Université Claude Bernard Lyon 1
I.S.T.I.L. , Ecole Polytechnique Universitaire de Lyon
Rapport de stage

Maître de stage : Mr Pierre Dreyfuss (Laboratoire J.A. Dieudonné - Nice)
Tuteur universitaire : Mr Daniel Leroux (ISTIL EPU - Lyon)

La méthode BFN, une nouvelle méthode d'assimilation de données pour le code NEMO

Baptiste CHOUZET

Nice, le 25 août 2010

1 Remerciements

Par le présent, je tiens à vous remercier Mr Dreyfuss de m'avoir accepté comme stagiaire dans votre équipe au sein du laboratoire J.A. Dieudonnée de l'université de Nice.

Mon stage a été très formateur et les activités auxquelles j'ai pris part, du 17 mai au 13 août, m'ont permis de consolider mes connaissances théoriques. Cette expérience aura été très enrichissante, dans un domaine qui m'était inconnu jusqu'à maintenant.

Je souhaite également transmettre par votre intermédiaire mes remerciements à toutes les personnes qui m'ont encadré durant ces trois mois. Grâce à leur expertise, et aussi à leur gentillesse, j'ai découvert la diversité des échanges au sein d'un laboratoire.

Table des matières

2 Laboratoire

Le laboratoire de mathématiques J.A. Dieudonné de la Faculté des Sciences de Nice fut créé en 1964 sous le décanat du premier doyen, le Professeur Jean Alexandre Dieudonné. Il est issu de l'Ecole Normal Supérieure et obtint son doctorat en 1931

J. A. Dieudonné fut un des deux principaux rédacteurs des textes de la série Bourbaki. Il commença sa carrière de mathématicien par des travaux sur l'analyse polynomiale. Il travailla sur une grande variété de domaines mathématiques incluant la topologie générale, la topologie des espaces vectoriels, la géometrie algébrique, la théorie des invariants et les groupes classiques.

Actuellement, le laboratoire se situe au coeur de la cité des sciences Valrose de l'Université Nice Sophia-Antipolis.

3 Introduction

Grâce, notamment, aux puissants moyens de calculs maintenant disponibles, la modélisation en géophysique, en vue de la prévision a connu d'important développement ces dernières décennies. Les fluides géophysiques : l'air, l'eau atmosphérique, océanique ou terrestre sont régis par les équations de la mécanique des fluides : conservation de masse, de l'energie, loi de comportement, toutefois certaines spécificitées doivent être prises en compte.

Pour comprendre la nécessité des méthodes d'assimilation de données, il est important de comprendre la complexité du domaine d'étude, qui sera dans notre cas la modélisation de l'océan par le code NEMO.

Unicité d'une situation : tout épisode géophysique est unique, une même situation ne se produit pas ou ne peut être dupliquée, ce qui signifie que les sciences de l'environnement ne sont pas *stricto sensu*, c'est à des sciences expérimentales où une hypothèse est validée par une répétition d'expériences. Un modèle géophysique devra donc être testé et validé avec des données associées à des épisodes tous différents.

Non linéarité : Les processus géophysiques sont fondamentalement non-linéaires. Ceci induit des interactions et des cascades d'énergie entre les différentes échelles en temps et en espace. Une difficulté majeure provient du fait que les phènoménes inférieures à la troncature, due aux discrétisations, peuvent correspondre à de très importants flux d'énergie dont il faudra tenir compte dans la modélisation.

Fermeture : Les seules équations de la dynamique des fluides ne sont pas suffisantes pour faire une prévision, il faut en outre une condition initiale et des conditions aux limites. Dans la plupart des cas, les fluides géophysiques n'ont pas de frontières naturelles, de même aucune condition initiale, de type solution stationnaire, ne s'impose naturellement.

L'assimilation de données est l'ensemble des techniques qui permettent de combiner, de façon optimale (en un sens à définir), l'information mathématique contenue dans les équations et l'information physique provenant des observations en vue de reconstituer l'état de l'écoulement.

Jacques Blum
Laboratoire J.A. Dieudonné, Nice

On peut donc tout de suite imaginer les enjeux de la prévision du comportement de l'océan à court terme (suivi de nappe de déchets, pêche,déplacement des plateformes pétroliéres, etc.), à moyen terme (prévisions saisonnières d'épisodes tel que l'ouragan catherina, etc.) ou encore à long terme (changement climatique).

L'objectif me concernant a été d'installer le code NEMO sous une configuration particulière (GYRE), d'apporter les modifications de code nécessaire à l'application de la méthode, puis de tester les différentes performances du code sous sa version parallèle.

Première partie

Présentation du problème

4 Méthode générale de discrétisation dans NEMO

4.1 Présentation des équations

Les équations sont discrétisées sur une grille de type C (Arakawa 1972) par des schémas spatiaux d'ordre deux. Les variables considérées sont la température T, la salinité S (qui sont ici données), la vitesse U en 3 dimensions et la hauteur d'eau H.
Le schéma temporel est un schéma de saute-mouton (Leap Frog) associé à un filtre d'Asselin (1972) à l'exception des termes de diffusion qui sont traités avec un schéma d'Euler explicite pour l'horizontale et un schéma d'Euler-implicite pour la verticale.
L'organigramme représente les étapes de la résolution des équations primitives dans NEMO à l'intérieur de la boucle temps sous notre configuration.

On écrit ci-dessous le récapitulatif des équations continues que NEMO utilise :

$$\frac{\partial T}{\partial t} = -\nabla.(TU) + D^T + F^T$$

$$\frac{\partial S}{\partial t} = -\nabla.(SU) + D^S + F^S$$

$$\rho = \rho(T, S, p)$$

$$\frac{\partial U_h}{\partial t} = -\left[(\nabla \wedge U) \wedge U + \frac{1}{2}\nabla(\mid U \mid^2)\right]_h - f.z \wedge U_h - \frac{1}{\rho_0}\nabla_h p + D^U + F^U$$

$$\frac{\partial p}{\partial z} = -\rho g$$

$$\nabla.U = 0$$

$$\frac{\partial \eta}{\partial t} = -div_h((H + \eta)\overline{U_h}) + [P - E]$$

4.2 Présentation de l'algorithme de résolution

Initialisation

Dans un premier temps, l'algorithme définit le domaine, les conditions aux limites et fixe les paramètres physiques nécessaires au modèle. Nous allons voir les principales étapes de cette initialisation qui s'effectuent dans la subroutine *opa_init* qui se situe dans le fichier *opa.f90*.

lib_mpp.f90 : initialise la bibliothèque de calcul parallèle.

model.f90 : Mise en place des équations liés à notre configuration.

phys_cst.f90 : initialise tout les paramètres physiques du domaine.

domain.f90 : Initialise les paramètres du domaine, en appelant les différentes subroutines :
- *dom_nam* : Lit les différents paramètres physiques liés au domaine
- *dom_hgr* : Initialise le maillage horizontal
- *dom_zgr* : Initialise le maillage vertical et définit le profil

dom_cfg.f90 : Initialise les paramètres physiques liés à la configuration.

sbc_mod.f90 : Initialise la condition aux limites de surface.

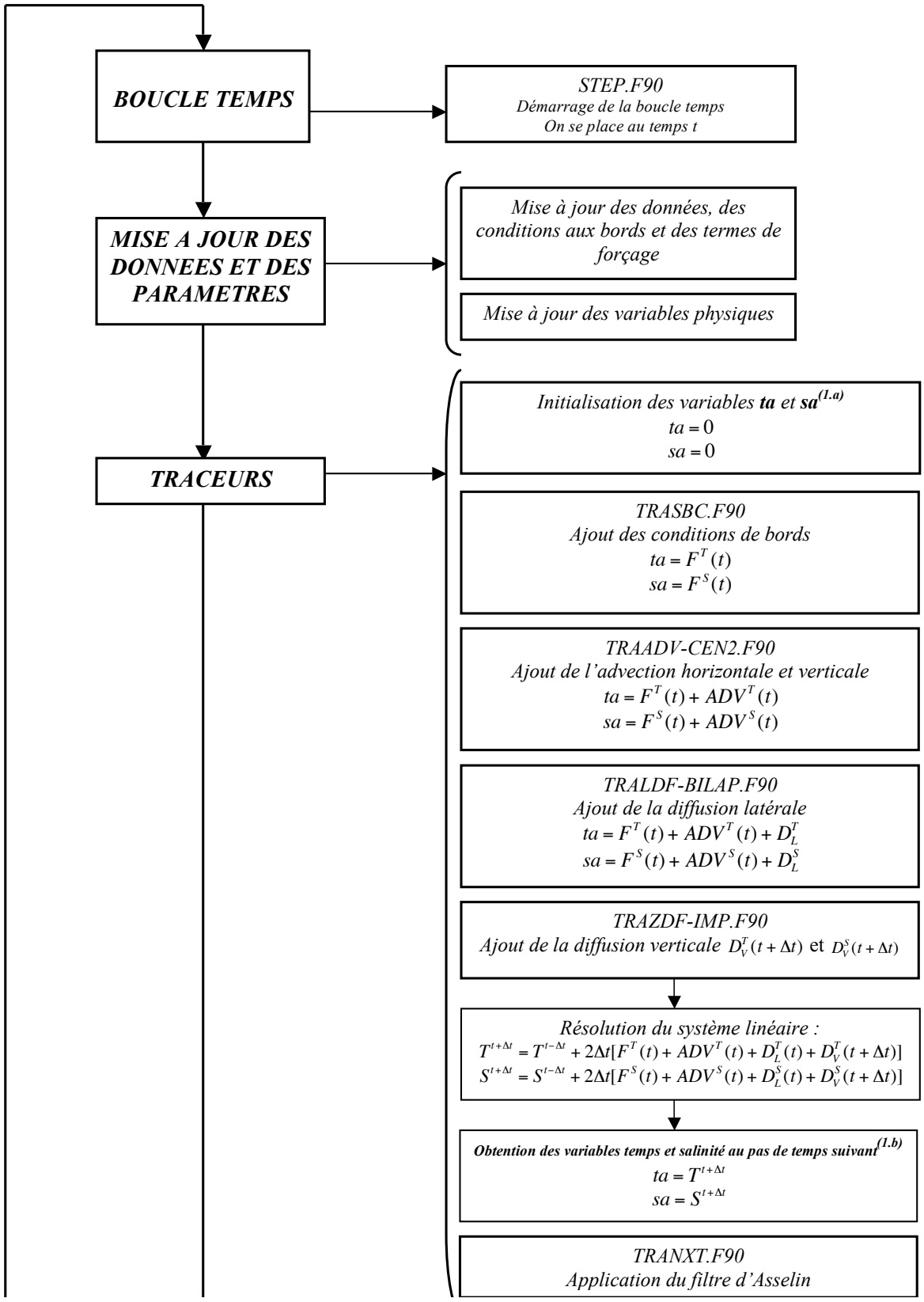
ldf_dyn_init.f90 et zdfini.f90 : Initialise les paramètres dynamiques physiques horizontal et vertical.

istate.f90 : Mise en place de la condition initiale par l'appel de la fonction *istate_GYRE*. Les composantes sur u et v sont initialisées à 0, le forçage du vent ainsi que la température et la salinité sont définies par l'intermédiaire de fonctions analytiques. La hauteur d'eau est définie constante à t=0.

