_s$ is given by:

$$\begin{aligned} f^s_{xx,sub} = f_i\left[N^s_{x,sub}(1)N^i_x + N^s_{x,sub}(2)N^s_{x,tri}(1) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(1)\right] \\ + f_s\left[N^s_{x,sub}(2)N^s_{x,tri}(2)\right] \\ + f_{(s+1)^*}\left[N^s_{x,sub}(2)N^s_{x,tri}(3) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(2)\right] \\ + f_{(s+2)^*}\left[N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(3)\right] + N^s_{x,sub}(1)\sum_{t=1}^{n} f_t N^{i,t}_x. \end{aligned} \tag{2.20}$$

Now the second partial derivative can be found on every subtriangle. The second partial derivative on every vertex v^i is again approximated by taking the average of all the derivatives on the N neighbouring subtriangles:

$$f^i_{xx} = \frac{1}{N}\sum_{s=1}^{N} f^s_{xx,sub}.$$

Using the approximation from equation (2.20), this becomes:

$$f^i_{xx} = \frac{1}{N}\sum_{s=1}^{N} f_i \left[N^s_{x,sub}(1)N^i_x + N^s_{x,sub}(2)N^s_{x,tri}(1) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(1)\right] + \frac{1}{N}\sum_{s=1}^{N} f_s \left[N^s_{x,sub}(2)N^s_{x,tri}(2)\right] + \frac{1}{N}\sum_{s=1}^{N} f_{(s+1)^*} \left[N^s_{x,sub}(2)N^s_{x,tri}(3) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(2)\right] + \frac{1}{N}\sum_{s=1}^{N} f_{(s+2)^*} \left[N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(3)\right] + \frac{1}{N}\sum_{s=1}^{N} \left(N^s_{x,sub}(1)\sum_{t=1}^{n} f_t N^{i,t}_x\right).$$

In the last term of this equation, the indices of the inner sum do not depend on the indices of the outer sum, so they can be written as two sums. To make the equations more compact, indices s and t can be switched. With this, the last term can be given by:

$$\frac{1}{N}\sum_{s=1}^{N} \left(N^s_{x,sub}(1)\sum_{t=1}^{N} f^t N^{i,t}_x\right) = \frac{1}{N}\sum_{s=1}^{N} f_s N^{i,s}_x \sum_{t=1}^{N} N^t_{x,sub}.$$

Because the sums are taken over all N triangles, the order of summation does not matter, so the sum terms with $s+1$ and $s+2$ can both start with s. So in the last sum, $(s+1)^* = s$ and $(s+2)^* = s$ will be substituted. This gives:

$$f^i_{xx} = \frac{1}{N}\sum_{s=1}^{N} f_i \left[N^s_{x,sub}(1)N^i_x + N^s_{x,sub}(2)N^s_{x,tri}(1) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(1)\right] + \frac{1}{N}\sum_{s=1}^{N} f_s \left[N^s_{x,sub}(2)N^s_{x,tri}(2)\right] + \frac{1}{N}\sum_{s=1}^{N} + f_s \left[N^{(s-1)^*}_{x,sub}(2)N^{(s-1)^*}_{x,tri}(3) + N^{(s-1)^*}_{x,sub}(3)N^s_{x,tri}(2) + N^{(s-2)^*}_{x,sub}(3)N^{(s-1)^*}_{x,tri}(3)\right] + \frac{1}{N}\sum_{s=1}^{N} f_s N^{i,s}_x \sum_{t=1}^{N} N^t_{x,sub}.$$

Now, this can be rearranged to get:

$$f^i_{xx} = \frac{1}{N}\sum_{s=1}^{N} f_i \left[N^s_{x,sub}(1)N^i_x + N^s_{x,sub}(2)N^s_{x,tri}(1) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(1)\right] + \frac{1}{N}\sum_{s=1}^{N} f_s \left[N^s_{x,tri}(2)\left(N^s_{x,sub}(2) + N^{(s-1)^*}_{x,sub}(3)\right) + N^{(s-1)^*}_{x,tri}(3)\left(N^{(s-1)^*}_{x,sub}(2) + N^{(s-2)^*}_{x,sub}(3)\right) + N^{i,s}_x \sum_{t=1}^{n} N^t_{x,sub}(1)\right].$$

Again, there are two terms that only depend on properties of the mesh. These terms can be collected in two new neighbour-functions, which are defined below:

$$N^i_{xx} = \frac{1}{N}\sum_{s=1}^{N} \left[N^s_{x,sub}(1)N^i_x + N^s_{x,sub}(2)N^s_{x,tri}(1) + N^s_{x,sub}(3)N^{(s+1)^*}_{x,tri}(1)\right],$$

$$N^{i,s}_{xx} = \frac{1}{N}\left[N^s_{x,tri}(2)\left(N^s_{x,sub}(2) + N^{(s-1)^*}_{x,sub}(3)\right) + N^{(s-1)^*}_{x,tri}(3)\left(N^{(s-1)^*}_{x,sub}(2) + N^{(s-2)^*}_{x,sub}(3)\right) + N^{i,s}_x \sum_{t=1}^{n} N^t_{x,sub}(1)\right].$$

With this neighbour functions, the second derivative partial derivative with respect to x can be approximated as a linear combination of the function-values vertex v_i and its neighbours v_s:

$$f^i_{xx} = f^i N^i_{xx} + \sum_{s=1}^{n} f^s N^{i,s}_{xx}. \tag{2.21}$$

In the same way, it can be derived that f^i_{yy} can be approximated by:

$$f^i_{yy} = f^i N^i_{yy} + \sum_{s=1}^{n} f^s N^{i,s}_{yy}. \tag{2.22}$$

Since $N^i_{xx}, N^i_{yy}, N^{i,s}_{xx}, N^{i,s}_{yy} \in \mathbb{R}$, the partial derivatives of f on vertex v_i can be approximated a linear combination of the function values f_i on v_i and f_s on its neighbouring vertices.

3 Iteration methods

In UFEMISM, successive over relaxation (SOR) is used to solve the equations in the SSA on an irregular mesh.
All methods described in this section can be used to find the solution $\boldsymbol{x}$ of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ with $A \in \mathbb{R}^{n\times n}$ and $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$. To find the solution, a first guess $\boldsymbol{x}^0$ of the solution has to be made. Then, a better approximation of the solution can be given as a function of the first guess. In general, the next estimate $\boldsymbol{x}^{k+1}$ is given as a function of the previous estimate $\boldsymbol{x}^k$. This iterative scheme converges to the solution of the linear system. The iterative methods that are described in this section are all stationary iterative methods. Stationary iterative methods can be written as

$$\boldsymbol{x}^{k+1} = G\boldsymbol{x}^k + \boldsymbol{h},$$

with the iteration matrix $G \in \mathbb{R}^{n\times n}$ and vector $\boldsymbol{h} \in \mathbb{R}^n$. Here, the matrix G and vector $\boldsymbol{h}$ do not depend on the iteration step k.
In this section, three iterative methods are described.

3.1 Jacobi method

The first method is the Jacobi method. The linear system $A\boldsymbol{x} = \boldsymbol{b}$, describes n equations. Every equation of the system is given by:

$$\sum_{j=1}^{n} a_{i,j}x_j = b_i.$$

After rewriting each equation, every component of $\boldsymbol{x}$ can be written as a linear combination of the other components:

$$x_i = \frac{b_i - \sum_{j\neq i}^{n} a_{i,j}x_j}{a_{i,i}}.$$

The Jacobi method uses this equation to find the next iteration step. For every component x_i of vector $\boldsymbol{x}$, x_i^{k+1} can be calculated using the equation:

$$x_i^{k+1} = \frac{b_i - \sum_{j\neq i}^{n} a_{i,j}x_j^k}{a_{i,i}}. \tag{3.1}$$

When $k \to \infty$, this converges to the solution of the linear system.

3.2 Gauss-Seidel method

The second method is the Gauss-Seidel method. This method is based on the Jacobi method. However, the difference is that the Gauss-Seidel uses the updated values of $\boldsymbol{x}$ as soon as they are available whereas the Jacobi method only uses the values from the previous iteration step. Therefore, it is important to calculate the components of $\boldsymbol{x}$ one by one. First, x_1^{k+1} is calculated using equation 3.1. Then, this value can already be used in the calculation of x_2^{k+1}:

$$x_2^{k+1} = \frac{b_2 - \left(a_{2,1}x_1^{k+1} + \sum_{j=3}^{n} a_{2,j}x_j^k\right)}{a_{2,2}}.$$

Because x_1^{k+1} is calculated first, this component already has an updated version. The Gauss-seidel method uses this updated version of x_1 to calculate the next iteration step. In general, the Gauss-Seidel method uses the equation below to determine the next iteration step for $\boldsymbol{x}$:

$$x_i^{k+1} = \frac{b_i - \sum_{j<i} a_{i,j}x_j^{k+1} - \sum_{j>i} a_{i,j}x_j^k}{a_{i,i}}. \tag{3.2}$$

In this step, the components x_j with $j < i$ have already been updated, so for these components, the updated version can be used to determine the next iteration of x_i.

