Rapport de projet pluridisciplinaire :
Diffusion non linéaire en sciences de la terre modélisation des glaciers

Tania Mendes, Mathieu Nguyen, Samson Petros,
Yazid Bouhria, Elyas Assili, Maryam Aarab

Encadrante : Laetitia Le Pourhiet

19 janvier 2022

Table des matières

Introduction

L'écoulement d'un glacier dépend de plusieurs facteurs qui sont principalement l'accumulation de neige, la fonte de glace, la gravité et le niveau de la pente du lit rocheux. Les écoulements sont considérés comme fluides, incompressibles, visqueux et non linéaires, d'où l'intérêt de les modéliser avec les relations de Stokes.
C'est un processus continu, mais il est très difficile de les résoudre analytiquement. Pour cela, les modélisateurs utilisent donc des méthodes numériques. L'intérêt ici va être d'énoncer les relations générales de Stokes appliquées aux glaciers, d'énoncer le modèle Shallow Ice Approximation (SIA) afin d'en déterminer les avantages et les limites dans l'analyse de l'écoulement d'un glacier en fonction du temps.

Dans le cadre de ce projet, nous allons utilisé le langage Julia pour la résolution de systèmes d'équations différentielles par **différences finies** afin de pouvoir prédire l'écoulement glaciaire du Groenland. Nous utilisons Julia car c'est un langage de programmation moderne, polyvalent et qui permet de parralèliser des GPU sans demander trop de ressources. C'est un langage créé en 2009 par des chercheurs et ouvert au grand public en 2012. C'est un langage de haut niveau, dynamique et conçu pour des calculs scientifiques. Sa syntaxe est similaire à Python, R ou encore Matlab.

1 Présentation des techniques de résolution

1.1 Les équations de Stokes

Les équations de Stokes [1] ont comme principal objectif de décrire les mouvements des fluides. Pour procéder correctement, il faut connaître sa vitesse en tout point de l'espace : son champ de vitesse. Ainsi les équations de Stokes permettent de décrire le champ de vitesse d'un fluide. Dans un fluide nous considérons deux types de forces à savoir, les forces de volume (donc de poids) et les forces de surface correspondant aux frottements visqueux.

Afin de pouvoir appliquer ces équations dans le cas des écoulements des glaciers, nous considérerons ces derniers comme des fluides incompressibles, visqueux et non linéaires. Il s'agit d'un système d'équations pouvant être appliqué à tout type de glacier.

Ce système permet une très grande précision dans les résultats puisqu'il prend en compte toutes les contraintes au nombre de neuf, qu'elles soient longitudinales, transversales, verticales ou encore horizontales.

Le second membre correspond au terme d'inertie. Ce terme peut être comparé aux effets visqueux par le nombre de Reynolds :

$$Re = \frac{\rho V L}{\mu} \tag{1}$$

avec V une vitesse caractéristique et L une longueur caractéristique.

L'équation de Navier-Stokes peut être assimilé à la deuxième loi de Newton appliquée aux fluides

$$\sum \vec{F} = m\vec{a} \tag{2}$$

avec $\vec{F}$ les forces s'exerçant sur le volume, m sa masse et $\vec{a}$ son accélération. Or, la glace est assimilée à un fluide très visqueux, le nombre de Reynolds est alors petit et ainsi on peut négliger l'inertie. Donc l'accélération est nulle, ce qui nous mène à l'équation suivante :

$$\sum \vec{F} = 0 \tag{3}$$

La somme des forces est la somme des forces $\vec{F}v$ qui agissent sur tout le volume et des forces $\vec{F}s$ qui sont les frottement sur les bords et à l'intérieur de la glace.

Les forces de volume $\vec{F}v$ induites par la gravité, agissent sur toutes les particules du volume élémentaire et s'écrient :

$$\vec{F}v = \Delta_m \vec{g} \tag{4}$$

avec Δ_m la masse élémentaire et g l'accélération de la pesanteur. Δ_m peut se réécrire de cette manière :

$$\Delta_m = \int \rho dV = \rho dxdydz \tag{5}$$

avec ρ la masse volumique et $dV = dxdydz$ le volume élémentaire.
Ainsi la $\vec{F}v$ la force de volume est

$$\vec{F}v = \int \rho \vec{g} dV \tag{6}$$

Les forces de surface $\vec{F}s$ sont les frottement au bord et à l'intérieur du glacier. Pour le cas d'un glacier nous prenons en compte les contraintes normales à la surface σ :

$$\vec{F}s = \int \sigma \vec{n} dS \tag{7}$$

avec $\vec{n}$ le vecteur normal à la surface dirigé vers l'extérieur. Le théorème de divergence permet de réécrire $\vec{F}s$:

$$\vec{F}s = \int \nabla \sigma dV \tag{8}$$

Le tenseur de contrainte σ s'écrit :

$$\sigma = 2\mu\varepsilon - I_d P \tag{9}$$

avec μ le coefficient de viscosité, $\varepsilon = \frac{1}{2}(\nabla v + \nabla v^t)$ le taux de déformation, I_d la matrice identité et P la pression. $2\mu\varepsilon$ est la partie qui concerne la déformation et I_d est la partie isotrope. En intégrant l'expression du tenseur de contrainte on obtient :

$$\vec{F}s = \int \nabla(2\mu\varepsilon - I_d P)dV = \int \nabla 2\mu\varepsilon dV - \int \nabla P dV \tag{10}$$