Résolution

La boucle de résolution des équations régissant le domaine passe par plusieurs étapes de calcul. On pourra voir sur l'organigramme, en page suivante, ces différentes étapes.

Les différents indices de l'organigramme renvoient aux remarques suivantes :
- (1.a) : ta et sa représentent tout d'abord les contributions du second membre de l'équation d'évolution en temps de la température et de la salinité.
- (1.b) : *ta* et *sa* représentent, à la fin des calculs sur les traceurs, la température et la salinité au pas de temps suivant.
- (2.a) : *ua* et *va* représentent les contributions du second membre de l'équation d'évolution en temps de la vitesse horizontale.
- (2.b) : *ua* et *va* représentent, à la fin des calculs sur la dynamique, la vitesse horizontale au pas de temps suivant.


BOUCLE TEMPS
STEP.F90
Démarrage de la boucle temps
On se place au temps t
MISE A JOUR DES DONNEES ET DES PARAMETRES
Mise à jour des données, des conditions aux bords et des termes de forçage
Mise à jour des variables physiques
TRACEURS
Initialisation des variables ta et sa(1.a)
ta = 0
sa = 0
TRASBC.F90
Ajout des conditions de bords
ta = F^T(t)
sa = F^S(t)
TRAADV-CEN2.F90
Ajout de l'advection horizontale et verticale
ta = F^T(t) + ADV^T(t)
sa = F^S(t) + ADV^S(t)
TRALDF-BILAP.F90
Ajout de la diffusion latérale
ta = F^T(t) + ADV^T(t) + D_L^T
sa = F^S(t) + ADV^S(t) + D_L^S
TRAZDF-IMP.F90
Ajout de la diffusion verticale D_V^T(t+Δt) et D_V^S(t+Δt)
Résolution du système linéaire :
T^{t+Δt} = T^{t-Δt} + 2Δt[F^T(t) + ADV^T(t) + D_L^T(t) + D_V^T(t+Δt)]
S^{t+Δt} = S^{t-Δt} + 2Δt[F^S(t) + ADV^S(t) + D_L^S(t) + D_V^S(t+Δt)]
Obtention des variables temps et salinité au pas de temps suivant(1.b)
ta = T^{t+Δt}
sa = S^{t+Δt}
TRANXT.F90
Application du filtre d'Asselin

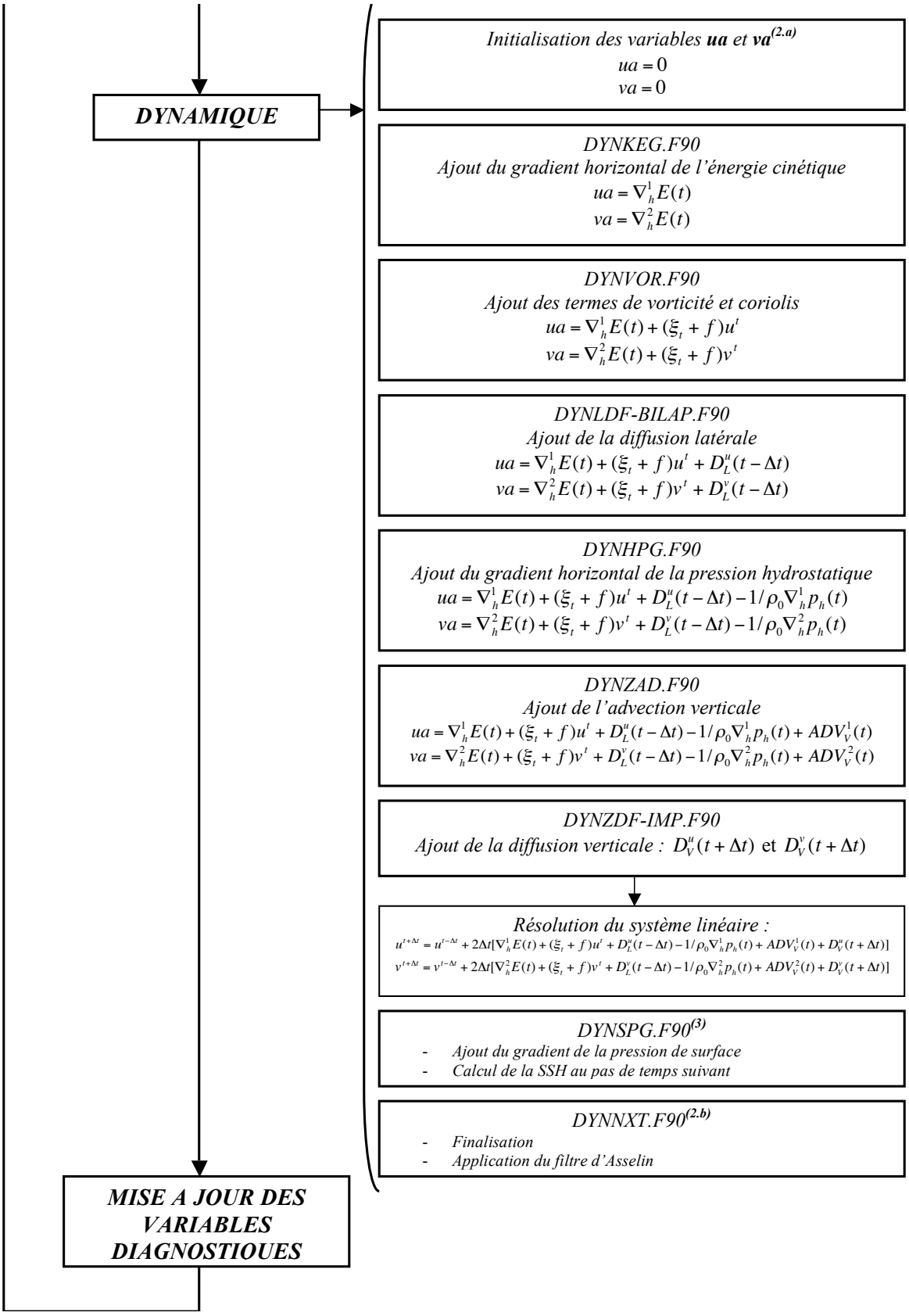


FIGURE 1 – *Organigramme de la résolution des équations primitives dans NEMO*

4.3 Définition du système de coordonnées

Dans le code NEMO (dans notre configuration GYRE), les coordonnées géographiques ont été choisies. Elles sont définies de la manière suivante :

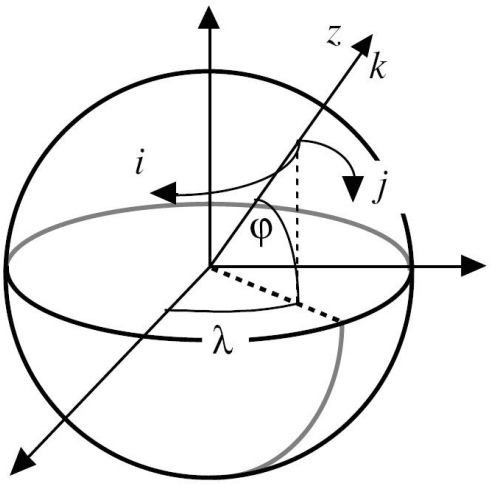


FIGURE 2 – *Système de coordonnées géographiques*

Le système de coordonnées curvilignes (i,j,k) va nous permettre de nous situer à l'intérieur de notre maillage,et le système (λ, ϕ, z) va définir :

- $\phi(i,j) \in \left[\frac{-\pi}{2}; \frac{\pi}{2}\right]$ nous donne la latitude.
- $\lambda(i,j) \in [-\pi; \pi]$ nous donne la longitude.
- z = a + z(k) où a représente le rayon de a terre et z(k) représente une hauteur d'eau référencée.

4.4 Définition du domaine

Pour définir notre maillage, plusieurs paramètres vont devoir être fixés. Nous allons voir ici quels paramètres sont proposés et quelle configuration a été choisie pour notre étude.

Taille du domaine On va pouvoir fixer la taille de notre domaine grâce aux paramètres *jpiglo, jpjglo et jpk* dans le module *par_oce.F90*. De plus on pourra fixer la taille des sous domaines de chaque processeur, quand le code fonctionne sous sa version parallèle, grâce aux variables *jpi et jpj*.

Grille horizontale jphgr mesh=5 correspond à notre configuration GYRE. Une rotation du domaine est effectuée pour maximiser les différents courants relatifs au domaine dans notre configuration. La construction du domaine s'opère en deux phases, la première définit les longitudes et latitudes extrèmes, puis on effectue une rotation du domaine. Voir l'annexe 1 pour voir la représentation du domaine.

Grille verticale Dans le code NEMO plusieurs types de coordonnées sur z sont envisageable :

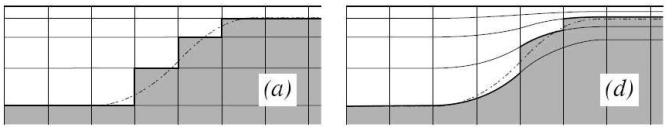


FIGURE 3 – *Différents types de coordonnées sur z*

On va pouvoir fixer le type de coordonnées voulu, soit ici des coordonnées sur z régulière (de type (a)), avec la variable *ln zco=true* dans la namelist. Pour completer le maillage vertical, il est necéssaire de fixer 3 paramètres :

- Le profil du fond, donnée en mètre.
- Le nombre de niveau sur z du modèle. La discrétisation verticale est représenté en Annexe 2.
- La détermination des coefficients de transformation du domaine, pour passer d'une partie de sphère à un domaine plat.