3.3 Successive Overrelaxation (SOR)

The Successive Overrelaxation method is based on the Gauss-Seidel method. First, it calculates the next iteration step $\bar{x}^{k+1}$ with the Gauss-Seidel method. Then it checks the difference between x^k and $\bar{x}^{k+1}$. This value is multiplied with an $\omega \in \mathbb{R}$ and added to the previous iteration step:

$$x_i^{k+1} = x_i^k + \omega \left(\bar{x}_i^{k+1} - x_i^k \right). \tag{3.3}$$

When $\omega = 1$, this method is the same as the Gauss-Seidel method. For a larger ω, the difference between x_i^k and x_i^{k+1} will also be larger. In [1] it is stated that the method can only converge if $\omega \in [0, 2]$. However, convergence is not guaranteed for every $\omega \in [0, 2]$ since it depends on the linear system that is solved. In most books, but also in UFEMISM, the SOR method is rewritten as:

$$x_i^{k+1} = \omega \bar{x}_i^{k+1} + (1 - \omega) x_i^k. \tag{3.4}$$

This method is used to solve the SSA equation in UFEMISM. The solution of the SSA gives the velocity of the ice on every vertex, so then $\boldsymbol{x}$ denotes a vector where x_i is the velocity on vertex v_i. In section 4 this will be explained in more detail.

3.4 Convergence of iterative methods

The iteration methods described in the sections above give a solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$. Here, a way to determine the convergence of those iteration methods is shown [5]. Each iterative method can be written as a matrix equation

$$\boldsymbol{x}^{k+1} = G\boldsymbol{x}^k + \boldsymbol{h},$$

with the iteration matrix $G \in \mathbb{R}^{n \times n}$ and vector $\boldsymbol{h} \in \mathbb{R}^n$. The solution $\boldsymbol{x}^*$ of the linear system should satisfy $\boldsymbol{x}^* = G\boldsymbol{x}^* + \boldsymbol{h}$. The matrix G depends on the linear system and the iterative method that is used. In [6], [1] and [5], matrix G is given for the three iteration methods that are described in this section. The theorem below shows convergence of such a method and is written with the help of [5]. For this theorem, recall that for a matrix $G \in \mathbb{R}^{n \times n}$ with eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$, the spectral radius is defined by: $\rho(G) = \max(|\lambda_1|, |\lambda_2|, ..., |\lambda_n|)$.

Theorem 3.4.1. *Let $\boldsymbol{x}^{k+1} = G\boldsymbol{x}^k + \boldsymbol{h}$ be an iterative method for solving the linear system $A\boldsymbol{x} = \boldsymbol{b}$ with solution $\boldsymbol{x}^*$ and let $\boldsymbol{e}^k = \boldsymbol{x}^k - \boldsymbol{x}^*$ be the error on iteration step k. Assume G has n different eigenvalues λ_i with $i \in [1, n]$ with n linearly independent inventors. Then $\lim_{k \to \infty} ||\boldsymbol{e}^k|| = 0$ if and only if $\rho(G) < 1$.*

Proof. The error in iteration step $k + 1$ is given by $\boldsymbol{e}^{k+1} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^*$. Filling in the approximation for $\boldsymbol{x}^{k+1}$ and the true solution $\boldsymbol{x}^*$ gives:

$$\boldsymbol{e}^{k+1} = G\boldsymbol{x}^k + \boldsymbol{h} - (G\boldsymbol{x}^* + \boldsymbol{h}).$$

This can be rewritten in such a way that:

$$\boldsymbol{e}^{k+1} = G\left(\boldsymbol{x}^k - \boldsymbol{x}^* \right),$$

where $\boldsymbol{x}^k - \boldsymbol{x}^* = \boldsymbol{e}^k$, so the error in iteration step $k + 1$ can be given as a function of the error on iteration step k:

$$\boldsymbol{e^{k+1}} = G\boldsymbol{e}^k, \tag{3.5}$$

Now, let $\boldsymbol{e}^0 = \boldsymbol{x}^0 - \boldsymbol{x}^*$ be the initial error. Then equation (3.5) shows that $\boldsymbol{e}^1 = G\boldsymbol{e}^0$. Then $\boldsymbol{e}^2 = G\boldsymbol{e}^1 = G^2\boldsymbol{e}^0$. So in general, it can be proven by induction that the error in iteration step k can be given as a function of the initial error:

$$\boldsymbol{e}^k = G^k\boldsymbol{e}^0. \tag{3.6}$$

To get an estimation of the error on iteration step k, G^k needs to be calculated. Therefore, the eigenvectors and eigenvalues of G are calculated. Because the eigenvectors of G are linearly independent, they form a

basis of $\mathbb{C}^n$, so every vector in $\mathbb{C}^n$ can be written as a linear combination of the eigenvectors of G. This can also be done with the initial error $\boldsymbol{e}^0$:

$$\boldsymbol{e}^0 = c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + ... + c_n\boldsymbol{v}_n.$$

Applying G on every eigenvector v_i is the same as multiplying it by its eigenvalue λ_i, so applying G to $\boldsymbol{e}^0$ gives:

$$G\boldsymbol{e}^0 = \lambda_1 c_1\boldsymbol{v}_1 + \lambda_2 c_2\boldsymbol{v}_2 + ... + \lambda_n c_n\boldsymbol{v}_n.$$

Every iteration, the error is multiplied with matrix G, so after k iterations, the error is given by:

$$\boldsymbol{e}^k = G^k\boldsymbol{e}^0 = \lambda_1^k c_1\boldsymbol{v}_1 + \lambda_2^k c_2\boldsymbol{v}_2 + ... + \lambda_n^k c_n\boldsymbol{v}_n = \sum_{i=1}^{n} \lambda_i^k c_i\boldsymbol{v}_i.$$

Now, the norm of $\boldsymbol{e}^k$ can be estimated using the triangle inequality. This gives:

$$||\boldsymbol{e}^k|| \leq |\lambda_1|^k|c_1|||\boldsymbol{v}_1|| + |\lambda_2|^k|c_2|||\boldsymbol{v}_2|| + ... + |\lambda_n|^k|c_n|||\boldsymbol{v}_n||. \tag{3.7}$$

The value of λ_i^k will approach zero if k goes to infinity in and only if λ_i has the property that $|\lambda_i| < 1$. Therefore, in this case, all the terms $|\lambda_i|^k|c_i|||\boldsymbol{v}_i||$ in equation (3.7) will approach zero if iteration step k approaches infinity. So therefore $\lim_{k\to\infty} ||\boldsymbol{e}^k|| = 0$ which proves the theorem. □

This theorem shows that an iterative method converges if the iteration matrix G has n linearly independent eigenvectors $\boldsymbol{v}_i$ with corresponding eigenvalues λ_i which have the property that $|\lambda_i| < 1$. If this is the case, convergence will occur for any initial guess $\boldsymbol{x}^0$.
When working with iterative methods, in most cases, the exact solution $\boldsymbol{x}^*$ of the system is unknown. Therefore, it makes sense to have a look at the convergence ratio, which is given by:

$$\frac{||\boldsymbol{e}^{k+1}||}{||\boldsymbol{e}^k||}.$$

There is another relation between the spectral radius of a iteration matrix G and the convergence rate. This is given in the next theorem.