Ainsi en récapitulant la somme des forces, on obtient

$$\begin{aligned} \sum \vec{F} = \vec{F}s + \vec{F}v = 0 \\ \Leftrightarrow \int \nabla 2\mu\varepsilon dV - \nabla \int P dV + \int \rho\vec{g} dV = 0 \\ \Leftrightarrow -\int \nabla 2\mu\varepsilon dV + \nabla \int P dV = \int \rho\vec{g} dV \\ \Leftrightarrow -\nabla 2\mu\varepsilon + \nabla P = \rho\vec{g} \end{aligned} \tag{11}$$

Ainsi, nous retrouvons bien l'expression des équations de stokes

$$\begin{cases} -div(2\mu\varepsilon(\vec{v})) + \nabla P = \rho\vec{g} \\ div(\vec{v}) = 0 \end{cases} \tag{12}$$

avec $\nabla = (\frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z})$, $\vec{v}$ la vitesse, $\vec{g}$ l'accélération de pesanteur, ρ la masse volumique, μ le coefficient de viscosité et ε le taux de déformation. $div(\vec{v}) = 0$ car il n'y a pas de variation de volume.

Néanmoins, le Groenland est une calotte grande de milliers de kilomètres. Il faut alors un nombre considérable de points pour résoudre ces équations. De plus du fait de la complexité de celles-ci ainsi que des géométries de la calotte, cela est coûteux en temps de calcul. Il existe de nombreuses simplifications de ces équations afin de minimiser le temps de calcul. Parmi elle, une des plus connue est l'approximation de la couche mince (Shallow Ice Approximation)appliquée pour la première fois sur la glace par Kolumban Hutter (1983). Ce modèle permet de réduire le problème de résolution 3D à un problème 2D. Il permet des calculs beaucoup plus simples. Ainsi, les calottes étant relativement minces, soit quelques kilomètres d'épaisseur, nous pouvons négliger les (variations verticales de la vitesse) les contraintes de cisaillement longitudinales et utiliser le modèle de l'approximation de la couche mince.

1.2 Shallow ice approximation (SIA)

Pour cette partie, nous nous sommes renseignées à travers le cours de Martina Schäfer [3] et aussi grâce à un article publié dans le Cambridge University Press [4].
D'après une étude [5], des chercheurs assurent que la résolution du système des équations de Stokes est très coûteuse en temps pour des applications réalistes (avec une complexité de **nlog(n)**). C'est pour cette raison qu'ils utilisent une simplification de ces équations basée sur le fait que sur de très grandes calottes, l'épaisseur est extrêmement faible par rapport à la dimension horizontale. Pour cela, nous allons utiliser le modèle SIA (Shallow Ice Approximation). Sa validité dépend du rapport d'aspect caractéristique ζ des objets glaciaires :

$$\zeta = \frac{e}{L} \tag{13}$$

Avec e l'épaisseur du glacier et L la largeur du glacier. Ce rapport est par exemple de 10^{-3} pour l'Antarctique et de 5.10^{-3} pour le Groenland. En effet, la validité de la SIA diminue lorsque le rapport d'aspect caractéristique ζ augmente.

De plus, on néglige ici les interactions internes (aussi appelées bridge effect), ce qui nous permet de séparer les vitesses horizontales et verticales. μ est considéré isotrope et s'écrit :

$$\mu = \frac{B}{2\varepsilon(\vec{v})^{\frac{n-1}{n}}} \tag{14}$$

Où B est la solidité de la glace et n est le coefficient de la loi de Glen (on prendra n = 3)
La vitesse horizontale devient donc solution de :

$$-div(2\mu\varepsilon(\vec{v})) = \rho\vec{g}\frac{\partial S}{\partial x} \tag{15}$$

L'évolution de la surface est déterminée en résolvant une équation hyperbolique, celle-ci est dictée par la conservation de masse. C'est cette équation que l'on retiendra car elle nous permettra d'étudier la hauteur de la glace au fur et à mesure du temps. Cette équation s'écrit :

$$\frac{\partial H}{\partial t} = -\nabla H\vec{v} + Ms - Mb \tag{16}$$

Avec H l'épaisseur de la glace, $\vec{v}$ la vitesse horizontale moyenne, Ms la conservation de masse en surface et Mb la fonte de la glace à la base. Etant donné que $\vec{v}$ est la vitesse horizontale, c'est également une dérivée partielle selon x. On peut donc écrire cette équation SIA comme la résolution d'une équation de diffusion non linéaire :

$$\frac{\partial H}{\partial t} = D(H)\frac{\partial^2 H}{\partial x^2} + M \tag{17}$$

avec D le coefficient de diffusion, H l'épaisseur du glacier et M la conservation de masse.
Ce coefficient D est non linéaire et se calcule par :

$$D(H) = aH^{n+2}\sqrt{(}\nabla S\nabla S)^{n-1} \tag{18}$$

Avec a la viscoté de la glace (qui est une constante) et n la **loi de Glen** qui est égale à 3.
L'épaisseur H est donnée par l'équation suivante :

$$H(x) = S(x) - B(x) \tag{19}$$

où B est la hauteur de l'encaissant rocheux et S la topographie de la glace. H est représentée sur la figure 1 :