4.5 Définition des conditions limites

Conditions de bords On choisi des conditions aux limites fermées. Pour cela on fixe la variable *jperio=0*. On obtient alors des murs solides, les premières et dernières lignes et colonnes sont fixées à zéros.

Profil de fond Pour notre configuration on va choisir un fond plat. Pour cela on fixe la variable *ntopo=0* dans la namelist.

Conditions de surface A la surface de l'océan un forcage est exercé par le vent, cette condition va être appliqué sur la surface (i.e. à z=1) sous la forme :

$$(\frac{\partial T}{\partial k}\frac{A^{vm}}{e_3})_{z=1} = \frac{1}{\rho_0}\tau \qquad \text{avec} \qquad \tau = (\tau_u, \tau_v)$$

5 Les méthodes d'assimilation de données

L'assimilation de données se situe dans la catégorie des "Problèmes Inverses". En météorologie et en océanographie, l'assimilation de données recouvre deux types de problèmes : d'une part l'identification de l'état initial ; dans ce cas toutes les informations (modèle, données) sont réunies en vue de déterminer le meilleur état initial correspondant à ces informations. D'autre part, il peut aussi s'agir d'identifier certains paramètres mal connus du modèle (liés aux paramétrisations des petites échelles, aux conditions aux limites, etc.)

On distingue en assimilation de données deux grandes classes de méthodes :

Les méthode de filtrage ou séquentielles : elles sont basées sur la théorie de l'estimation statistique, dont le principal représentant est le filtre de Kalmann. Les données sont assimilées au fur et à mesure. A chaque pas de temps où l'on dispose de mesures, on estime le "meilleur" compromis entre l'état prévu par le pas de temps précédent et les observations effectuées.

Les méthodes variationnelles : elles utilisent toutes les observations disponibles pendant un laps de temps fixé (appelée fenêtre temporelle, qui varie généralement de 10 à 30 jours en océanographie) pour estimer un état inital (ie au début de la fenêtre temporelle) qui soit le "meilleur" compromis entre toutes les observations et le résultat de la prévision précédente. L'idée est de calculer une fonction coût mesurant, au sens des moindres carées, l'écart entre les observations et leurs équivalents donnés par le modèle, puis de chercher l'état optimial minimisant cette fonction coût. Ce qui nous amène à résoudre un problème de minimisation. Ceci permet de ne pas rechercher (ou approcher) la matrice de gain K mais de chercher directement, par une méthode de descente, l'optimum de la fonction coût considérée.

Nous allons alors voir la méthode BFN (*the Back and Forth Nudging method*), qui peut être vu comme une méthode d'assimilation de données variationnelles (Voir [1]).

6 La méthode BFN : the Back and Forth Nudging algorithm

6.1 Nudging direct

On s'assure que les équations du modèles ont été discrétisées (différences finies, éléments fins ou discrétisation spectrale). Ainsi, le modèle (non discrétisé en temps) est soumis aux équations de la dynamique de la forme :

$$\begin{cases} \frac{dX}{dt} = F(X) \quad 0 < t < T \\ X(0) = x_0 \end{cases} \tag{1}$$

On va alors définir un opérateur d'observation C, de façon à pouvoir comparer X_{obs} et C(X(t)). Cette opération est nécessaire car dans la réalité les données observées dans l'espace considéré ne représentent qu'une trés petite partie des points considérés dans notre modèle.On aura alors :
dim(X_{obs})=m << dim(X(t))=n dim(C)=m_Xn dim(F)=n_Xn

Dès lors, si nous appliquons le nudging à notre modèle, nous allons obtenir le système suivant :

$$\begin{cases} \frac{dX}{dt} = F(X)+K(Xobs - C(X)) \quad 0 < t < T \\ X(0) = x_0 \end{cases} \tag{2}$$

Ici, K représente la matrice de nudging ou de gain.

6.2 Nudging rétrograde

Une fois le système (1) résolut, on obtient une condition finale. Elle permet alors de définir la condition initiale du système de "retour" :

$$\begin{cases} \frac{d\tilde{X}}{dt} = F(\tilde{X}) \quad T < t < 0 \\ \tilde{X}(T) = \tilde{X}_0 \end{cases} \tag{3}$$

En appliquant le nudging à ce modèle retour on obtient le système suivant, ou K' reprèsente la matrice de nudging retour :

$$\begin{cases} \frac{d\tilde{X}}{dt} = F(\tilde{X})-K'(Xobs - C(\tilde{X})) \quad T < t < 0 \\ \tilde{X}(T) = \tilde{X}_T \end{cases} \tag{4}$$

6.3 Back and Forth Nudging (BFN) algorithm

Cet algorithme (développé par Mr Auroux et Mr Blum en 2005) consiste à résoudre successivement les systèmes (2) et (4) pour k≠1 :

$$\begin{cases} \frac{dX}{dt} = F(X) + K(Xobs - C(X)) \quad 0 < t < T \\ X(0) = x_0 \end{cases} \tag{5}$$

$$\begin{cases} \frac{d\tilde{X}}{dt} = F(\tilde{X}) - K'(Xobs - C(\tilde{X})) \quad T < t < 0 \\ \tilde{X}(T) = \tilde{X}_T \end{cases} \tag{6}$$

On définit un X(0)=x_0 puis on résout le système direct (5). Alors on obtient X_1(T), qui sera la condition initiale du système rétrograde (6). La résolution du système (6) nous fournit $X_2(0)$ et ainsi de suite.

De plus, si les observations X_{obs} sont discrètes en temps, c'est à dire disponibles uniquement aux temps $(t_i)_{i=1..N}$, alors le terme de nudging est uniquement ajouté en ces temps t_i :

$$\frac{dX}{dt} = F(X) + \sum_{i=1}^{N} K(Xobs - C(X))\delta(t - t_i) \tag{7}$$

7 Résultats numériques

Tout d'abord, nous avons fait tourner le code NEMO à partir d'une condition initiale considérée vrai. Nous avons alors obtenus une modélisation des différentes variables (hauteur d'eau, salinité, vitesses de flux etc...) sur l'espace temporel considéré. Cet état va être considéré comme notre expérience de référence, soit la réalité.

7.1 Expérience de référence dans la configuration GYRE

Le code NEMO enregistre ces résultats numériques dans des fichiers de type .NetCdf. Pour lire ce genre de fichier, nous avons du installer le logiciel *ferret* (La présentation du logiciel ce situe dans la partie aspects informatiques, traité par la suite).

7.1.1 Profil de la salinité

Le profil de la salinité est défini constant en fonction de la hauteur z à t=0. Elle est calculée à chaque pas de temps.

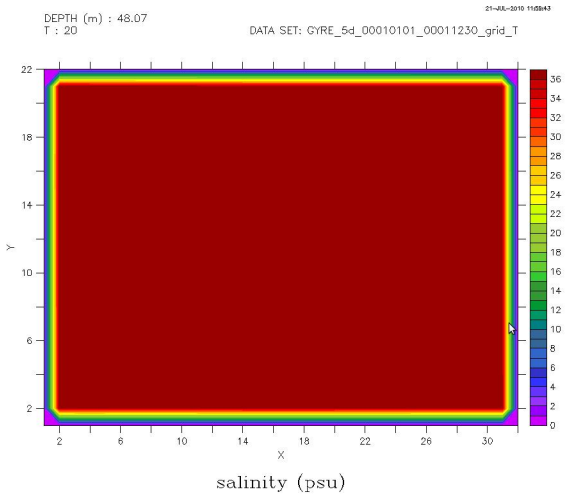


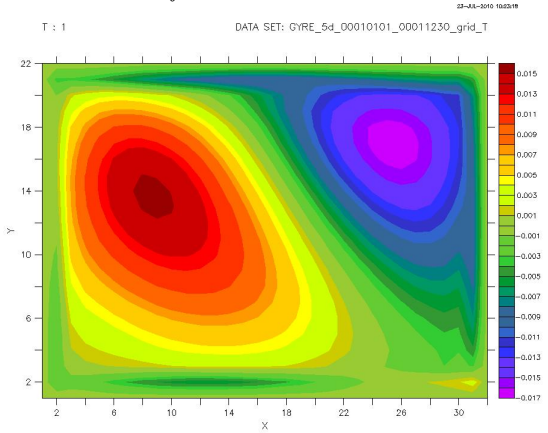
FIGURE 4 – *Salinité dans le domaine*

On remarque ici que la salinité est toujours constante à notre 20ème pas de temps, on peut donc dire qu'elle n'évolue pas.

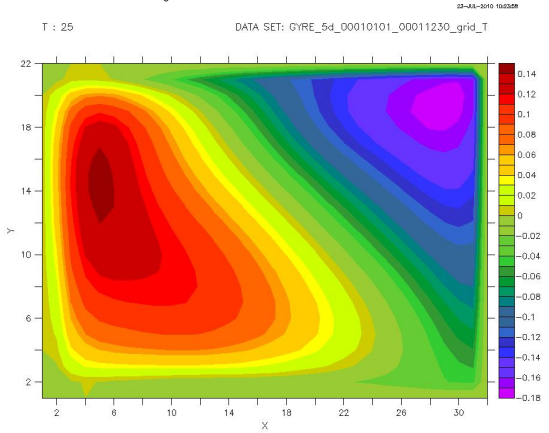
7.1.2 Profil de la Hauteur d'eau

Nous allons ici représenter l'évolution de la hauteur d'eau SSH représentée par la variable SOSSHEIG. A t=0, on définit un profil de hauteur d'eau proche de la réalité puis on regarde son évolution.