Theorem 3.4.2. *Let $\lambda_q = \rho(G)$ and assume that $|\lambda_q| > |\lambda_i|$ for all $i \neq q$. Then, $\lim_{k\to\infty} \frac{||\boldsymbol{e}^{k+1}||}{||\boldsymbol{e}^k||} = \lambda_q$.*

Proof. In the proof of theorem 3.4.1, the error on iteration k can be estimated by:

$$\boldsymbol{e}^k = G^k\boldsymbol{e}^0 = \lambda_1^k c_1\boldsymbol{v}_1 + \lambda_2^k c_2\boldsymbol{v}_2 + ... + \lambda_n^k c_n\boldsymbol{v}_n.$$

Now, the convergence ratio can be given by:

$$\frac{||\boldsymbol{e}^{k+1}||}{||\boldsymbol{e}^k||} = \frac{||\lambda_1^{k+1} c_1\boldsymbol{v}_1 + \lambda_2^{k+1} c_2\boldsymbol{v}_2 + ... + \lambda_n^{k+1} c_n\boldsymbol{v}_n||}{||\lambda_1^k c_1\boldsymbol{v}_1 + \lambda_2^k c_2\boldsymbol{v}_2 + ... + \lambda_n^k c_n\boldsymbol{v}_n||}. \tag{3.8}$$

Without loss of generality, assume $\lambda_1 = \lambda_q$. The numerator and denominator can be multiplied by $|\lambda_1^k|^{-1}$. This gives:

$$\frac{||\boldsymbol{e}^{k+1}||}{||\boldsymbol{e}^k||} = \frac{|\lambda_1^k|^{-1}||\lambda_1^{k+1} c_1\boldsymbol{v}_1 + \lambda_2^{k+1} c_2\boldsymbol{v}_2 + ... + \lambda_n^{k+1} c_n\boldsymbol{v}_n||}{|\lambda_1^k|^{-1}||\lambda_1^k c_1\boldsymbol{v}_1 + \lambda_2^k c_2\boldsymbol{v}_2 + ... + \lambda_n^k c_n\boldsymbol{v}_n||}.$$

This can be rewritten:

$$\frac{||\boldsymbol{e}^{k+1}||}{||\boldsymbol{e}^k||} = \frac{\left|\left|\lambda_1 c_1\boldsymbol{v}_1 + \lambda_2(\frac{\lambda_2}{\lambda_1})^k c_2\boldsymbol{v}_2 + ... + \lambda_n(\frac{\lambda_n}{\lambda_1})^k c_n\boldsymbol{v}_n\right|\right|}{\left|\left|c_1\boldsymbol{v}_1 + (\frac{\lambda_2}{\lambda_1})^k c_2\boldsymbol{v}_2 + ... + (\frac{\lambda_n}{\lambda_1})^k c_n\boldsymbol{v}_n\right|\right|}. \tag{3.9}$$

Because it is assumed that λ_1 has the property that $|\lambda_1| > |\lambda_i|$ for all i, the ratio $\frac{|\lambda_i|}{|\lambda_1|} < 1$ for all i with $i \neq 1$. So then:

$$\lim_{k\to\infty} \left(\frac{|\lambda_i|}{|\lambda_1|}\right)^k = 0.$$

So then, taking the limit of equation (3.10) gives:

$$\lim_{k\to\infty} \frac{||\boldsymbol{e}^{k+1}||}{||\boldsymbol{e}^k||} = \frac{||\lambda_1 c_1 \boldsymbol{v}_1||}{||c_1\boldsymbol{v}_1||} = \frac{|\lambda_1|||c_1\boldsymbol{v}_1||}{||c_1\boldsymbol{v}_1||} = |\lambda_1|. \tag{3.10}$$

Since the assumption was that $\lambda_1 = \lambda_q$, this proves the theorem. □

So there is another relation between the spectral radius of the iteration matrix G and the convergence rate. When the spectral radius of an iteration is small, the ratio between $||\boldsymbol{e}^{k+1}||$ and $||\boldsymbol{e}^k||$ is also small. So an iterative method with a smaller spectral radius will also converge faster. This knowledge about convergence can be used in the model to check whether, and how fast a method converges. But this can only be done if the corresponding matrix can be calculated easily.

4 The Shallow Shelf Approximation (SSA)

The Shallow Shelf Approximation is one of the velocity components in UFEMISM that calculates the velocity of the ice in every vertex. In this section, a basic description of the physics behind the SSA is given. A more detailed description can be found in article [3], [7] and [11].
The Shallow Shelf Approximation has two main assumptions. First of all, it is assumed that there is almost no friction at the base, so the movement of the ice is mainly caused by stretching and not by deformation. Secondly, the ice thickness is assumed to be really small compared to the horizontal span of the ice sheet. Because the SSA calculates the sliding velocities, it is reasonable to assume that the velocities do not change in vertical direction. All the variables that are used in the SSA are vertically integrated. Therefore, only the horizontal components of the velocity are calculated.
The SSA is the simplest form of a membrane stress balance that can be derived from the Stokes equations [3]. The full derivation will not be given here, but the equations are explained for a better understanding of the equations that are used. In this document, the same notation as in article [2] is used as much as possible because the code UFEMISM also uses this notation.
In article [3], the SSA is given as the following pair of stress balance equations:

$$\frac{\partial T_{i1}}{\partial x_1} + \frac{\partial T_{i2}}{\partial x_2} + \tau_{b,i} = \rho g H \frac{\partial h}{\partial x_i}. \tag{4.1}$$

The first two terms in this equation are a measure for the membrane stresses held by viscous deformation. The driving stresses are given by $\rho g h \frac{\partial h}{\partial x_i}$ and $\tau_{b,i}$ is the stress held at the base by till strength. Also, ρ is the density of the ice, g the acceleration of gravity, H the thickness of the ice and h the ice surface elevation. The vertically integrated stress tensor is given by T. Every component of this tensor is given by T_{ij}. In article [3], every component of the stress tensor T_{ij} is calculated as a function of the strain rates and the viscosity. The strain rates can be calculated as a function of the velocity.
Let $\boldsymbol{v}$=(u, v) be the vertically integrated velocity of the ice and $\bar{v}$ the viscosity. Because this notation is used, the first and second component of $\tau_{b,i}$ are given by $\tau_{b,u}$ and $\tau_{b,v}$. Then, in [2], equation (4.1) is given by:

$$\frac{\partial}{\partial x}\left[2\bar{v}H\left(2\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right] + \frac{\partial}{\partial y}\left[\bar{v}H\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y}\right)\right] + \tau_{b,u} = \rho g H \frac{\partial h}{\partial x}, \tag{4.2}$$

$$\frac{\partial}{\partial x}\left[\bar{v}H\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \frac{\partial}{\partial y}\left[2\bar{v}H\left(\frac{\partial u}{\partial x} + 2\frac{\partial v}{\partial y}\right)\right] + \tau_{b,v} = \rho g H \frac{\partial h}{\partial y}. \tag{4.3}$$

In the article of Bueler and Brown [3], the basal shear stress is estimated in the same way as in Schoof [11] and is given by:

$$\tau_{b,u} = -\tau_c \frac{u}{\sqrt{\delta^2 + u^2 + v^2}}, \tag{4.4}$$

$$\tau_{b,v} = -\tau_c \frac{v}{\sqrt{\delta^2 + u^2 + v^2}}. \tag{4.5}$$

Here, τ_c represents the yield stress of the material at the base of the ice sheet and is given by $\tau_c = c_0 + tan(\phi)(\rho g H - p_w)$. The cohesion c_0 is assumed to be equal to zero. The till friction angle is given by ϕ and $(\rho g H - p_w)$ is the effective pressure calculated using the pressure of the ice sheet and the pore water pressure p_w. The δ is added by Bueler and Brown [3] to avoid dividing by zero if the velocities are zero. Now, let $||\boldsymbol{v}|| = \sqrt{\delta^2 + u^2 + v^2}$. After substituting $\tau_{b,u} = -\tau_c \frac{u}{||\boldsymbol{v}||}$ and $\tau_{b,v} = -\tau_c \frac{v}{||\boldsymbol{v}||}$ in equations (4.2) and (4.3), those equation can be written as:

$$\frac{\partial}{\partial x}\left[2\bar{v}H\left(2\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right] + \frac{\partial}{\partial y}\left[\bar{v}H\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] - \tau_c \frac{u}{|\boldsymbol{v}|} = \rho g H \frac{\partial h}{\partial x}, \tag{4.6}$$

$$\frac{\partial}{\partial x}\left[\bar{v}H\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \frac{\partial}{\partial y}\left[2\bar{v}H\left(\frac{\partial u}{\partial x} + 2\frac{\partial v}{\partial y}\right)\right] - \tau_c \frac{v}{|\boldsymbol{v}|} = \rho g H \frac{\partial h}{\partial y}. \tag{4.7}$$

To make this equation and the following equations more readable, the notation u_x is used for the partial derivative $\frac{\partial u}{\partial x}$ and u_y for the partial derivative $\frac{\partial u}{\partial y}$. In [2], The vertically averaged viscosity for the ice is given

as a function of the ice velocity:

$$\bar{v} = \frac{1}{2}\int_b^h A(T^*)^{\frac{-1}{n}} dz \left[u_x^2 + v_y^2 + u_x v_y + \frac{1}{4}(u_y + v_x)^2 + \left(\frac{\epsilon^2}{L_v^2}\right)\right]^{\frac{1-n}{2n}}. \tag{4.8}$$