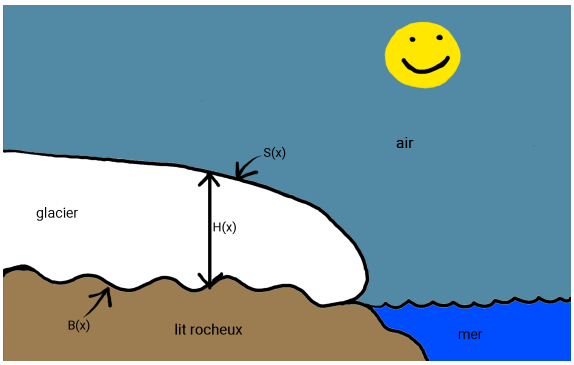

FIGURE 1 – Représentation de l'épaisseur de la glace H, la hauteur du lit rocheux B et la topologie de la glace S

Les données topographiques, l'élévation du substratum rocheux et l'épaisseur de la glace proviennent du jeu de données BedMachine Greenland v3.

Il faut savoir que l'écoulement d'un glacier dépend de plusieurs facteurs : la neige, glace, pluie, fonte, perte de glace, gravité. La conservation de masse M se calcule de la manière suivante :

$$M = accumulation + ablation \tag{20}$$

Ainsi la conservation de masse permet d'équilibrer les 2 phénomènes à la surface en fonction de la ligne d'équilibre. L'altitude de la ligne d'équilibre (ELA) est où la zone d'accumulation, au sommet du glacier, est égale à la zone d'ablation, en bas du glacier. Nous pouvons la distinguer sur la figure 2 :

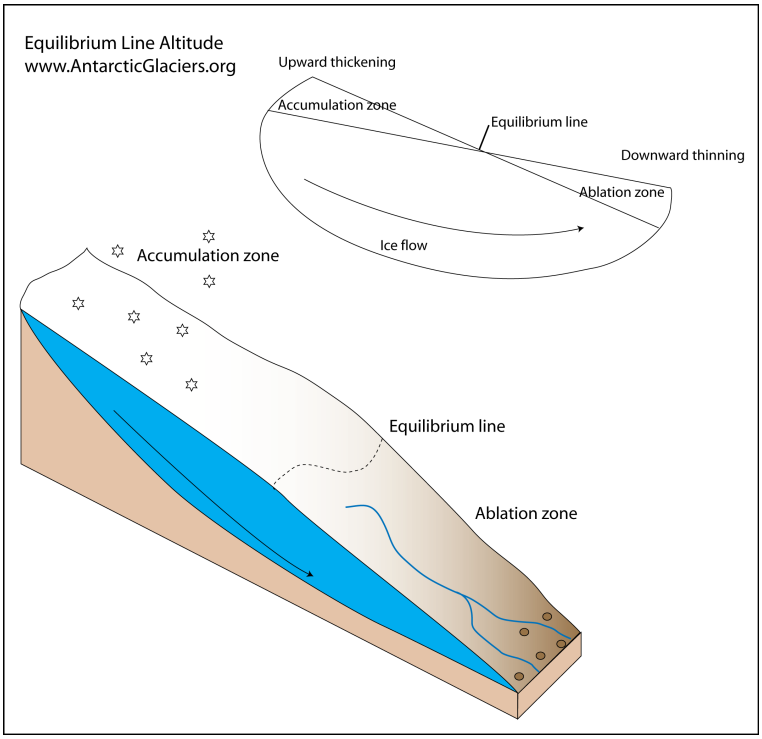

FIGURE 2 – Représentation de la ligne d'équilibre sur un glacier, déterminée par la conservation de masse M, c'est-à-dire là où la zone d'abblation et la zone d'accumulation se rencontrent

Dans le code, la conservation de masse M est calculée à partir de ∇B le gradient du bilan massique, *zELA* l'altitude de la ligne d'équilibre (ELA) (qui dépend de la latitude) et *bmax* l'accumulation maximale.

2 Simulations numériques

2.1 Les équations de Stokes

Pour créer nos simulations numériques, on va devoir déterminer des dérivées partielles. Mais celles-ci étant longues à calculer, nous utiliserons ici la **méthode des différences finies** afin d'approcher ces dérivées avec des calculs plus faciles et rapides.
Cependant, cette méthode étant compliquée, nous ne considèrerons que la définition suivante : On considère un intervalle et on le discrétise en plusieurs sous-intervalles.

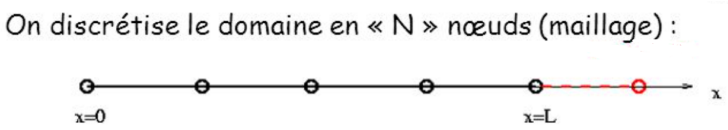

FIGURE 3 – Séparation de l'intervalle en plusieurs sous-intervalles de même taille. Afin d'obtenir une approximation de notre dérivée partielle, il suffit de prendre la valeur des deux extrêmités de chaque sous-intervalle et de faire la différence des deux.

2.2 Comprendre l'algorithme à introduire

Nous avons commencé par étudier un dossier GITHUB public [6] qui modélisait la fonte de glacier.