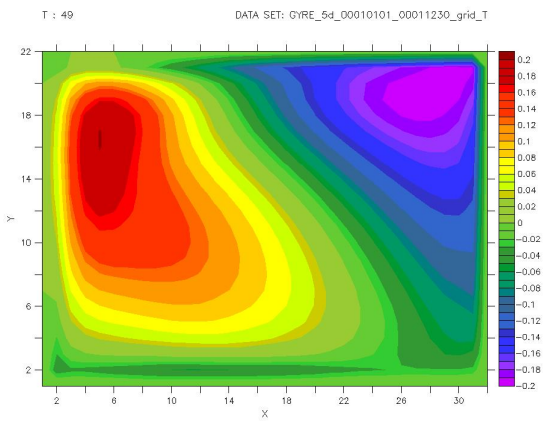
Profil de la hauteur d'eau à t=0



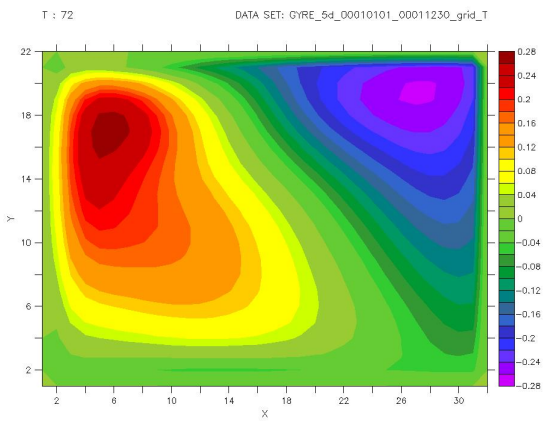
Profil de la hauteur d'eau à t=25



Profil de la hauteur d'eau à t=49



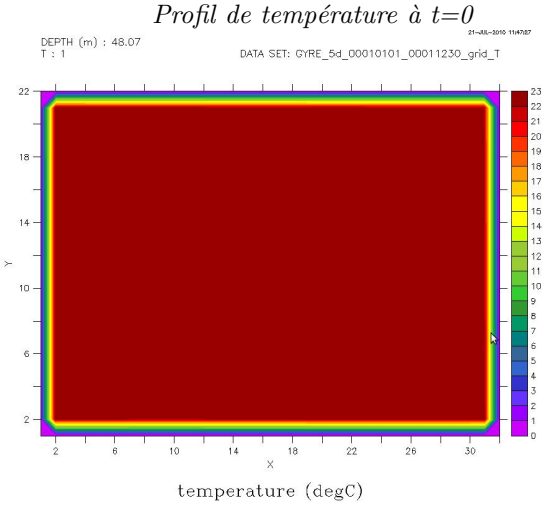
Profil de la hauteur d'eau à t=72



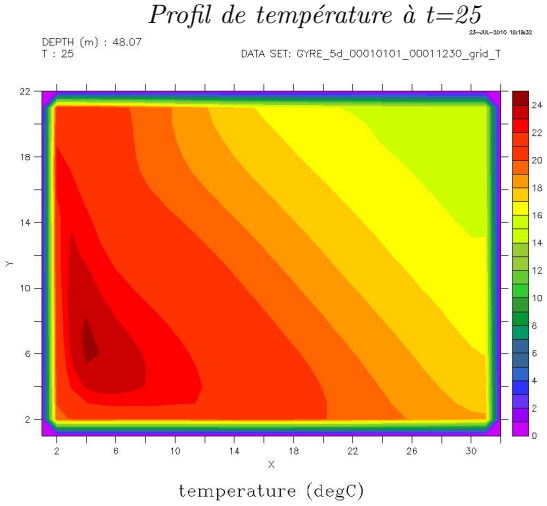
7.1.3 Profil de la température

Nous allons ici présenter l'évolution de la température représentée par la variable VOTEMPER. A t=0 on considère une température constante par couche z, puis on regarde son évolution. Nous allons voir ici l'évolution à une profondeur de 48.07m, soit k=5.

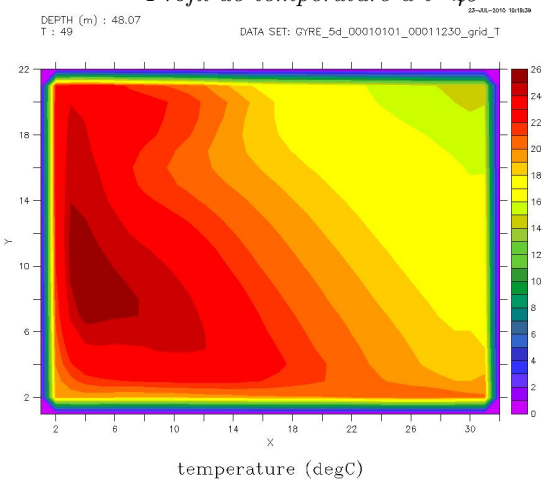
Profil de température à t=0



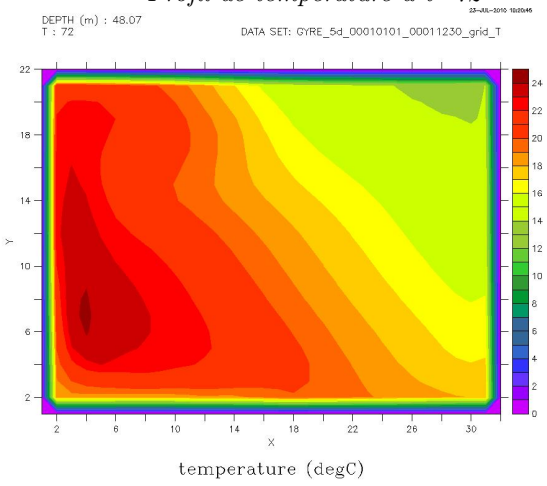
Profil de température à t=25



Profil de température à t=49



Profil de température à t=72



On remarque ici, que le forcage du vent (surface boundary condition) et la force de Coriolis engendrent une rotation et modifient le profil de température au fur et à mesure du temps.

7.2 Principe des expériences jumelles

On va ici réaliser des expériences dites jumelles, qui sont résumées par le schéma suivant :

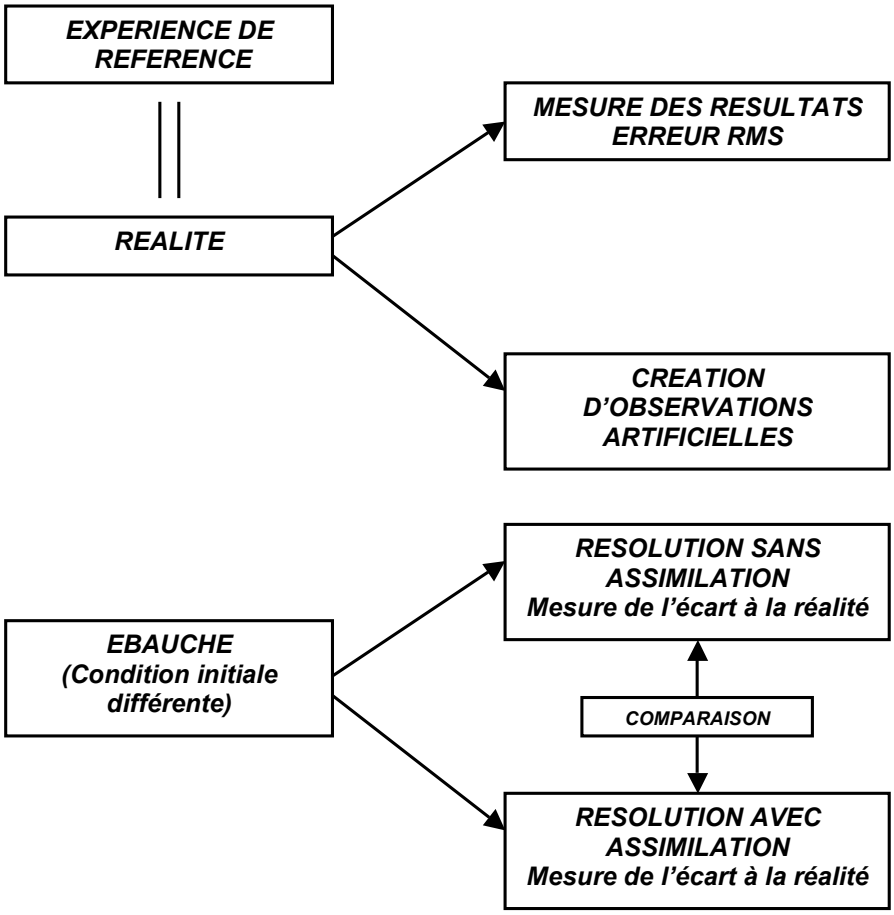


FIGURE 5 – *Principe des expériences jumelles*

Ici l'unité utilisée est l'unité RMS relative, qui représente l'erreur en pourcentage, obtenue par le calcul de l'écart relatif entre les valeurs issues de l'état assimilé et celles issues de l'état vrai :