Here, $A(T^*)^{\frac{-1}{n}}$ is the ice hardness in $pa/s^{\frac{1}{3}}$ and n is the Glen exponent in ice flow law and is equal to three. The term $\left(\frac{\epsilon^2}{L_v^2}\right)$ is introduced in Bueler and Brown [3] to avoid that the viscosity equals zero if the velocity is equal to zero.
In UFEMISM, the two equations from the SSA are simplified a bit more. Here, this is only shown for the first part. The second part can be derived in the same way. The derivatives in equation (4.6) can be calculated using the product rule. Also, according to [2], $\frac{\partial}{\partial x}\bar{v}H << \bar{v}H$, so this terms can be ignored. This gives:

$$2\bar{v}H\left(2u_{xx} + v_{yx}\right) + \bar{v}H\left(u_{yy} + v_{xy}\right) - \tau_c \frac{u}{|\boldsymbol{u}|} = \rho g H \frac{\partial h}{\partial x}. \tag{4.9}$$

Because of continuity, v_{xy} is equal to v_{yx}. Let $\tau_b = \frac{\tau_c}{|\boldsymbol{u}|}$. After rearranging the terms above the first part of the SSA becomes:

$$4u_{xx} + u_{yy} + 3v_{xy} - \tau_b \frac{u}{\bar{v}H} = \frac{\rho g h_x}{\bar{v}}. \tag{4.10}$$

Using the same approximation as above, the second part of the SSA becomes:

$$4v_{yy} + v_{xx} + 3u_{xy} - \tau_b \frac{v}{\bar{v}H} = \frac{\rho g h_y}{\bar{v}}. \tag{4.11}$$

Now, the SSA is given by equation (4.10), (4.11) and (4.8). With these equations, the velocities on every vertex on the mesh can be calculated for each timestep.
With the viscosity equation filled in into the other two, this would be two coupled elliptic differential equations, which are hard to solve numerically. In UFEMISM, an approach that is also used in Bueler and Brown [3] is used. Two nested iterative loops are used to solve the equations on every time step. First, an outer loop calculates the viscosity using equation (4.8). To do this, the velocities of the previous time step are used. This gives a first approximation of the viscosity. After this, the velocities are calculated using an iteration scheme in the inner loop. When this iteration scheme is completed, the new velocities are found and can be used in the outer loop again to calculate a new viscosity. In the model, this process is repeated several times until the approximation is good enough.
In the inner loop, the velocities are calculated based on the viscosity from the outer loop. To do this, the two differential equations (4.10) and (4.11) are solved numerically using Successive Overrelaxation. The derivatives of u and v are replaced by the approximations from section 2, equation (2.21). This gives:

$$u_{xx}^i = u_i N_{xx}^i + \sum_{s=1}^{n} u_s N_{xx}^{i,s},$$

$$u_{yy}^i = u_i N_{yy}^i + \sum_{s=1}^{n} u_s N_{yy}^{i,s}.$$

In the term with the basal shear stress, v is replaced by u_i because the x-component of the velocity on vertex v_i is needed to calculate the basal shear stress in direction x. With this, equation (4.10) becomes:

$$4\left(u_i N_{xx}^i + \sum_{s=1}^{n} u_s N_{xx}^{i,s}\right) + \left(u_i N_{yy}^i + \sum_{s=1}^{n} u_s N_{yy}^{i,s}\right) + 3v_{xy} - \frac{\tau_b u_i}{\bar{v}H} = \frac{\rho g h_x}{\bar{v}}. \tag{4.12}$$

To be able to use the SOR method, the goal is to write the velocity on vertex v_i as a linear combination of the velocities on the other vertices. To do this, u_i is isolated. This gives:

$$u_i\left(4N_{xx}^i + N_{yy}^i - \frac{\tau_b}{\bar{v}H}\right) + \sum_{s=1}^{n} u_s\left(4N_{xx}^{i,s} + N_{yy}^{i,s}\right) + 3v_{xy} = \frac{\rho g h_x}{\bar{v}}. \tag{4.13}$$

The term $\left(4N^i_{xx} + N^i_{yy} - \frac{\tau_b}{\bar{v}H}\right)$ only consists of variables that can be calculated in the outer loop. The neighbour functions only depend on properties of the mesh, so can be calculated before iterations start. The viscosity and τ_b are calculated in the outer loop based on the velocities of the previous time step. Therefore, define $e^i_u = \left(4N^i_{xx} + N^i_{yy} - \frac{\tau_b}{\bar{v}H}\right)$. This term can be calculated before the iteration process in the inner loop starts. With this, the first component of the velocity vector in vertex v_i becomes:

$$u_i = \frac{1}{e^i_u}\left(-\sum_{s=1}^{n} u_s\left(4N^{i,s}_{xx} + N^{i,s}_{yy}\right) - 3v_{xy} + \frac{\rho g h_x}{\bar{v}}\right). \tag{4.14}$$

Using the same method, the second component of the velocity vector in vertex v_i becomes:

$$v_i = \frac{1}{e^i_v}\left(-\sum_{s=1}^{n} v_s\left(4N^{i,s}_{yy} + N^{i,s}_{xx}\right) - 3v_{xy} + \frac{\rho g h_y}{\bar{v}}\right). \tag{4.15}$$

In the inner loop of the code that solves the SSA, equations (4.14) and (4.15) are solved using SOR. In section 3, this method is given by:

$$u^{k+1}_i = (1-\omega)u^k_i + \omega u^{k+1}_{i,gs}.$$

Here, u^{k+1}_i is the new velocity on iteration step $k+1$, $u^{t+1}_{i,gs}$ is the next iteration step based on Gauss-Seidel method, u^i_k is the velocity on iteration k and ω is the extrapolation factor of the SOR-method.
The next iteration step on vertex v_i based on Gauss-Seidel is given by:

$$u^{k+1}_{i,gs} = \frac{1}{e^i_u}\left(-\sum_{s=1}^{n} u_s\left(4N^{i,s}_{xx} + N^{i,s}_{yy}\right) - 3v_{xy} + \frac{\rho g h_x}{\bar{v}}\right).$$

In the code of UFEMISM, all the vertices and their velocities are stored in a variable. When a velocity at a neihbour vertex is needed, the model will read it from that variable. When the velocity on a vertex is updated, it is replaced in that variable. So when calculating the new iteration, the most accurate velocities on the neighbour vertices are used. Now equation (4.14) and (4.15) can be put into the SOR iteration. This gives:

$$u^{k+1}_i = (1-\omega)u^k_i + \omega\frac{1}{e^i_u}\left(-\sum_{s=1}^{n} u_s\left(4N^{i,s}_{xx} + N^{i,s}_{yy}\right) - 3v_{xy} + \frac{\rho g h_x}{\bar{v}}\right). \tag{4.16}$$

This can be rearranged to get the equation that is used in the model. Also the equation for v_i is given, which can be derived in the same way:

$$\begin{aligned} u^{k+1}_i &= u^k_i - \frac{\omega}{e^i_u}\left(\sum_{s=1}^{n} u_s\left(4N^{i,s}_{xx} + N^{i,s}_{yy}\right) + e^i_u u^k_i + 3v_{xy} - \frac{\rho g h_x}{\bar{v}}\right), \\ v^{k+1}_i &= v^k_i - \frac{\omega}{e^i_v}\left(\sum_{s=1}^{n} v_s\left(4N^{i,s}_{xx} + N^{i,s}_{yy}\right) + e^i_v v^k_i + 3u_{xy} - \frac{\rho g h_y}{\bar{v}}\right). \end{aligned} \tag{4.17}$$

This is the equation that can be found in the inner loop of the model.
So for every timestep, the two nested iterative loops have to be computed. The outer loop is repeated six times, but the inner loop needs a lot of iterations to be close enough to the solution. Also, the inner loop needs to be done for every vertex of the mesh. If this process can be made more efficient, it will reduce the computation time.

5 Demo model

To get a better understanding of how differential equations can be solved on a irregular triangular mesh, a demo model is used. This demo model can be used to test different iterative methods on an irregular mesh. If these tests were done directly in UFEMISM, it would take a lot of time since the SSA equations have a lot of input values from other parts of the model. Since the goal of this study is to investigate how to increase the convergence rate of the SOR method on an irregular mesh, there is no need for all those extra input values.

In the demo model, differential equations can be solved using the same method as in UFEMISM: with different iterative methods on an irregular mesh. Therefore, the behaviour of different iterative methods can be examined more easily. Furthermore, an equation that is a bit simpler than the equations in the SSA will make it easier to understand how iterative methods behave on an irregular mesh.
An important note here is that the equations that are solved in the demo model have nothing to do with the equations in the ice model. It has no physical meaning to solve an arbitrary equation on a mesh like this. The only reason for doing this is to get a better understanding of the methods used in UFEMISM in such a way that not the whole ice model needs to be runned.
In this section, a description of the demo model is given. After that, the methods defined in section 2 and 3 are tested. Hopefully, this will give some insight in how the process of solving equations on this mesh can become more efficient. Results from the experiments in this demo model can eventually be tested with the SSA.