Il était composé de trois codes qui résolvent l'équation de diffusion linéaire avec différentes méthodes en 1D.
Tout d'abord un premier code explicite qui avait pour but de nous initier à la méthode des différences finies, en calculant simplement l'équation de diffusion :

```
while t<ttot
       qH          .= -D*diff(H)/dx          # flux
       dHdt        .=  -diff(qH)/dx          # rate of change
       H[2:end-1] .= H[2:end-1] .+ dt*dHdt
       t += dt; it += 1
   end
```

On y applique la méthode des différences finies deux fois ce qui correspond à la dérivée seconde de H par rapport à celle de x dans l'équation 17. Nous faisons cela afin de calculer dHdt que nous ajoutons à H qui nous permet de calculer l'équation de diffusion. Ce code s'exécute au total *ttot* fois.

On a également un deuxième code qui rajoute un résiduel *ResH*, une solution itérative ainsi qu'une tolérance linéaire *tol*, forçant le programme à s'arrêter lorsque cette tolérance est atteinte.

```
while t<ttot
        iter = 0; err = 2*tol
        # Picard-type iteration
        while err>tol && iter<itMax
            qH         .= -D*diff(H)/dx              # flux
            ResH       .= -(H[2:end-1]-Hold[2:end-1])/dt -diff(qH)/dx
            dHdtau     .= ResH                       # rate of change
            H[2:end-1] .= H[2:end-1] + dtau*dHdtau
            iter += 1; err = norm(ResH)/length(ResH)
        end
        ittot += iter; it += 1; t += dt
        Hold .= H
    end
```

Ce *err* se calcule à chaque itération grâce au résiduel *ResH*. Ce résiduel est l'erreur générée par les simplifications mathématiques que l'on a créé avec la méthode des différences finies (celui-ci aurait été égal à 0 si nous avions calculé exactement nos dérivées partielles). L'erreur est déterminée avec la norme et la longueur de *ResH*. Une fois que *err* est inférieur à notre tolérance, nous sortons de la boucle et donc du code.

Puis, un troisième code semblable au code implicite mais qui introduit en plus une variable *damp* itérative qui permet au programme de converger plus rapidement.

```
while t<ttot
        iter = 0; err = 2*tol
        # Pseudo-transient iteration
        while err>tol && iter<itMax
            qH         .= -D*diff(H)/dx              # flux
            ResH       .= -(H[2:end-1]-Hold[2:end-1])/dt -diff(qH)/dx
            dHdtau     .= ResH + damp*dHdtau         # damped rate of change
            H[2:end-1] .= H[2:end-1] + dtau*dHdtau
            iter += 1; err = norm(ResH)/length(ResH)
        end
        ittot += iter; it += 1; t += dt
        Hold .= H
    end
```

Ce *damp* est une valeur déterminée par l'utilisateur mais doit être compris entre 0 et 1. Plus ce damp est bas, plus le nombre d'itérations sera grand. Cependant, il ne faut pas prendre un damp trop grand car avec trop peu d'itérations, on risque de ne pas avoir un modèle satisfaisant (car il serait trop imprécis).

Ces trois codes sont linéaires, car le coefficient de diffusion D est considéré comme une constante. De plus, ils considéraient le glacier comme étant une courbe parfaitement gaussienne pour ne pas avoir à extraire des données d'un autre fichier pour modéliser ce glacier.

On avait enfin un quatrième code qui résolvait le problème de l'écoulement des glaciers en 2D à l'aide du SIA. Il inclut des itérations non linéaires, c'est-à-dire que ce coefficient de diffusion D variait à chaque point et à chaque itération de la boucle, ainsi que les conditions initiales provenant des données.
Notre but était donc de mieux comprendre le fonctionnement de ce code en 2D pour écrire à notre tour un code 1D non linéaire qui résout le problème de SIA.

2.3 Code 1D non linéaire proposé

En s'inspirant de ces codes, nous avons créé notre propre code non linéaire en 1D, qui respecte l'équation (17). L'idée était de reprendre un code similaire au code non linéaire 2D et de ne conserver qu'une seule dimension, c'est-à-dire toutes les variables dépendantes de x. Les composantes dépendantes de y n'ont donc pas été prise en compte. La boucle principale s'effectue de la manière suivante :

```
while err>tolnl && iter<itMax # On sort lorsque l'erreur est en
dessous de la tolérance  (ou trop d'itérations)
        Err         .= H
        M           .= min.(grad_b.*(S .- z_ELA), b_max)
        dSdx        .= diff(S)/dx
        D           .= a*av(H).^(npow+2) .*dSdx.^(npow-1)
        qH          .= .-av(D).*diff(S[1:end-1])/dx
        ResH        .= .-(diff(qH)/dx .+ inn(M[1:end-1]))
        dtau        .= dtausc*min.(10.0, cfl./(epsi .+ av(D[2:end])))
        dHdt        .= ResH + damp.*dHdt
        H[2:end-2]  .= max.(0.0,H[2:end-2] .+ dtau.*dHdt)
        H[Mask.==0] .= 0.0
        S           .= B .+ H
```

Nous avons recréé l'équation (17) en calculant les dérivées partielles selon x avec la méthode des différences finies. La conservation de masse M est calculé grâce à plusieurs variables qu'on prend dans les données et qu'on réévalue grâce à une fonction intermédiaire. Elle nous permet de recalculer l'accumulation de neige à chaque itération, avec une augmentation minimum de b_max (qu'on a considéré à 0.15m par an). La fonction utilisée a été retrouvé dans le GitHUB proposée et est la suivante :

```
function mass_balance_constants(xc, yc)
 # Permet de récupérer toutes les valeurs nécessaires pour l'ablation
    b_max    = 0.15            # max. Mass balance rate
    lat_min, lat_max = 60, 80
    Xc, Yc   = [Float32(x) for x=xc,y=yc], [Float32(y) for x=xc,y=yc]
    Yc2      = Yc .- minimum(Yc); Yc2 .= Yc2/maximum(Yc2)
    grad_b   = (1.3517 .- 0.014158.*(lat_min.+Yc2*(lat_max-lat_min)))./100.0.*0.91
    z_ELA    = 1300.0 .- Yc2*300.0
    return grad_b, z_ELA, b_max
end
```