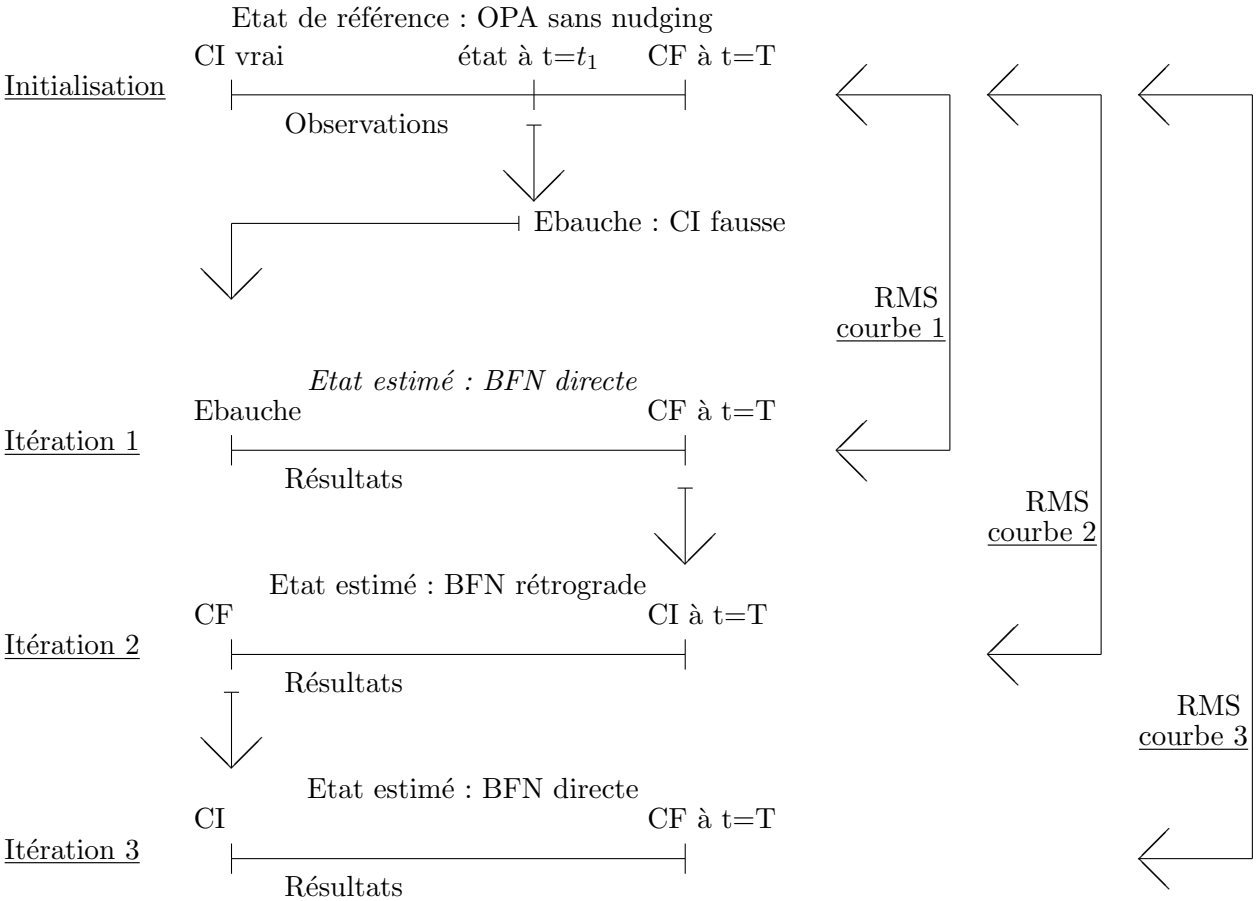
$$RMS_{relative} = \frac{\|X_{true}(t) - X(t)\|}{\|X_{true}(t)\|}$$

7.3 Expérience 1 : Influence sur l'erreur RMS de la méthode d'assimilation de données BFN

Nous allons voir ici l'influence de la méthode BFN sur l'erreur RMS. Dans un premier temps nous expliquerons comment cette méthode théorique est appliquée au sein du code NEMO. Alors, on pourra calculer l'erreur RMS à chaque itération et nous interprèterons les résultats obtenus.

7.3.1 Application de la méthode BFN au sein du code NEMO

Une fois notre état de référence créé (X_{true}), nous allons pouvoir définir une ébauche et appliquer la méthode BFN. Nous allons voir à partir du schéma suivant les différentes étapes de l'algoritme.



Remarque : Les courbes 1, 2 et 3 correspondent aux courbes de la résolution sur l'erreur RMS situées dans la partie suivante.

Nous allons voir ici les différentes opérations à effectuer au sein du code pour executer cet algorithme.

- Initialisation : A partir d'une condition initiale fixée, on définit notre état de référence. On en extrait l'ébauche qui correspond à l'état de référence à un instant t=t_1.

- Itération 1 : En premier lieu, à partir de l'ébauche et des observations nous allons estimer un nouvel état, avec l'algorithme du Nudging direct.
  A l'aide d'une subroutine *RMS* nous allons pouvoir calculer l'erreur RMS entre les observations et l'état estimé.
  Ensuite, nous allons apporter les modifications du code (puis le recompiler) pour effectuer l'opération de BackNudging, et utiliser une subroutine *CIbackNudg* qui va permettre d'aller chercher dans les fichiers résultats de l'itération 1 la condition finale pour la définir comme condition intiale de la méthode rétrograde de l'itération suivante.

- Itération 2 : A partir de l'état final de l'itération 1 et des observations nous estimerons un nouvel état, avec l'algorithme du Nudging rétrograde.
  A l'aide d'une subroutine *RMS* nous pourrons alors calculer l'erreur RMS entre les observations et l'état estimé.
  Ensuite, nous apporterons les modifications du code (puis le recompiler) pour effectuer l'opération de Nudging direct, et utiliser une subroutine *CINudgDirect* qui va permettre d'aller chercher dans les fichiers résultats de l'itération 2 la condition finale pour la définir comme condition intiale de la méthode directe de l'itération suivante.

- Itération 3 : On effectue la même opération que pour l'itération 1.

7.3.2 Résultat de la méthode sur l'erreur RMS

On va observer ici les différentes évolutions de la température dans notre modèle, en appliquant ou non le nudging aux différents instant t.

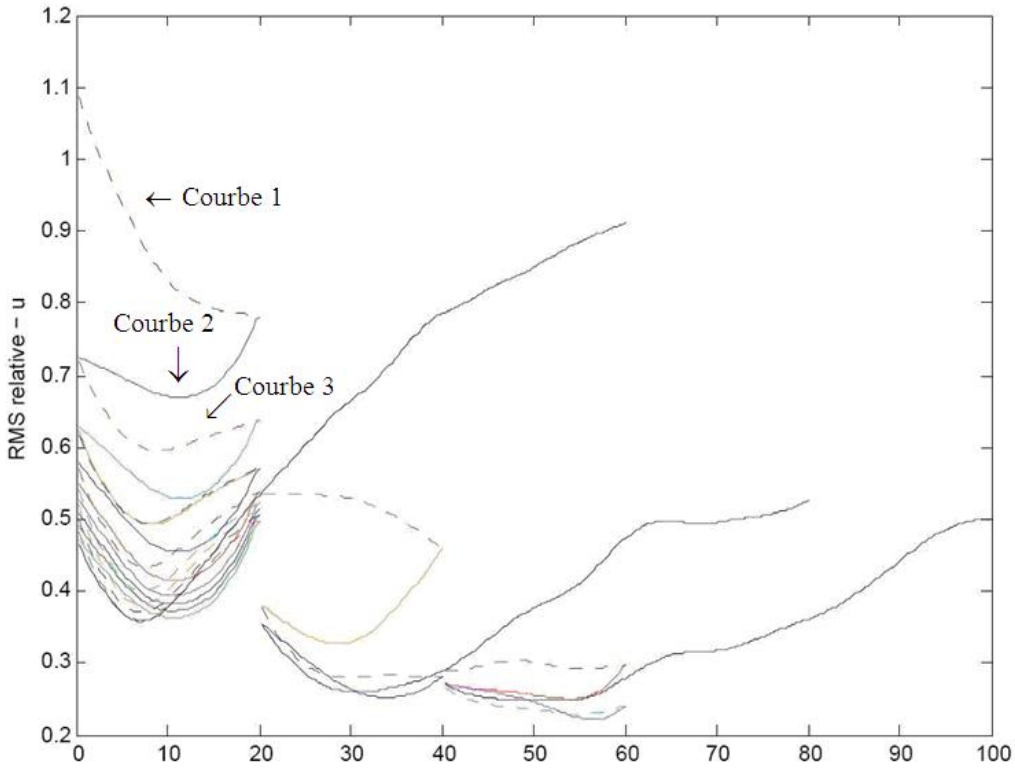


FIGURE 6 – *Evolution de l'erreur RMS sur la température avec et sans Nudging*

L'expérience réalisée ici se décompose de la manière suivante :

- Définition d'un état de référence représentant la réalité.
- Choix d'une condition intiale à t = 0 différente de la réalité (RMS=1.1).
- Appliquation de l'algorithme du nudging entre t = 0 et t = 20 jours. La convergence nous fournit la condition intiale optimale de notre schéma à t = 0. On obtient alors la condition finale de notre première résolution, qui va être notre condition intiale de la prochaine résolution à t = 20 jours.
- On effectue ici 2 résolutions :
  - Un résolution de notre système sans le nudging de t = 20 jours à t = 60 jours
  - Un résolution de notre système avec le nudging de t = 20 jours à t = 40 jours
- On effectue encore les 2 types de résolutions, en partant de la condition initiale fournit par la convergence du nudging entre t = 20 jours et t = 40 jours.

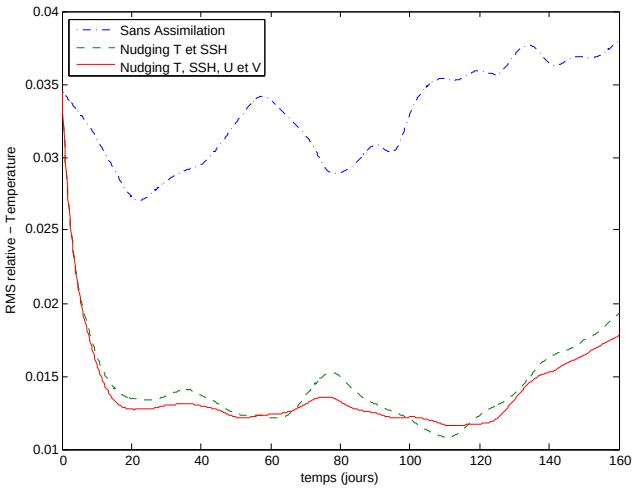
On remarque ici que sans le nudging, le modèle s'éloigne de la réalité. On remarque donc l'interêt d'utiliser cette méthode pour ajuster nos modèles au plus juste.

7.4 Expérience 2 : Comparaison de la résolution avec ou sans assimilation de données

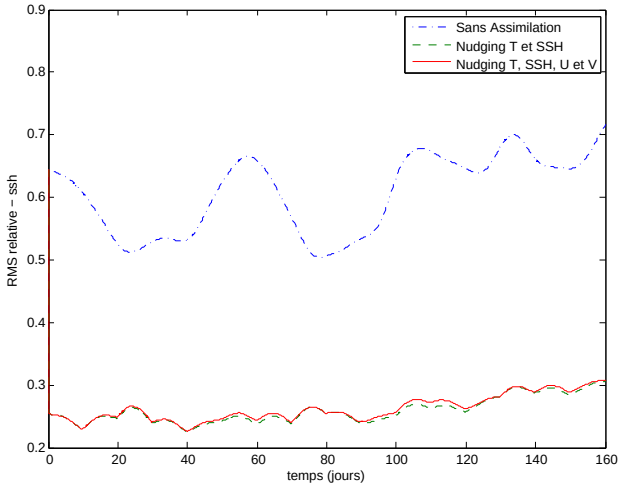
7.4.1 Résultats numériques du Nudging Direct

On débute notre algorithme d'une condition initiale fausse (RMS $\neq 0$) et on observe les différentes évolutions avec l'application, ou non, du nudging direct.

