Diffusion non linéaire en sciences de la terre
modélisation des glaciers

16 décembre 2021

Table des matières

Résumé

Résumé du contenu du document.

1 Introduction

L'écoulement des glaciers est un processus continu, mais il est très difficile de le simuler mathématiquement. Pour cela, les modélisateurs utilisent des méthodes numériques, qui résolvent les équations en une série d'étapes. L'intérêt ici va être d'énoncer les relations générales de Stokes appliqués 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.

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. Dans le cadre de ce projet, nous allons utilisé le langage Julia pour la résolution de systèmes d'équations différentielles afin de pouvoir prédire l'écoulement glaciaire du Groenland.

2 Présentation des techniques de résolution

2.1 Pourquoi Julia ?

Julia est un langage de programmation moderne et polyvalent, crée 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 scientifique. Sa syntaxe est similaire à Python, R ou encore Matlab.

2.2 Les équations de Stokes

L'équation de Navier-Stokes

Les équations de Stokes ont comme principal objectif de décrire les mouvements des fluides. Pour décrire correctement un fluide en mouvement, 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 pression et les forces visqueuses.

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. Résoudre les équations de stokes revient à résoudre ce système :

$$\begin{cases} -div(2\mu\varepsilon(\vec{v})) + \nabla p = \rho\vec{g} \\ div(\vec{v}) = 0 \end{cases} \tag{1}$$

avec $\vec{v}$ la vitesse, ∇p le gradient de pression, $\vec{g}$ l'accélération de pesanteur,ρ la masse volumique, μ la viscosité et $\varepsilon(\vec{v})$ le taux de déformation. Néanmoins, le Groenland une calotte tellement grande de milliers de kilomètres, qu'il faut un nombre considérable de points pour résoudre ces équations. De plus du fait de la complexités des equations et des géométries de la calotte, cela est coûteux en temps de calcul.

Nous observons que les calottes sont relativement mince, soit quelques kilomètres d'épaisseur, ainsi nous pouvons négliger les variations verticales de la vitesse / les contraintes de cisaillement longitudinal. Nous en concluons, que pour minimiser le temps de calcul, nous ne sommes pas obligé d'utiliser un modèle en 3D en résolvant les équations de Stokes mais que nous pouvons simplement utilisé un modèle simplifié en 2D. Le modèle SIA intervient justement en tant qu'approximation des relations de Stokes, permettant des calculs beaucoup plus simples pour un modèle en 2D.

2.3 Shallow ice approximation (SIA)

L'équation SIA simplifiée
Détails sur l'équation SIA simplifiée

Les chercheurs assurent que la résolution du systèmes des équations de Stokes est très couteuse en temps pour des applications réalistes. C'est pour cette raison que nous allons utiliser 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 horizontal. Pour cela, nous allons utilisé le modèle SIA (shallow ice approximation). Sa validité dépend du rapport d'aspect caractéristique ζ des objets glaciaires :

$$\zeta = \frac{e}{L} \tag{2}$$

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 considère ici les [intéractions internes] (aussi appelés bridge effect), ce qui nous permet de séparer les vitesses horizontales et verticales. μ est considéré isotrope et s'écrit :

$$\mu = \frac{B}{2\varepsilon(\vec{v})^{\frac{n-1}{n}}} \tag{3}$$

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 :

$$- div(2\mu\varepsilon(\vec{v})) = \rho\vec{g}\frac{\partial S}{\partial x} \tag{4}$$

Et l'évolution de la surface est déterminée grâce à une équation hyperbolique utilisant la méthode des différences finies et 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 :

$$\frac{\partial H}{\partial t} = -\nabla H\vec{v} + Ms - Mb \tag{5}$$

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 :

$$\frac{\partial H}{\partial t} = D\frac{\partial^2 H}{\partial x^2} + M \tag{6}$$

avec D le coefficient de diffusion, H l'épaisseur du glacier et M la conservation de masse.