Puis à partir de là, nous avons fait en sorte de n'obtenir qu'un tableau en 1D afin qu'on puisse l'utiliser dans notre code. Puis nous calculons un premier dSdx à l'aide des différences finies, qui nous servira ensuite pour calculer le coefficient de diffusion D qu'on a introduit à l'équation (18). Ensuite on reproduit l'équation (17) en deux lignes, en effectuant deux fois les différences finies. On calcule le flux qH puis le résidu *ResH* en introduisant le M précédant. Une fois que cela est fait, on calcule *dHdt* en introduisant le damp itérative pour converger plus rapidement vers une solution. Une fois que cela est fait, nous réatribuons un **Mask** afin que chaque point considéré soit au minimum au niveau de la mer, et non en dessous, puis nous remettons à jour la surface S. On introduit aussi une variable *cfl* qui est un coefficient suffisant pour assurer que notre modèle converge vers une solution cohérente.

Nous avons introduit plusieurs tableaux : le coefficient de diffusion D, le flux *qH*, le résidu de l'équation ResH ainsi que les surfaces B, H et S, dont la taille varie en fonction des calculs qu'on leur applique. Par exemple, les méthodes des différences finies sur un tableau de taille *nx* renvoie un autre tableau de taille *nx-1* (car on applique *nx-1* opérations sur *nx* valeurs).

Ce code remet à jour toutes les valeurs jusqu'à ce que l'erreur déterminée descende en dessous d'une tolérance prédéfinie. On introduit également un pas d'itération maximal pour éviter que le programme ne s'arrête jamais.

A chaque tour de boucle, nous réinitialisons le coefficient D, la conservation de masse M et la dérivée itérative *dtau*. Puis nous calculons les dérivées partielles *dSdx*, *qH* et *ResH* en effectuant les différences finies. Toutes ces valeurs nous permettent de mettre à jour la dérivée de temps *dHdt* ainsi que l'épaisseur de la glace H et enfin la surface S. Nous appliquons également un masque afin de ne pas atteindre le niveau de la mer et nous gardons en mémoire l'épaisseur H afin d'en déterminer l'erreur.

Afin de sortir de cette boucle, nous avons introduit une **solution itérative**, semblable au code implicite décrit précédemment. Nous initialisons une variable *err* et nous restons dans la boucle tant que cette erreur est supérieure à la tolérance non linéaire définie (car lorsque celle-ci sera atteinte, les variations qui suivent n'auront plus d'effet sur notre modèle). Nous avons décidé de recalculer cette erreur tous les *nout* fois, qu'on a initialisé à **cinquante**, afin de ne pas faire trop de calculs. Au bout de cinquante itérations, nous actualisons la valeur de la variable *err* avec le code suivant :

```
if mod(iter, nout)==0
            Err .= Err .- H
            err = norm(Err)/length(Err)
    if isnan(err)
             error("""NaNs encountered.  Try a combination of:
            decreasing ‘damp‘ and/or ‘dtausc‘, more smoothing steps""")
            end
end
```

Dans cette condition, on recalcule notre *Err* en la comparant avec la hauteur de glace H actuelle. Puis on prend sa norme et on la divise par sa longueur pour obtenir notre *err*. La boucle s'arrête lorsque l'erreur est plus petite que la tolérance imposée.
On a également une condition qui vérifie que *err* est définie, c'est-à-dire qu'on vérifie que cette erreur ne tend pas vers l'infini. Cela peut arriver lorsque la longueur de *Err* est nulle ou bien lorsque sa norme tend vers l'infini (ce qui peut arriver lorsqu'une des valeurs dans la boucle est mal définie). Si cela arrive, on sort du code en envoyant une erreur, signalant les étapes à effectuer pour eventuellement régler le problème.

Maintenant que l'on a un modèle résolvant l'équation non linéaire H en 1D, il nous faut à présent vérifier que ce modèle est juste. Pour cela, nous avons extrait des données du Groenland datant de 2020.

3 Résultats

3.1 Extraction les données

Pour s'assurer que le code compilait, nous avons simplifié certaines variables du problème. La conservation de masse était constante et les surfaces H, B et S étaient modélisées par des fonctions mathématiques. Une fois cela fait, nous avons extrait des valeurs d'un autre fichier contenant toutes les valeurs du Groenland **BedMachineGreenland_96_176.jld**.