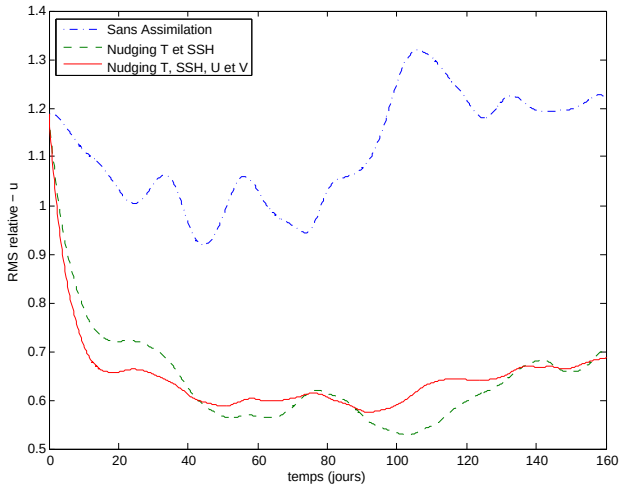
Résultats sur la tempéraure



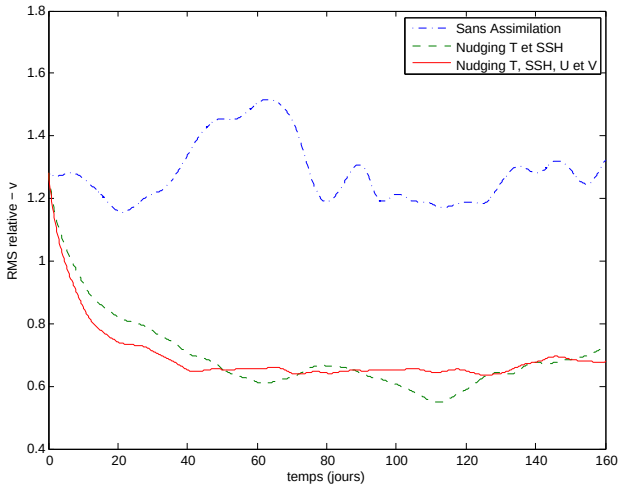
Résultats sur la hauteur d'eau



Résultats sur U (vitesse du courant horizontale)



Résultats sur V (vitesse du courant verticale)

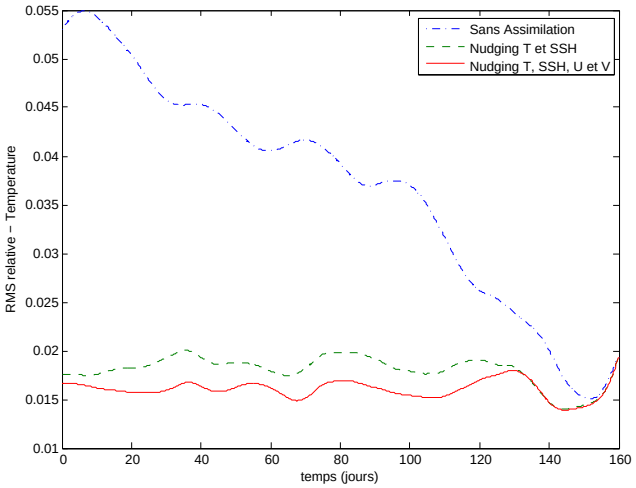


On remarque ici qu'en partant d'une condition initiale fausse, l'absence de nudging ne permet pas de converger vers la réalité. Contrairement, l'application du nudging permet de faire diminuer l'erreur RMS.

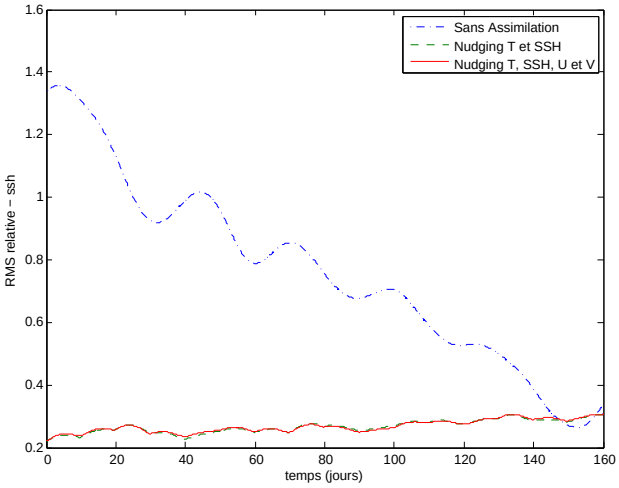
7.4.2 Résultats numériques du Nudging Rétrograde

On débute notre algorithme d'une condition finale optimale (RMS ≈ 0) et on observe les différentes évolutions avec l'application, ou non, du nudging rétrograde.
Remarque : Il faut suivre l'évolution de la courbe de droite à gauche pour le nudging rétrograde.

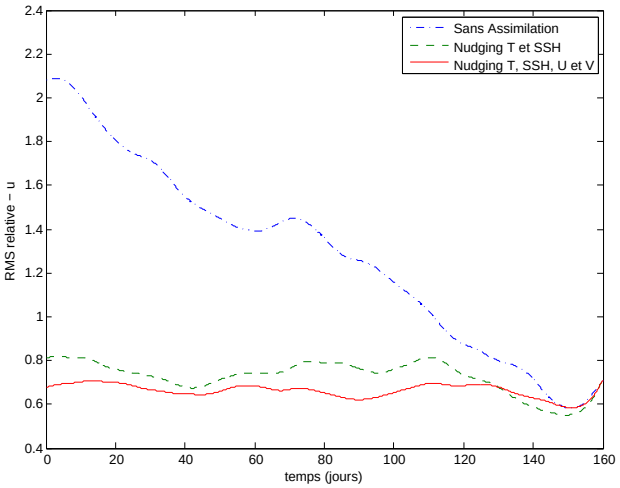
Résultats sur la tempéraure



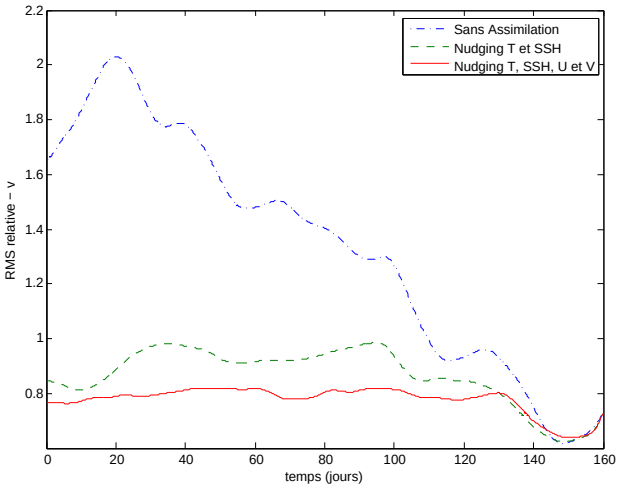
Résultats sur la hauteur d'eau



Résultats sur U (vitesse du courant horizontale)



Résultats sur V (vitesse du courant verticale)



On remarque ici qu'en partant d'une condition finale optimale, l'absence de nudging engendre une erreur sur le calcul de notre condition intiale. Contrairement, l'application du nudging permet de maintenir l'erreur RMS au cours de la résolution.
La performance du nudging rétrograde est moins importante que le nudging direct, car il n'a pas de sens physique.

Deuxième partie

Aspects informatiques

8 Installation et compilation du code NEMO

Avant tout, la première étape a été d'installer et compiler le code NEMO dans un environnement parallèle, au laboratoire J.A. Dieudonnée. La machine parallèle est de type : ...

Le code NEMO est téléchargeable à partir du site http ://www.nemo-ocean.eu/ pour cela, on a utilisé le fichier d'installation fournit par le site. Cependant nous avons été confronté à plusieurs problèmes, et certaines modifications on été apportées.

1) Definir l'alias suivant pour utiliser SVN (here in ksh) :
alias svn_ano='svn co http ://forge.ipsl.jussieu.fr/igcmg/svn/modipsl/trunk modipsl'

2) Creér et aller dans le répertoire de travail :
mkdir TRY ; cd TRY

3) Extract modipsl :
svn_ano

4) Extract NEMO :
cd modipsl/util
./model NEMO

5) Choix de la configuration GYRE :
../modeles/UTIL/fait_config GYRE

6) Installation des makefiles (the following argument "target host " usually corresponds to your platform name resulting from the command ./w_i_h)

Se rappeler de modifier *TRY/modipsl/util/AA_make.gdef* for target lxiv8 :

....
#-Q- lxiv8 F_C = mpif90 -f90=ifort -c -cpp
....
....
#-Q- lxiv8 F_L = mpif90 -f90=ifort
....
#-Q- lxiv8 NCDF_LIB = -L/usr/local/lib -lnetcdf

Fortran Source utilise mpi.mod. Nous allons devoir faire :
Creér un fichier mpi.f90 dans modipsl/lib avec le contenu suivant :

MODULE mpi
INCLUDE 'mpif.h'
END MODULE mpi

Puis le compiler avec la commande suivante :
mpif90 -f90=ifort -c mpi.f90
./ins_make -t lxiv8

7) Compilation :
cd ../config/GYRE
gmake

Remarque : Au sein du code NEMO, un système de clés a été créé pour permettre de sélectionner uniquement certaines partie du code. Ceci a pour objectif de compiler uniquement les parties de code nécessaire à la configuration choisie. C'est à la création du *makefile* que l'on va sélectionner les clés nécessaire à la compilation. Dans notre configuration les clés utilisées sont :
$key\_zco, key\_gyre, key\_dynspg\_flt, key\_ldfslp, key\_zdftke, key\_vectopt\_loop$
$key\_vectopt\_loop, key\_vectopt\_memoryetkey\_iomput$
Lorsque nous utilisons la version parallèle du code, il ne faut pas oublier d'ajouter la clé *key_mpp_mpi* puis de recompiler le code pour pouvoir l'utiliser.