5.1 Demo Laplace-equation

For this demo model, a Matlab code is used that solves the Laplace equation on an irregular mesh using the Jacobi method, the Gauss-Seidel method and SOR. The mesh in this example is made with the mesh refinement algorithm of UFEMISM and is based on the geometry of Greenland. In figure 6 all the vertices $v_i = (x_i, y_i)$ of the mesh are given. The mesh has absolutely nothing to do with the equation that will be solved, but this mesh is used because it is the same type of mesh as used in the ice model. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a function defined on every vertex of the mesh. Function f is the solution of the Laplace equation, which is a second order partial differential equation. The Laplace equation is given by:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0. \tag{5.1}$$

To define the boundary conditions, let x_{min} and x_{max} be the minimum and maximum value of the x-coordinates of the vertices v_i in the mesh. Then y_{min} and y_{max} are defined in the same way.
If v_i is at the most left part of the map ($x_i = x_{min}$), $y > \frac{1}{4}(y_{max} - y_{min})$ and $y < \frac{3}{4}(y_{max} - y_{min})$, the boundary condition is given by $f(v_i) = 1$. This is indicated with the red line in figure 6. In all other cases $f(v_i) = 0$ at the boundary.
To solve this problem iteratively, the equation needs to be discretized. To do this, the method described in section 2 is used.

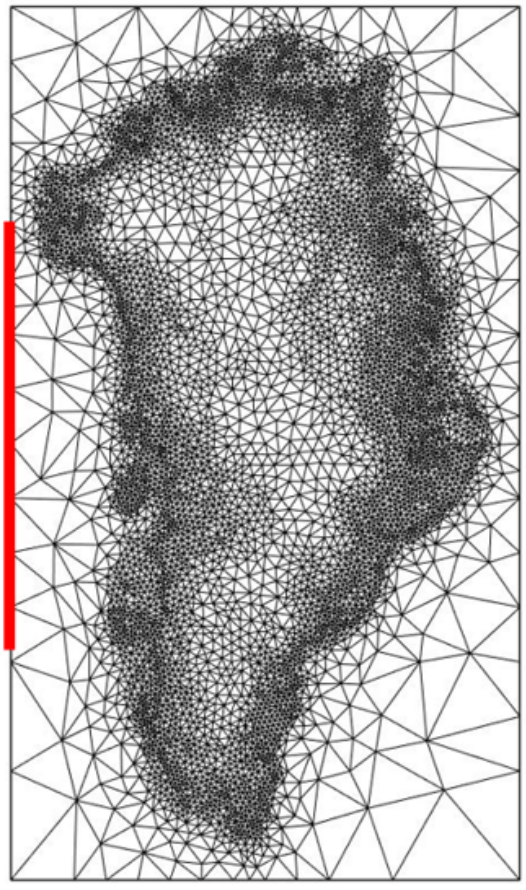


Figure 6: *The mesh that is used for the demo model. The red live indicates the place where the boundary condition $f_i = 1$ holds.*

The partial derivatives in equation (5.1) can be approximated in vertex v_i as linear combination of the function value in v_i and its neighbouring vertices. Using equations (2.21) and (2.22), an approximation of the Laplace equation can be made in every vertex v_i . The notation f_i is used again for the function f in vertex v_i:

$$f_i \left(N^i_{xx} + N^i_{yy}\right) + \sum_{s=1}^{n} f_s \left(N^{i,s}_{xx} + N^{i,s}_{yy}\right) \approx 0. \tag{5.2}$$

In this equation, this sum is taken over all the vertices v_s that are a neighbour of v_i. This equation only gives the discretisation for the Laplace equation on vertex v_i.
For all vertices v_i, these equations can be put in a linear system of the form $A\boldsymbol{f} = \boldsymbol{b}$. To do this, recall that every vertex in the mesh can be denoted with its vertex index i (section 2).
If the mesh has n vertices, there are also n equations like equation (5.2) that need to be solved. Let $\boldsymbol{f} = (f_1, f_2, ..., f_n)$ denote the values of f on every vertex v_i.
First, assume vertex v_i is not at the boundary. Then for matrix $A \in \mathbb{R}^{n\times n}$, row i can be defined:

$$a_{ii} := \left(N^i_{xx} + N^i_{yy}\right)$$

$$a_{ij} := \begin{cases} \left(N^{i,s}_{xx} + N^{i,s}_{yy}\right) & \text{if } v_i \text{ and } v_j \text{ are neighbours,} \\ 0 & \text{anywhere else.} \end{cases}$$

In this way, component i of vector $\boldsymbol{b}$, can be calculated by multiplying row i of matrix A with vector $\boldsymbol{f}$. It can be verified that this product is equal to equation (5.2) so component i of vector $\boldsymbol{b}$ should be equal to zero.
This way of defining matrix A works for all vertices v_i that are not on the boundary. If v_k is a vertex that is on the boundary, applying matrix A should not change the value of f^k. Therefore, if v_k is a boundary vertex, the row k of matrix A can be defined:

$$a_{ij} := \delta_{ij}$$

Then, component k of vector $\boldsymbol{b}$ is equal to the boundary condition defined for v_k, which can be zero or one depending on the location of verex k.
Now, let $\boldsymbol{f}^*$ be the solution of the linear system $A\boldsymbol{f} = \boldsymbol{b}$. This solution can be calculated in Matlab by calculating $\boldsymbol{f}^* = A\backslash\boldsymbol{b}$. Figure 7 shows the solution on every vertex.

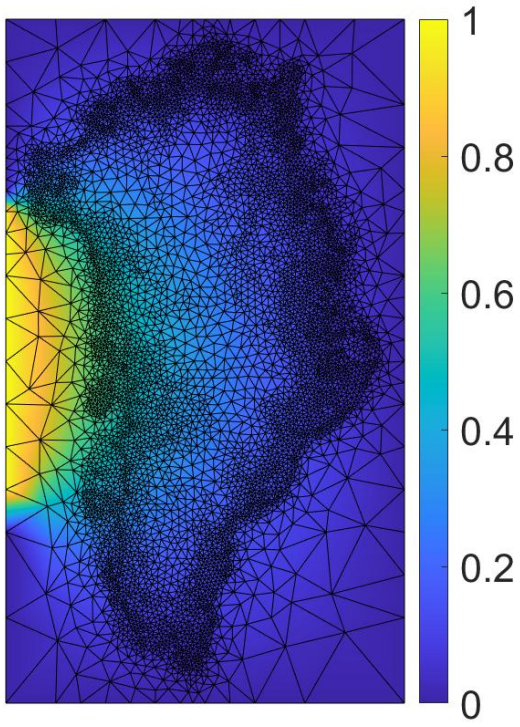


Figure 7: *This figure shows the solution $\boldsymbol{f}^*$ of the linear system $A\boldsymbol{f} = \boldsymbol{b}$.*

5.1.1 Finding the solution using iterative methods

In UFEMISM, the equations of the SSA are solved using iterative methods. The goal is to test those methods in the demo model. Therefore, the methods described in section 3 are used to find the solution of the linear system that is defined in the previous section. Equation (5.2) gives the discretisation of the Laplace equation on every vertex v_i. This can be rewritten in such a way that for every vertex v_i, f_i can be approximated by:

$$f(v_i) \approx \frac{-\sum_{s=1}^{n} f^s \left(N_{xx}^{i,s} + N_{yy}^{i,s}\right)}{\left(N_{xx}^{i} + f^i N_{yy}^{i}\right)}. \tag{5.3}$$

To solve this in an iterative way, an initial guess $\boldsymbol{f}^0$ is made:

$$f_i^0 = \begin{cases} b_i & \text{if } v_i \text{ is at the boundary ,} \\ 0 & \text{everywhere else.} \end{cases} \tag{5.4}$$

If v_i is at the boundary of the mesh, b_i denotes the boundary condition of vertex v_i. Using this initial guess, the Jacobi method can be used get the solution. The next iteration is given by the equation below:

$$f_i^{k+1} = \frac{-\sum_{s=1}^{n} f_s^k \left(N_{xx}^{i,s} + N_{yy}^{i,s}\right)}{\left(N_{xx}^{i} + N_{yy}^{i}\right)}. \tag{5.5}$$

In the same way, the next iteration step can be found using the Gauss-Seidel method. This will be done for every vertex v_i, starting with $i = 1$ in a loop. Without any remark, equation (5.5) gives the Jacobi method. Assume v_i has a neighbour v_j and $j < i$. Then the value of f_j^{k+1} is already computed in the loop, so in equation (5.5), f_s^{k+1} should be used in stead of f_s^k. When this is applied, it gives the Gauss-Seidel approximation.
To solve the Laplace equation using SOR, the next iteration step from Gauss-Seidel is used. This can be substituted in equation (3.4) and the resulting formula gives the next iteration step of the SOR method:

$$f_i^{k+1} = (1-\omega) f_i^k + \omega \frac{-\sum_{s=1}^{n} f_s^k \left(N_{xx}^{i,s} + N_{yy}^{i,s}\right)}{\left(N_{xx}^{i} + N_{yy}^{i}\right)}. \tag{5.6}$$

Again, note that if there is a neighbour v_s with 'vertex-index' j which has the property that $j < i$, the new value should be used. After every iteration step, the relative error is calculated by looking at the maximum value of $(f_i^{k+1} - f_i^k)$. When this value is below some specific value, the iteration process is stopped.