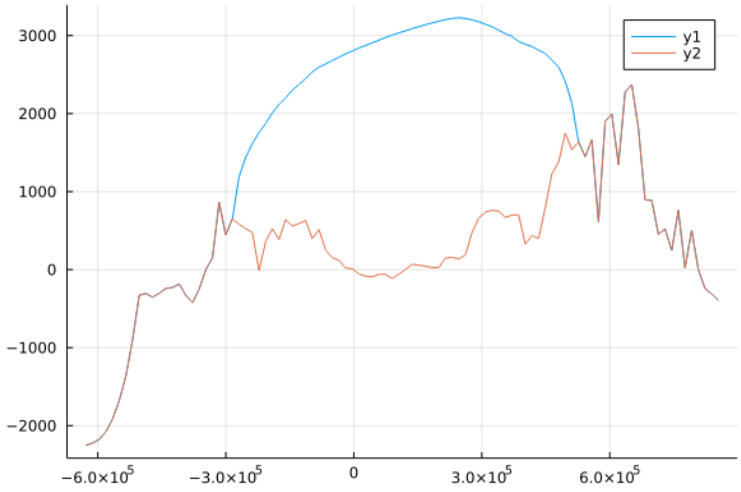

FIGURE 4 – Topologie du Groenland d'après les données récupérées. La courbe *y2* représente la hauteur du lit rocheux B et *y1* la topologie de la glace S.

Pour l'extraction des données nécessaires, il a fallu donc charger le fichier **BedMachineGreenland_96_176.jld**. Le fichier chargé est stocké dans une variable que l'on nommera *var*, qui est un objet donc il va falloir extraire la valeur pour chaque clé :

```
# Initial condition
# On charge notre fichier GeoData...
# Puis on extrait les valeurs dans les variables...
# On extrait toutes les valeurs nécessaires depuis le fichier helpers.jl...
var = load("BedMachineGreenland_96_176.jld") # ultra low res data
Zbed = var["Zbed"]
xc = var["xc"]
yc = var["yc]
Hice = var["Hice"]
Mask1 = var["Mask"]
```

On obtient un objet *Zbed* et *Hice* qui sont plus exactement de type GeoData. Après une recherche dans la documentation de GeoData Julia, nous n'avons pas réussi à trouver comment extraire le tableau (ou la matrice) contenu dans ces objets.

Après quelques recherches sur des forums (notamment stackoverflow), puis quelques *print()* dans la console, nous en avons déduis qu'il fallait faire Objet._clé_, ce qui est d'ailleurs une pratique assez courante que ce soit en Python ou même en Javascript par exemple.

Ainsi on peut récupérer les informations concernant le niveau de la terre et l'épaisseur de la glace. Pour notre étude nous avons décidé de récupérer les valeurs à partir de la 80ème colonne de la matrice :

```
B1 = ZBed.data # Niveau de la terre
H1 = Hice.data # Epaisseur de la glace

# On ne récupère qu'une colonne qu'on étudiera ensuite
# On choisit arbitrairement la colonne 80
B = B1[: , 80]
H = H1[: , 80]
```

Une fois nos données extraite, il était nécéssaire de faire apparaître notre masque pour le niveau de la mer, que l'on a extrait dans la variable *Mask1*, puis de la même manière que les valeurs *Zbed* et *Hice*, on récupère les données à l'aide de la clé data :

```
Mask1 = var["Mask"]
display(Mask1.data)
# Booléen qui nous permettra d'appliquer le masque pour le niveau de la mer
Mask = Mask1.data[: , 80]
```

Nos données extraites, on peut désormais passer au traitement et à l'utilisation de notre programme.

3.2 Afficher un premier graphique

Nous avons ensuite réajusté l'ablation M ainsi que les coefficients itératifs damp afin d'avoir une vitesse de convergence adaptée au problème. Nous affichons donc le résultat obtenu à l'aide d'un plot.

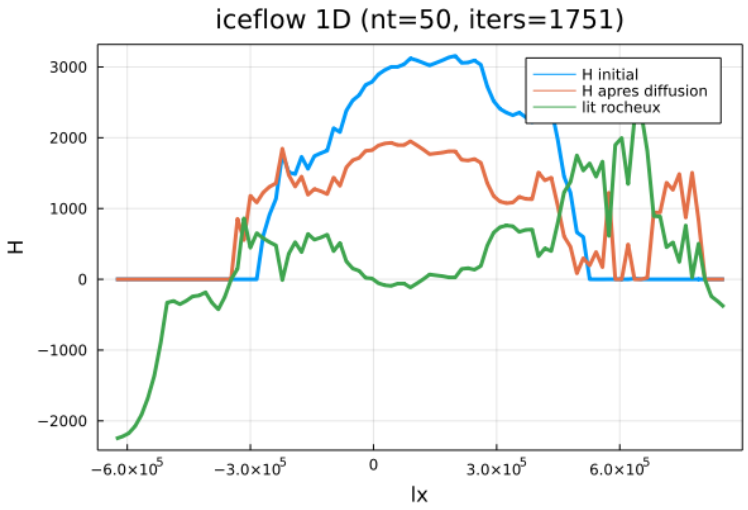

FIGURE 5 – Evolution du niveau de glace en fonction du temps. La courbe bleu représente la hauteur H au début, la courbe orange donne la hauteur H une fois la simulation terminée et la courbe verte montre le lit rocheux B.