9 Lancement d'un test :

1) Définir un environnement MPI et VSMP
source /workspace/IntelEnv-11.1.072

2) Copie des fichiers nécessaire au lancement du test, pour un nouveau run :
cp -a /workspace/dreyfuss/TRY/modipsl/config/GYRE/EXP00 EXP01

3) Pour lancer un run, il faut se placer dans le répertoire EXP01 et lancer la commande :
/workspace/dreyfuss/TRY/modipsl/bin/opa

4) Pour lancer un run avec MPI, il faut redéfinir à l'intérieur du code le nombre de processeurs utilisés et recompiler le code. Puis lancer la commande :
/workspace/runmpi-numabind ¡nb of processes¿ /workspace/dreyfuss/TRY/modipsl/bin/opa

5) Les résultats seront dans les fichiers opa.XXXXX et autres, dans le répertoire EXP01

10 Le debbuger, un outil d'analyse

Lancement du debbuger

1) Il faut tout d'abord définir l'environnement de travail :
source /workspace/IntelEnv-11.1.072

2) Puis lancer le debugger : *idb -gui /workspace/dreyfuss/TRY/modipsl/bin/opa*
Grâce au debugger on peut alors suivre pas à pas le déroulement de l'algorithme.

Intéret du debugger L'utilisation d'un debugger a été indispensable, au cours de ce stage, pour comprendre comment le code NEMO fonctionne. Il permet de pouvoir suivre le déroulement de l'algorithme, et de le faire avancer pas à pas.
C'est par ce moyen que l'on a pu définir les différentes parties du code qui servent pour l'initialisation, pour la résolution ainsi que pour l'obtention de résultats.

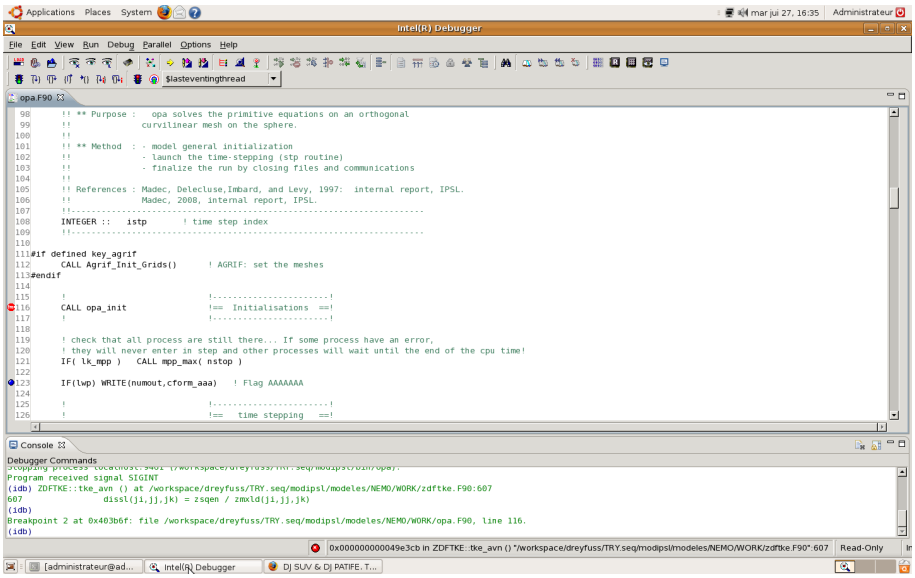


FIGURE 7 – *visualisation de l'interface graphique du logiciel idb*

Nous allons voir sur la figure suivante les différentes options pour faire avancer le code pas à pas, grâce aux différentes commandes du logiciel.

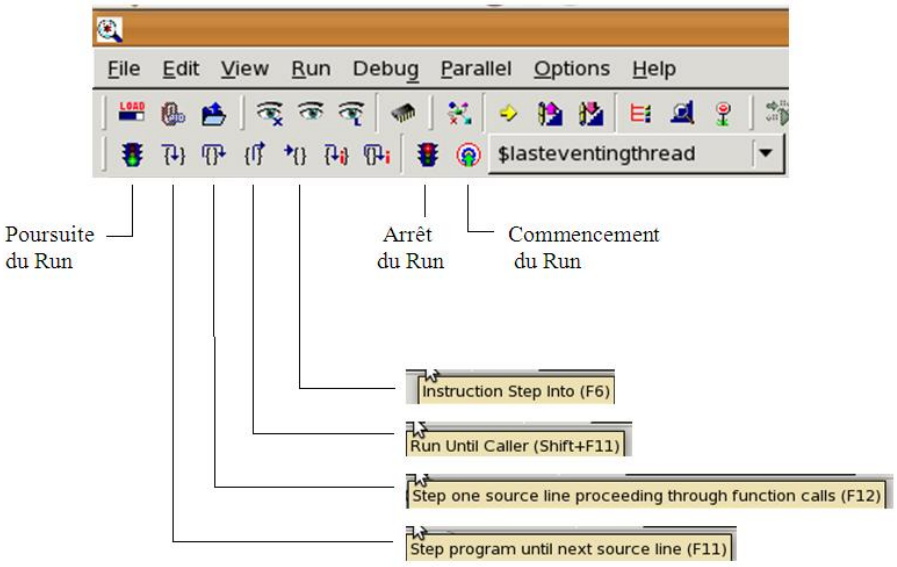


FIGURE 8 – *visualisation de l'interface graphique du logiciel idb*

11 Ferret, un logiciel de visualisation pour les fichiers netCdf

Ce logiciel nous permet de lire ces fichiers qui contiennent tous les résultats sur toutes les variables(Hauteur d'eau, salinité, température, etc).
C'est grace à la commande *show data* que l'on peut voir toutes les variables que l'on peut représenter. Nous allons pouvoir afficher leurs évolutions grâce à la commande *fill*.

1) Définir l'environnement de travail :
csh
source /usr local/ferret/ferret_paths.CSH

2) Lancer du programme :
/usr local/ferret/bin/ferret

3) Ouverture du fichier NetCdf par le logiciel ferret. On utilise la commande *use* :
use GYRE_5d_00010101_00011230_grid_T.nc

4) On peut alors voir toutes les variables contenues dans notre fichier, ainsi que leurs caractéristiques et leurs dimensions (I, J et K représentent les coordonnées spatiales, L le temps) :
show data

5) On obtient une représentation graphique avec la fonction *fill* et en fixant les différents paramètres I, J, K et L. Par exemple, pour représenter la température, on va fixer la hauteur au 5ème niveau et le temps au 20ème pas de temps :
fill VOTEMPER[k=5,l=20]
C'est avec cette commande qu'on à pu obtenir le profil de salinité présenté dans la partie résultats (6.1).

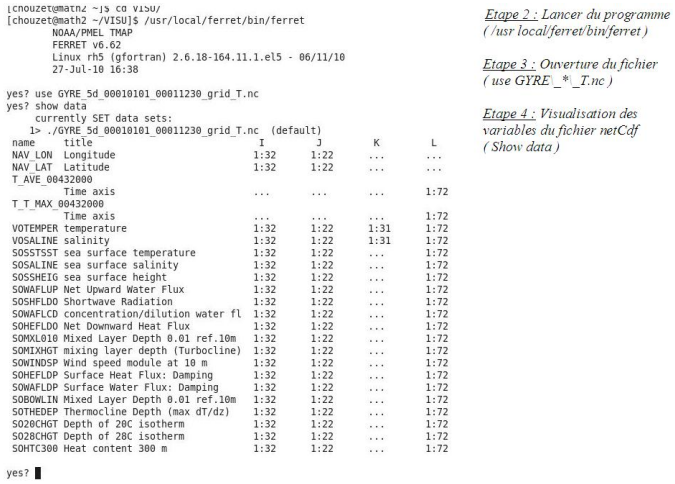


FIGURE 9 – *visualisation des différentes variables par ferret*

12 Mise en place de la méthode BFN au sein du code NEMO

Cette méthode a été mise en place par P. Bansart, actuellement en thèse au laboratoire. Nous allons voir ici les différentes modications de code qui ont été nécessaire au sein de la boucle de résolution, présentée en partie 1. En observant l'algorithme théorique de la méthode BFN, on peut tout de suite voir que les modifications apportées seront différentes si la méthode est directe ou rétrograde.

Nous allons voir ici les différents fichiers modifiés :

Initialisation

namelist :

Le signe de plusieurs paramètres va dépendre si la méthode est rétrograde ou directe :

- La diffusion verticale : *ahto* et *ahmo* sont positif dans le cas direct (négatif dans le cas rétrograde)
- La diffusion horizontale : *avto* et *avmo* sont positif dans le cas direct (négatif dans le cas rétrograde)
- Les pas de temps : *tdt,rdtmi,rdtmax* sont positif dans le cas direct (négatif dans le cas rétrograde)

istate

Une méthode de lecture de fichier netCdf a été mis en place, pour pouvoir définir une condition initiale à partir de nos fichiers résultats. C'est une subroutine qui à été crée : *rstseek_read*. L'ajout d'une variable *ln_rstseek* a été nécessaire dans la namelist pour activer cette subroutine.

Traceurs

trazdf_imp

Calcul des équations sur les traceurs. Application du nudging avec le paramètre K_T

tranxt

Cette fonction est appelée à la fin du pas de temps sur les traceurs. Elle écrit les résultats aux même pas de temps que notre fichier d'observation. C'est grâce à ces deux fichiers que l'on va pouvoir calculer l'erreur RMS.

Dynamique

dynzdf_imp

Cette fonction est appelée pour le calcul de u et v. Il y a ici une possibilité de pseudo nudging avec l'équilibre géostrophique. Cependant il n'est pas utilisée dans notre configuration.

dynspg_flt

Cette fonction est appelée pour le calcul de la hauteur d'eau. On calcul ici la hauteur d'eau relative η, ou la méthode de BFN a été appliquée.