5.1.2 Convergence of different iteration methods

In figure 8, the convergence of each of the three iteration methods is shown. This is done by calculating $||\boldsymbol{f}^k - \boldsymbol{f}^*||_\infty$ after every iteration k.
Figure 8a shows the convergence of the Jacobi method. From this figure, it can be concluded that the Jacobi method converges slow compared to the other two methods. After 400 iterations, there is a vertex where $|f_i^k - f_i^*| > 0.4$. Every component of $\boldsymbol{f}^*$, has the property that $f_i^* \in [0, 1]$, so this is still too large. The convergence of the Gauss-Seidel method is shown in figure 8b and is faster than the Jacobi method, but after 400 iterations, the value of $||\boldsymbol{f}^k - \boldsymbol{f}^*||_\infty$ is still too large.
The convergence rate of the SOR method is much faster. In figure 8c, convergence for $\omega = 1.795$ is shown. This value was determined experimentally to yield the lowest convergence rate. When choosing $\omega > 1.795$, divergence will occur. This can be seen in figure 8d.

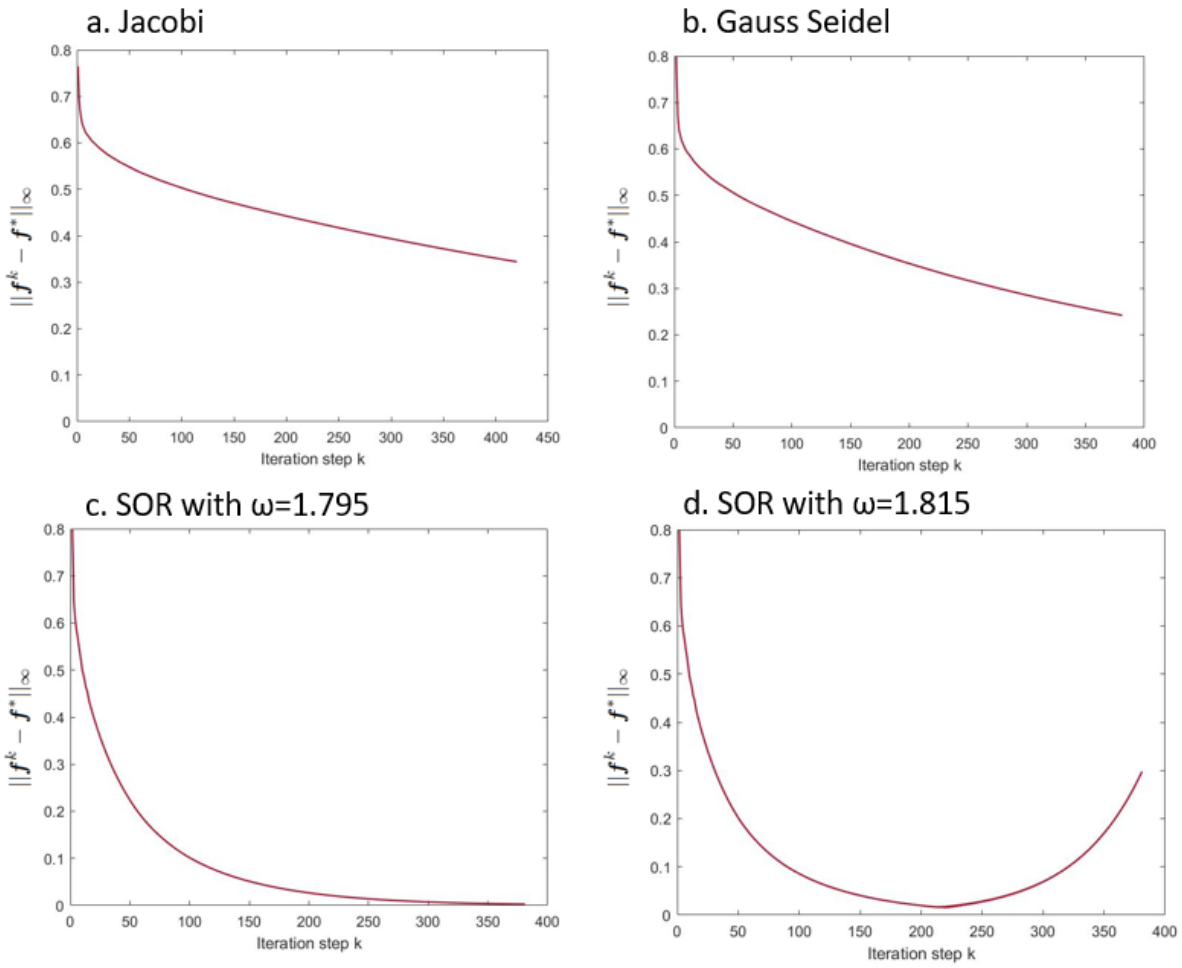
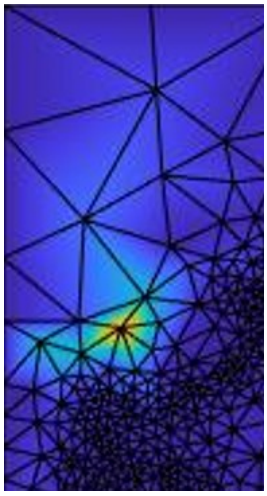
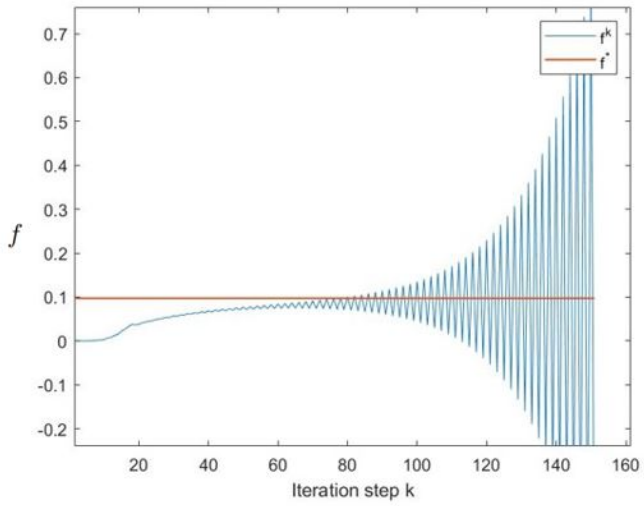


Figure 8: *In this figure, the Jacobi, Gauss Seidel and SOR method are compared. For every iteration step k, the value $||\boldsymbol{f}^k - \boldsymbol{f}^*||_\infty$ is shown. The convergence of SOR is shown for two different values of ω.*

It is remarkable that divergence starts to occur at one vertex. The vertex where divergence occurs has vertex index 6464 and is shown in figure 9a. In figure 9b, the values of f on vertex 6464 are shown for every iteration step. The red line indicates the solution f^* of the linear system on vertex 6464, so the method should converge to that red line.
Because of the way the first guess is chosen, f^{k+1} is equal to f^k for the first few iterations in vertex 6464. This is because the value of f on vertex v_i is a linear combination of the values of f on the surrounding vertices. It



(a) *In this figure, vertex* 6464 *is shown. The yellow color means that the solution is way bigger than at all the other places.*



(b) *For every iteration step k, the blue line shows the value of* f^k *on vertex* 6464.

Figure 9: Divergence on vertex 6464.

takes some iterations before there is a neighbour that changes this vertex. But from this moment, the values of f^k will improve for every k until divergence occurs. The solution starts to diverge after approximately 60 iterations. The divergence on this vertex will eventually influence the values on the other vertices. So, after a while, divergence will also occur at all the other vertices.