3.3 Chercher une situation proche du code 2D

Il nous faut à présent déterminer la ligne où les variations verticales soient les plus faibles, afin d'avoir le modèle le plus proche possible de la réalité adapté a notre code.
Pour cela, nous avons cherché dans les données que nous avions sur le Groenland la ligne "horizontale" qui nous permettait d'avoir des résultats suffisamment justes pour pouvoir appliquer notre code. Plus précisément, il fallait que la ligne choisie soit une ligne qui permette de négliger les vitesses verticales. Pour choisir judicieusement cette ligne, nous avons comparé les différentes données de vitesses verticales Vy que nous avions à notre disposition en faisant une somme entre deux lignes consécutives, voici le code qui nous a permis de trouver la ligne correspondante :

```
V = sqrt.(Vy.^2)
s=zeros(95)
for i = 1:95
    t=V[i,:]
    s[i]=sum(t)
end

for i = 2:94
    s1[i-1] = s[i-1] + s[i+1]
end
display(plot(1:95, s))
display(plot(1:94, s1))
```

A l'aide de ce programme, nous pouvons obtenir une courbe donnant les vitesses verticales sur chaque ligne :

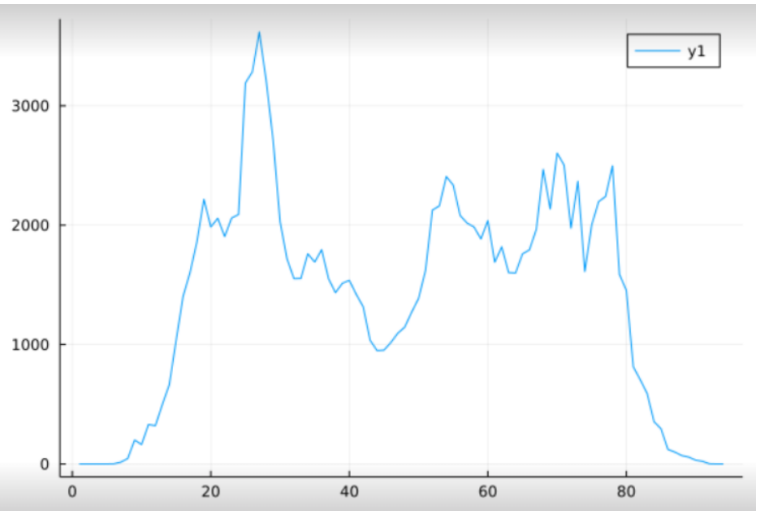

FIGURE 6 – Graphe représentant la somme des variations verticales par rapport aux lignes du Groenland. On ne considère pas les bordures du Groenland car on veut prendre la ligne qui correspond au rapport entre la vitesse horizontale et verticale la plus élevée, ceci est situé au centre du Groenland

En regardant au centre, nous observons que les vitesses verticales sont les plus faibles vers la ligne 43. Nous avons alors choisi cette ligne pour que notre modèle soit le plus cohérent avec notre modèle 2D.

En exécutant le code avec cette ligne, nous avons alors le résultat suivant :

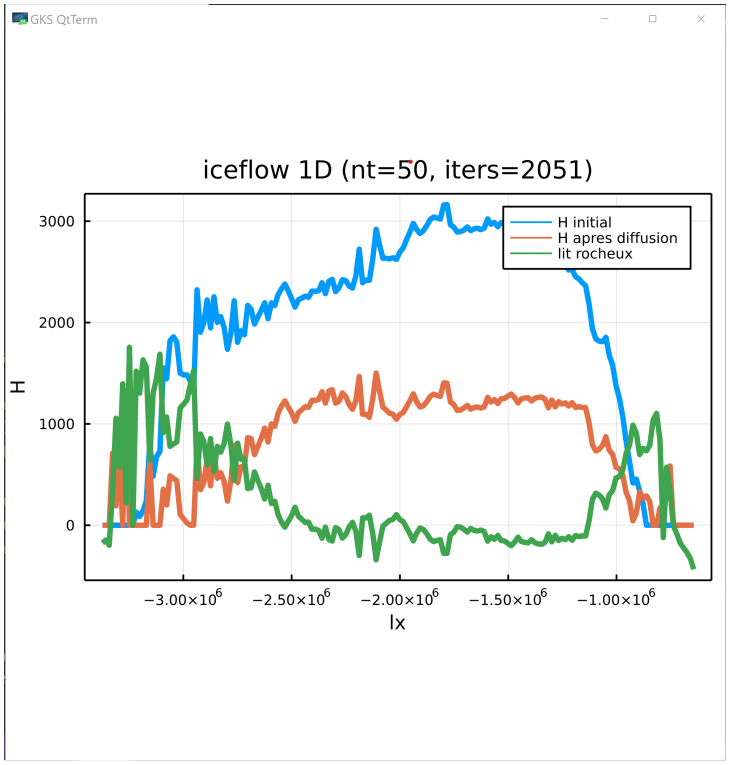

FIGURE 7 – Topologie du Groenland en appliquant la ligne 43. La courbe bleue représente la hauteur H au début, la courbe orange donne la hauteur H une fois la simulation appliquée pour la ligne 43 et la courbe verte montre le lit rocheux B.

3.4 Interpréter notre simulation

En observant notre schéma, nous pouvons à présent simuler l'évolution de la glace du Groenland et créer notre propre scénario. En conservant nos valeurs, nous pouvons observer une fonte de glace allant de 0 à 1500m de variation. Avec un simple calcul de moyenne, nous obtenons une variation moyenne de 1091m. Notre hauteur de base moyenne étant de 1907, cela nous donne une variation de 57% de la hauteur. A chaque itération, nous réattribuons des valeurs à tous les tableaux en fonction de a, qui est la viscosité de la glace **calculée avec un pas de temps annuel**. Nous en déduisons qu'une itération correspond à une année et donc que ce scénario sera atteinte au bout de 2051 ans.