L'épaisseur H est donnée par l'équation suivante :

$$H(x) = S(x) + B(x) \tag{7}$$

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

Les données topographiques, l'élévation du substratum rocheux et l'épaisseur

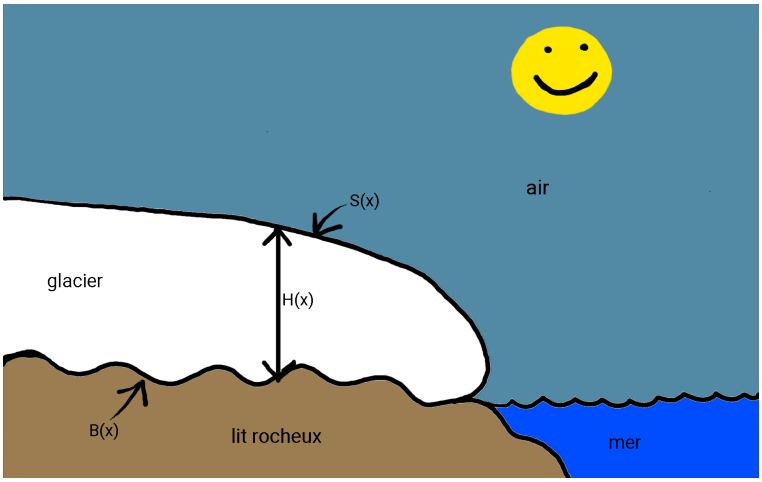

FIGURE 1 – Représentation de l'épaisseur de la glace H, la hauteur du lit rocheux B et la topologie de la glace S

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 :

$$M = accumulaiton + ablation \tag{8}$$

Ainsi la masse balance 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 ci-dessous :

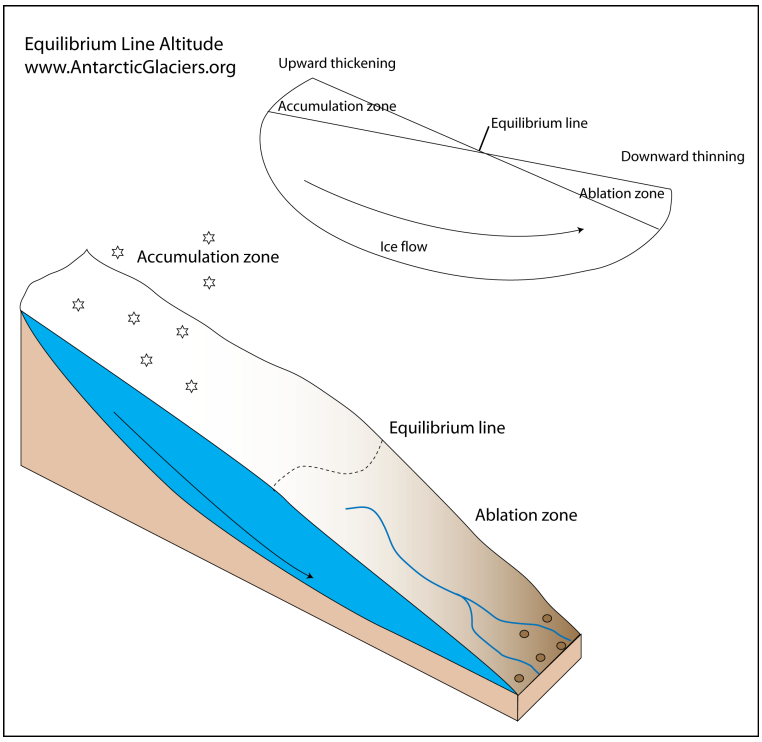

FIGURE 2 – Représentation de la ligne d'équilibre sur un glacier, déterminée par la conservation de masse M, c'est-à-dire là où la zone d'abblation et la zone d'accumulation se rencontrent

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

3 Simulations numériques

Nous avons commencé par étudier un dossier GITHUB public qui modélisait la fonte de glacier.
GitHUB du professeur

Il était composé de trois codes qui simplifiaient le problème en 1D : Un code simple qui ne s'occupe que de la fonte de la glace en fonction du temps. Un deuxième code rajoute une solution itérative ainsi qu'une tolérance de fin, forçant le programme à s'arrêter uniquement lorsque cette tolérance est atteinte. Enfin, un troisième code introduit un "damp" itératif qui permet au programme de converger plus rapidement.
Ces trois codes sont considérés linéaires, car le coefficient de diffusion D était une constante.
Un quatrième code considérait le problème en 2D et était considéré comme non linéaire, c'est-à-dire que ce coefficient D variait à chaque point et à chaque itération du problème. Notre but était donc de créer un code non linéaire qui simplifie le problème en 1D, à l'aide de tous ces codes.
En s'inspirant de ces codes, nous avons créé un code dont la boucle 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 (**??**) en calculant les dérivées partielles selon x avec la méthode des différences finies. Cela consiste à calculer la différence entre deux valeurs consécutives.
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.
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 [] 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.
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)
end
```

La boucle s'arrête lorsque l'erreur est plus petite que la tolérance imposée.

4 Résultats