5.1.3 Adjusting ω on some vertices

For a relaxation parameter that is too large, divergence starts to occur at one single vertex. This will eventually influence all the other vertices and will lead to oscillations that gradually spread out over the mesh. This leads to the following question: Maybe it helps to adjust ω only on this vertex. When a lower value of ω is used on this vertex, it will no longer diverge and can not influence all the other vertices anymore. At the same time, all the other vertices can be given a higher ω. Hopefully, this will lead to faster convergence. To see if this works, a small experiment is done. The vertex where the divergence occurred is vertex 6464. When taking another ω for this vertex, the method can be stable again.

Let $\omega = 1.6$ for this vertex and let $\omega = 1.85$ for all the other vertices. In this way, the method will converge. Again, when choosing a larger value for ω on all the other vertices, there is another vertex where divergence occurs. In this case vertex 6011 is unstable. After also lowering the value of ω for this vertex, the original value of ω can be a bit larger and the method will converge with less iterations.
After some trial with the Matlab model, the value for ω is adjusted for some vertices. When choosing $\omega = 1.6$ for the vertices with index $i \in \{6464, 6011, 6433, 6381, 5044\}$ and $\omega = 1.875$ anywhere else, the method converges faster. This result is shown in figure 10. Both figures show the norm $||\boldsymbol{f}^k - \boldsymbol{f}^*||_\infty$ for every iteration step k. From this figure, it can be concluded that adjusting ω for only five vertices already leads to faster convergence.
This small experiment shows that the optimal value for ω is different depending on the location in the mesh. Because the mesh is irregular, this value can vary substantially between the vertices. If there is one vertex where the iteration process behaves different than at all the other ones, this can lead to an average ω that is not optimal for all the other vertices. Therefore, it might be a good idea to make an ω that depends on the properties of the mesh, which can be calculated in advance to make the iteration process more efficient. If adjusting the relaxation parameter for only five vertices already helps that much, defining a unique relaxation parameter for every vertex can help to make the iteration process even faster.

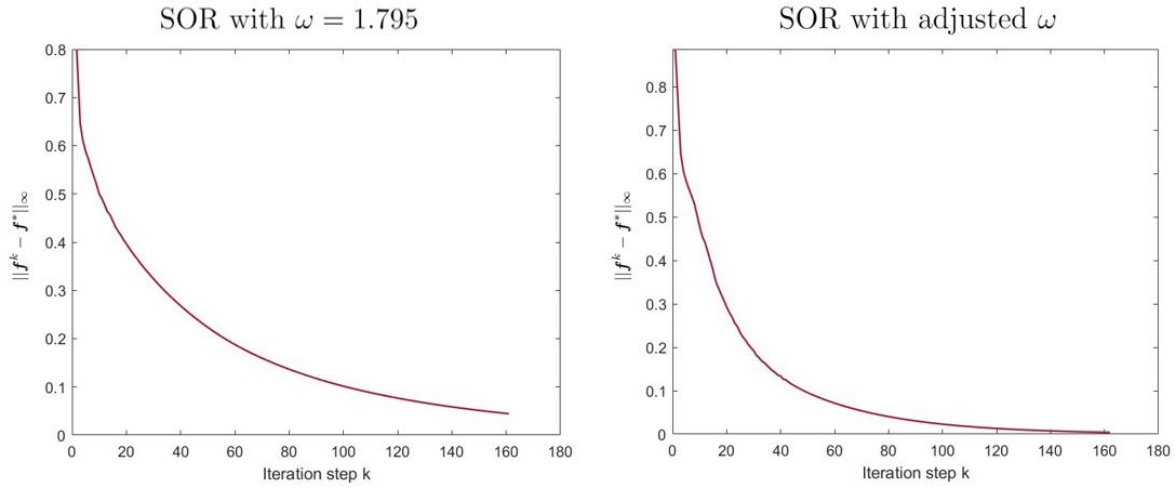
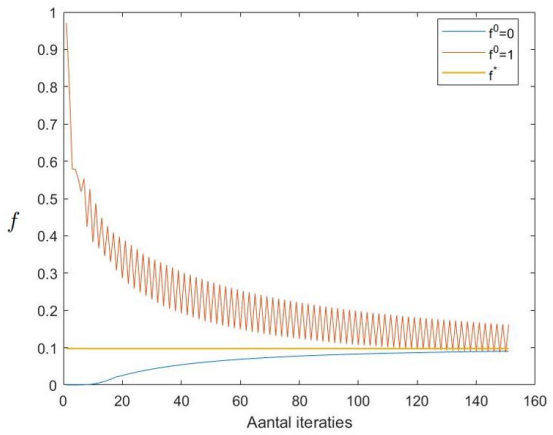


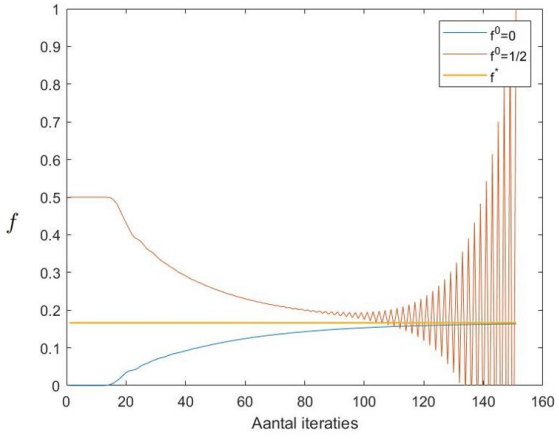
Figure 10: *In this figure, the convergence of two different methods is compared for the first* 160 *iterations. The left figure shows convergence for* $\omega = 1.795$ *on all the vertices. The right figure shows convergence with an adjusted* ω *on five vertices and a value of* $\omega = 1.795$ *on all the other vertices. It can be seen that the convergence in the right figure is much faster than in the left figure.*

5.2 Comparing the results with some convergence theory

The experiments in the sections above were done with one initial guess (equation (5.4)). However, the first guess can influence the behaviour of the convergence of an iteration method. Figure 11 shows two examples of a vertex where convergence behaves differently for two choices of $\boldsymbol{f}^0$. There is no proof that the value $\omega = 1.795$ from section 5.1.1 works for all $\boldsymbol{f}^0$.



(a) *In this figure, convergence for* $\omega = 1.795$ *is shown on vertex 6464. Here, the behaviour with two different values of* f^0 *is compared. They both converge, but when* $f^0 = 1$*, it is less stable.*



(b) *This figure shows a different behaviour for different* f^0 *an a vertex in the mesh. The orange line is the exact solution of the linear system. In this case,* $\omega = 1.6$ *on vertex 6464 and 1.9 anywhere else. When taking* $f^0 = 0$ *(blue), the method converges, but when taking* $f^0 = 0.5$*, the method is unstable (red).*

Figure 11

As described in section 3, every iterative method that finds the solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ can be described in such a way that $\boldsymbol{x}^{k+1} = G\boldsymbol{x}^k + \boldsymbol{h}$ with $G \in \mathbb{R}^{n\times n}$ the iteration matrix and $\boldsymbol{h} \in \mathbb{R}^n$. For the SOR method, matrix G depends on the value of ω and is given by [6]:

$$G_\omega = (D - \omega L)^{-1}[(1-\omega)D + \omega U]. \tag{5.7}$$

Here, D is the diagonal of matrix A, L is the lower triangle of A and U the upper triangle. After calculating this matrix, it can be checked with Matlab that this matrix has indeed n linearly independent eigenvectors, so theorem 3.4.1 should hold here. For $\omega = 1.795$, the spectral radius of the matrix G_ω can be calculated and is equal to 1.0587. This is a bit surprising because theorem 3.4.1 states that an iterative method converges for all f^0 if and only if the spectral radius of the iteration matrix is less then one. Therefore, the choice of $\omega = 1.795$ might not work for all f^0. Figure 11a shows that for $\omega = 1.795$, the method is still converging, but is less stable when choosing $f^0 = 1$. When calculating the eigenvalues of G_ω, there are only three eigenvalues which have the property that $|\lambda_i| > 1$. Those eigenvalues are only a bit larger than one in absolute value. All other eigenvalues are smaller than one in absolute value. Because there are more then 6000 eigenvalues, it can happen that the influence of those three eigenvalues is negligible for the first iterations. The influence of those three eigenvalues will only become significant after many iterations. In the experiments where the value of $\omega = 1.795$ was determined, the solution was close enough after 400 iterations, so maybe the three eigenvalues only become significant when iteration step k is much larger than that. This can be the reason why the value of $\omega = 1.795$ works.
To be sure that the method converges for all first guesses, the value of ω can be adjusted a bit. When taking ω slightly smaller, it can be checked that all eigenvalues of G_ω are less then one in absolute value. This is already true for $\omega = 1.75$, where the spectral radius is 0.99941. In this case the method is guaranteed to converge for all initial guesses f^0.
In UFEMISM, velocity of the previous timestep is used as a first guess, so the initial error can be different for every timestep. In this case, it is good to have an ω which guarantees convergence for all first guesses.
In section 3, another relation between the spectral radius of the iteration matrix and convergence ratio was shown. For large k, the the convergence rate can be approximated by the spectral radius of the iteration matrix. In the demo model, this can be checked by calculating the spectral radius and the convergence rate for a given ω. Therefore, a small spectral radius will lead to faster convergence. So to find the optimal value of ω, the goal is to find an iteration matrix G_ω which has a minimum spectral radius. In [14], some research is done on how to find the value of ω in such a way that the spectral radius of G_ω is minimal. Calculating an optimal ω mathematically takes much time since the convergence rate depends on the spectral radius. The experiment in section 5.1.3 shows that choosing an optimal ω for every vertex v_i can make the iteration method faster. There might also be ways to determine an optimal relaxation parameter that depends vertex v_i mathematically. There are more existing variations on the SOR method which can be helpful to do this, but more research has to be done for that.