Nous avons trouvé un article [7] spéculant la fonte de glace au Groenland en fonction de la hausse de CO2, car celui-ci est directement relié avec la hausse de température. Il effectue plusieurs simulations, en fonction de la variation de température considérée (variant entre 1.1°C et 6.4°C) :

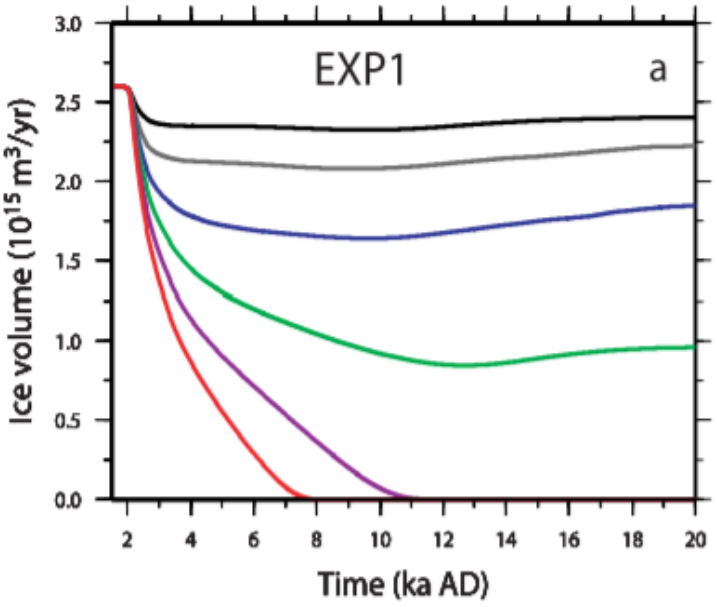

FIGURE 8 – Schéma indiquant le volume de glace du Groenload à partir de l'an 2000. Grâce à cette simulation, nous pouvons en déduire que le Groenload perdrait entre 10% et 63% de son volume, ce que l'on retrouve bien avec notre simulation (Notre simulation est proche de la courbe verte de ce schéma)

3.5 Modifier nos valeurs itératives

Etant donné que nous avons introduit des valeurs itératives comme damp, nous avons voulu savoir ce qu'on allait obtenir lorsqu'on fait varier cette valeur. Nous avons donc prit deux valeurs extrèmes 0.1 et 1 et nous avons tenté d'afficher le modèle obtenu :

```
ERROR: LoadError: NaNs encountered.  Try a combination of:
decreasing `damp` and/or `dtausc`, more smoothing steps
```

FIGURE 9 – Résultat du plot de notre code lorsque damp est à 0.1. Cela montre bien qu'avec trop peu d'itérations, notre modèle est incapable d'afficher une modélisation cohérente.

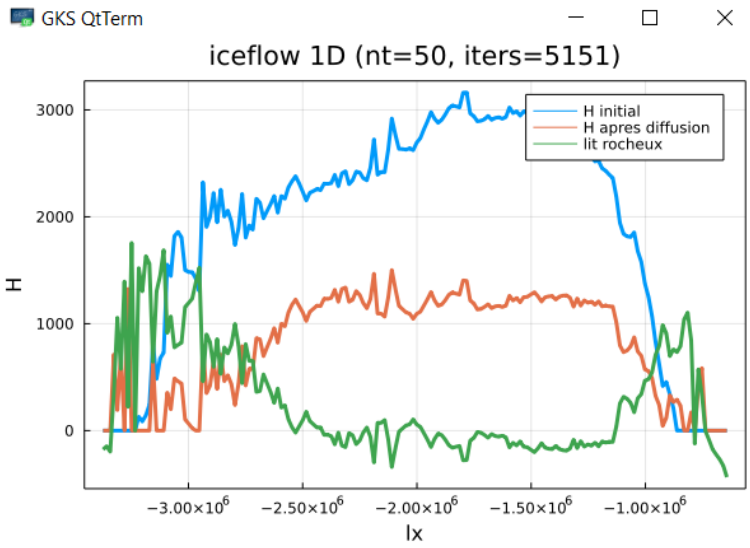

FIGURE 10 – Résultat du plot de notre code lorsque damp est à 1. On remarque que le résultat obtenu est semblable à celui présenté à la figure 7, mais on a fait un nombre d'itérations bien plus important pour parvenir à ce résultat. Cela montre bien l'intérêt d'avoir un damp permettant de réduire le temps de modélisation.

Références

[1] David Louapre. La mystérieuse équation de stokes.

[2] Thierry Menand. Introduction à la dynamique des fluides visqueux. 2020.

[3] Martina Schäfer. *MODÉLISATION DE L'ECOULEMENT DES GLACIERS TEMPÉRÉS*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2007.

[4] H. Seroussi, M. Morlighem, E. Rignot, A. Khazendar, E. Larour, and J. Mouginot. Dependence of century-scale projections of the greenland ice sheet on its thermal regime. *Journal of Glaciology*, 59(218) :1024–1034, 2013.

[5] Robin Chatelin. Méthodes numériques pour l'écoulement de stokes 3d.

[6] Ludovic Räss and Mauro. Solving differential equations in parallel with julia.

[7] Sylvie Charbit, D Paillard, and G Ramstein. Amount of co2 emissions irreversibly leading to the total melting of greenland. *Geophysical Research Letters*, 35(12) :n–an, 2008.