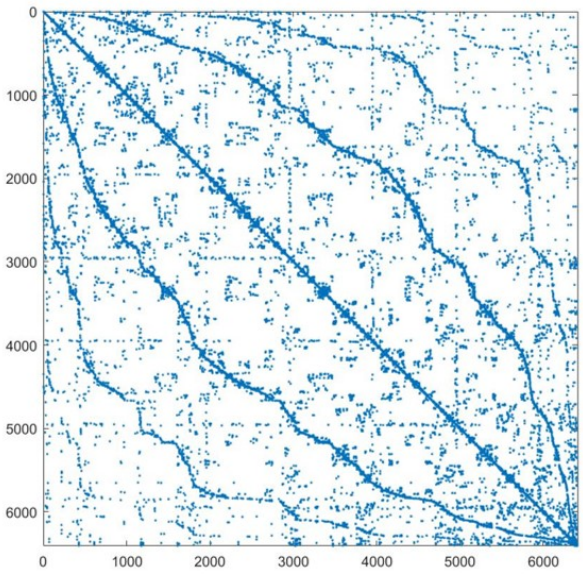


Figure 12: *In this figure, the symmetric pattern of the iteration matrix is visualized.*

Another remarkable thing is the symmetry of matrix A from the linear system in section 5.1.
For row i, the columns j of matrix A which corresponds to any neighbouring vertex are nonzero. When v_j is a neighbour of v_i, v_i is also a neighbour of v_j so matrix A is almost structurally symmetric. A matrix is structurally symmetric if it is symmetric for its nonzero elements. Because the boundary conditions

are included in matrix A, the rows of the matrix which correspond to the vertices on the boundary disturb this symmetry. If the boundary vertices are excluded, a new linear system $\tilde{A}\boldsymbol{x} = \tilde{\boldsymbol{b}}$ can be made that only includes the vertices that are not on the boundary. Then, this linear system is structurally symmetric. The symmetry of the matrix $\tilde{A}$ is visualized in figure 12.
The symmetry of the iteration matrix might be useful to estimate a unique ω for every vertex of the mesh.

To examine how to find a relaxation parameter ω that depends on the properties of the mesh, some more experiments with the demo model can be done. If there is a relation between the convergence on vertex v_i and some properties of the mesh on that vertex, this can help to define different values of ω on every vertex. Summarized: to find an optimal ω on every vertex, more experiments have to be done.

6 Conclusion and further research

The goal of this study was to find ways to improve the convergence rate of the SOR method on an irregular mesh. To investigate this, some experiments were done with a demo model. Here, different iteration methods were tested on the Laplace equation.

From the results of the demo model, it can be concluded that the SOR method converges much faster than the Jacobi or Gauss-Seidel method. With a relaxation factor $\omega = 1.795$, the SOR method is showing the fastest convergence. When taking a higher relaxation factor, divergence will occur on some vertices.
However, this relaxation factor was experimentally found. There is no proof that this method works for all first guesses. Therefore it might be better to take a value of ω which has the property that the spectral radius of the iteration matrix is less then one. For example, the value $\omega = 1.75$ already has this property. With this value of ω, the method might take a few more iterations, but convergence is guaranteed.

The optimal value of ω depends on properties of the linear system, which depends on the boundary conditions, the equation and the mesh that is used. So, for every equation this can work in another way. Therefore, it is important to mention that the optimal relaxation factor $\omega = 1.75$ only works for the Laplace equation. The value of ω can be different for another equation.
In the demo-model, the first guess used for the iteration process was not that accurate. Before the iterations start, only some values of f^0 on the left part of the mesh are equal to one, whereas the values of f^0 for the rest of the mesh are equal to zero. Since vertices only have an influence on their neighbours, it takes a while for all vertices are reached, so this leads to more iterations.
In the SSA equation of UFEMISM, the velocity has to be calculated for every timestep. The solution of the previous timestep can be used as a first guess for the solution of the current timestep. Since this guess of the solution is already close, the problem described above will not occur here.

Another important result from the experiments in the demo model is that the SOR method can become even faster when choosing an optimal value for ω which depends on vertex v_i. A small experiment with the demo model in section 5.1.3 showed that adjusting the relaxation parameter on only five vertices already helps to reduce the amount of iterations that are needed. Therefore, defining an optimal relaxation parameter on every vertex can reduce the amount of iterations even more.

Eventually, the goal is to apply the results from the demo model on the SSA equations in UFEMISM. Because the SSA consists of other equations with other boundary conditions, this leads to another linear system. Therefore, conclusions from the demo model can not immediately be implemented in the ice model. It would be helpful to find a correspondence between the value of ω and the properties of the mesh. If a relation between ω and the properties of the mesh can be found, this can help to define a relaxation parameter on every vertex. Eventually, this can be used in the SSA equation of UFEMISM to make it converge faster. To find any correspondence between ω and properties of an irregular mesh, some more research has to be done. This can be done by having a look at how the iteration process converges in every vertex. If there is a relation between the shape of the mesh around vertex v_i and the convergence rate, this knowledge might be useful to adjust the relaxation parameter for every vertex.

References

[1] Richard Barrett et al. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

[2] Constantijn J Berends, Heiko Gölzer, and Roderik SW van de Wal. "The Utrecht Finite Volume Ice-Sheet Model: UFEMISM (version 1.0)". In: *Geoscientific Model Development Discussions* (2020), pp. 1–32.

[3] Ed Bueler and Jed Brown. "Shallow shelf approximation as a "sliding law" in a thermomechanically coupled ice sheet model". In: *Journal of Geophysical Research: Earth Surface* 114.F3 (2009).

[4] Betham Davies. *What is the global volume of land ice and how is it changing?* URL: `http://www.antarcticglaciers.org/glaciers-and-climate/what-is-the-global-volume-of-land-ice-and-how-is-it-changing/`.

[5] Judith D Gardiner. "Fundamentals of matrix computations (David S. Watkins)". In: *SIAM Review* 35.3 (1993), pp. 520–521.

[6] A Hadjidimos. "Successive overrelaxation (SOR) and related methods". In: *Journal of Computational and Applied Mathematics* 123.1-2 (2000), pp. 177–199.

[7] Douglas R MacAyeal. "Large-scale ice flow over a viscous basal sediment: Theory and application to ice stream B, Antarctica". In: *Journal of Geophysical Research: Solid Earth* 94.B4 (1989), pp. 4071–4087.

[8] Gary L Miller, Steven E Pav, and Noel Walkington. "When and Why Ruppert's Algorithm Works." In: (2003), pp. 91–102.

[9] LW Morland and IR Johnson. "Steady motion of ice sheets". In: *Journal of Glaciology* 25.92 (1980), pp. 229–246.

[10] Jim Ruppert. "A Delaunay refinement algorithm for quality 2-dimensional mesh generation". In: *J. Algorithms* 18.3 (1995), pp. 548–585.

[11] Christian Schoof. "A variational approach to ice stream flow". In: *Journal of Fluid Mechanics* 556 (2006), p. 227.

[12] Andrew Shepherd et al. "Mass balance of the Greenland Ice Sheet from 1992 to 2018". In: *Nature* 579.7798 (2020), pp. 233–239.

[13] RSW van de Wal et al. "Uncertainties in long-term twenty-first century process-based coastal sea-level projections". In: *Surveys in Geophysics* 40.6 (2019), pp. 1655–1671.

[14] David M Young. *Iterative solution of large linear systems*. Elsevier, 2014.