Création d'un module restart :
Dans la configuration de P. Bansart, un module restart a été créé par nécessité. Il aura pour but d'accueillir plusieurs subroutines :

- *rstseek_read* : Elle lit les fichiers netCdf nécessaire à l'état initial dans chacune de nos itérations.
- *rst_write* : Elle permet d'écrire dans le type de fichier netCdf de cette configuration (appelé par le fichier step.F90)
- *rst_read* : Elle permet de lire le type de fichier netCdf de cette configuration (appelé par le fichier initdtr.F90)

Troisième partie

La méthode BFN en calcul parallèle

13 Méthode de parallélisation

Nous allons voir ici le principe de parallélisation, appliquée dans le code NEMO, sur un exemple simple : l'équation de la chaleur.

$$\frac{\partial u}{\partial t} - u'' = f(x,t)$$

Dans cet exemple et dans le code NEMO, le choix est fait de choisir un schéma de discrétisation dit explicite. C'est à dire que la connaissance des valeurs des variables au temps t^n permet un calcul direct des valeurs des variables au temps t^{n+1}, sans passer par la résolution d'un système global.
On choisira dans notre exemple un schéma d'Euler explicite de la forme :

$$\begin{aligned} u(t_{n+1}) &= u(t_n) + \Delta t(f(t_n) + u''(t_n)) \\ \Leftrightarrow u_i(t_{n+1}) &= u_i(t_n) + \Delta t(f(t_n) + \tfrac{u_{i-1}(t_n) - 2u_i(t_n) + u_{i+1}(t_n)}{h^2}) \end{aligned}$$

La parallélisation s'éffectue alors de la manière suivante :

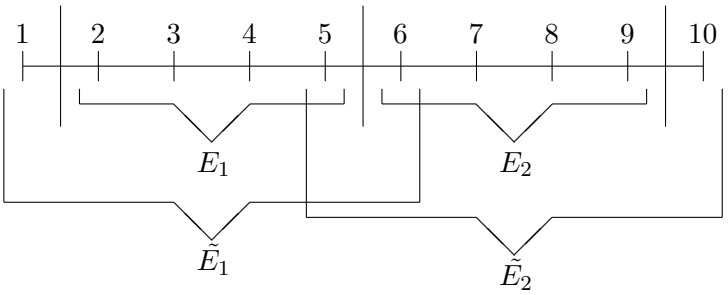


FIGURE 10 – *Principe de parallèlisation 1D*

Processus de résolution :

- $t = 0$: On connait $\tilde{E}_1$ et $\tilde{E}_2$ définit par la condition initiale et stockée
- $t = t_n$:
  - Calcul parallèle de $E_1(t_{n+1})$ à partir de $\tilde{E}_1(t_n)$ sur le processeur 1
  - Calcul parallèle de $E_2(t_{n+1})$ à partir de $\tilde{E}_2(t_n)$ sur le processeur 2
  - Mise à jour de $\tilde{E}_1(t_{n+1})$ et $\tilde{E}_2(t_{n+1})$ à partir de $E_1(t_{n+1})$ et $E_2(t_{n+1})$

Nemo, un code tridimensionel

Le code NEMO étant en trois dimensions, le parallèlisme s'effectue sur des couches z. On pourra voir un schéma de Discretisation en Annexe 3.

14 Conclusion

Ce stage a été une expérience professionnelle très enrichissante sur tous les plans : aussi d'un point de vue de l'approfondissement de mes connaissances en mathématiques et informatique que du point de vue relationnel. Il m'a permis d'apprecier le travail sur un code qui nécessite l'intervention de plusieurs équipes de recherches.

J'ai alors été confronté aux difficultés de travailler et de modifier un code de cette taille.
La première étape a été de bien identifer chaque partie du code ainsi que leurs fonctions. On pourra retenir les trois parties réalisant les étapes importantes : le choix de la configuration, l'initialisation et la résolution. Pour cela, l'apprentissage de l'utilisation d'un debugger a été nécessaire.
Puis, la compréhension de la méthode de résolution a été primordiale. Ceci pour effectuer les modifications nécessaires pour integrer la méthode au sein du code NEMO.

Une fois ce travail réalisé, on a pu apprécier l'efficacité de cette méthode sur la modélisation de l'ocean. Le manque de temps ne m'a pas permis de tester les performances de la version parallèle du code pour la méthode BFN.

Au terme de ce stage, j'ai eu la satisfaction d'avoir programmé une méthode au sein d'un code en développement. En effet, ce stage m'a permis non seulement d'approfondir mes connaissances en informatique mais aussi d'acquérir une expérience extrêmement valorisante d'un point de vue personnel.

Références

[1] Auroux D and Blum J 2008 A nudging-based data assimilation method : the Back and Forth Nudging (BFN) algorithm *Nonlinear Processes in Geophysics* **15** pp 305–319

[2] Nodet M 2005 Modélisation mathématique et assimilation de données lagrangiennes pour l'océanographie.*Nice Univ. Presse, Thèse*

[3] Bansard P 2010 Application de la méthode BFN dans le code NEMO. *Nice Univ. Presse, Thèse en cours*

Quatrième partie

Annexes

Représentation du domaine GYRE

On pourra voir l'emplacement de notre domaine par rapport à la terre. Il s'agit d'une partie de l'atlantique nord située à latitude moyenne :

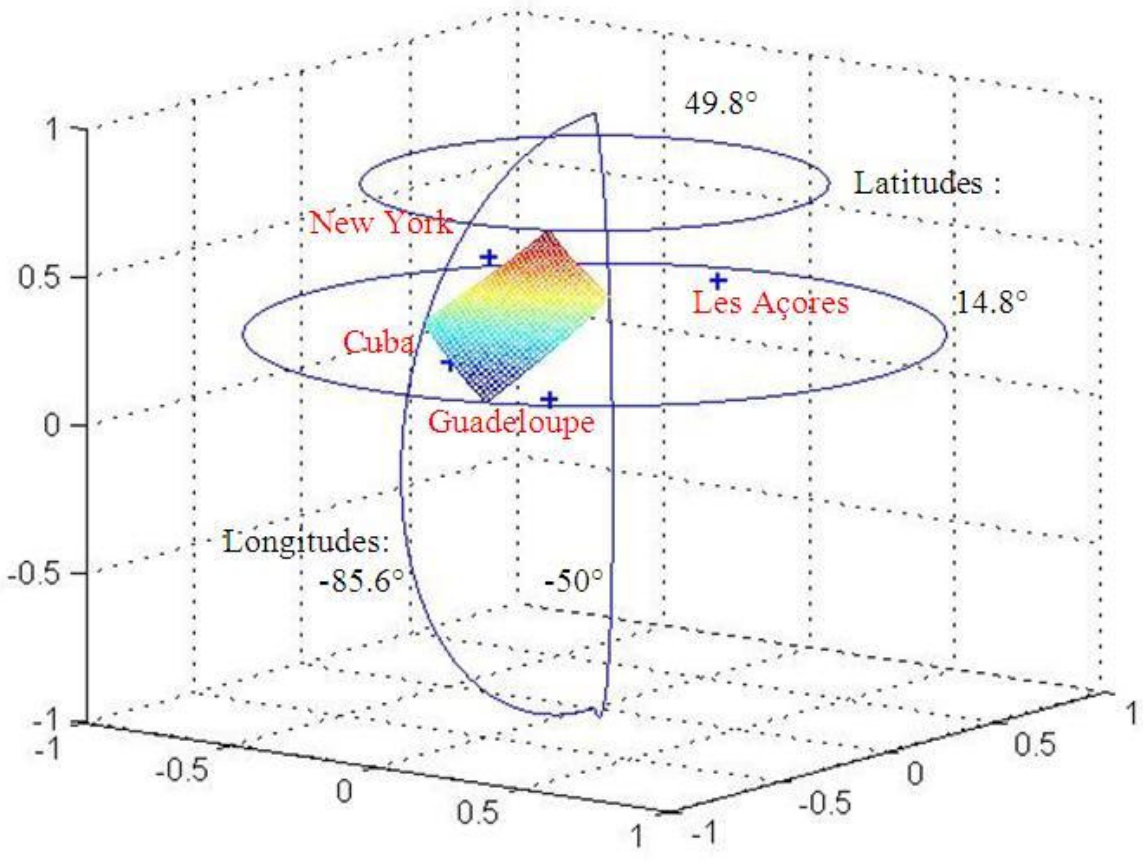


FIGURE 11 – *domaine GYRE*

Niveau d'eau sur la profondeur z

La profondeur maximum est de 4451.26m, elle est découpée en 31 couches. Nous allons voir les différentes hauteurs d'eau considérées au sein de notre configuration :

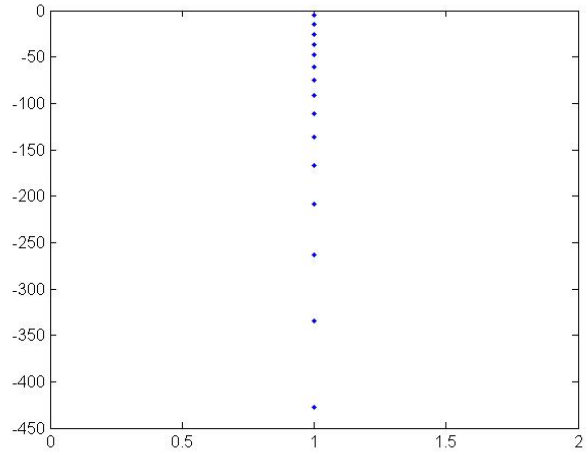


FIGURE 12 – *15 premiers niveaux d'eau sur la profondeur de 0 à -500m*

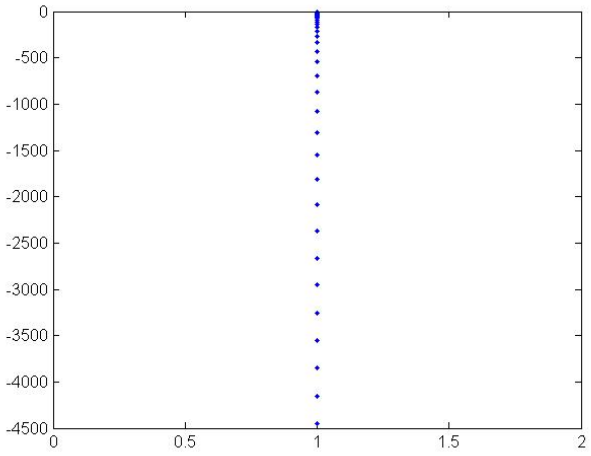


FIGURE 13 – *Niveau d'eau sur la profondeur de 0 à -4500m*

Découpe du maillage en vue du calcul parallèle

On pourra voir ici le découpage du domaine sur une couche z pour utiliser notre méthode parallèle.

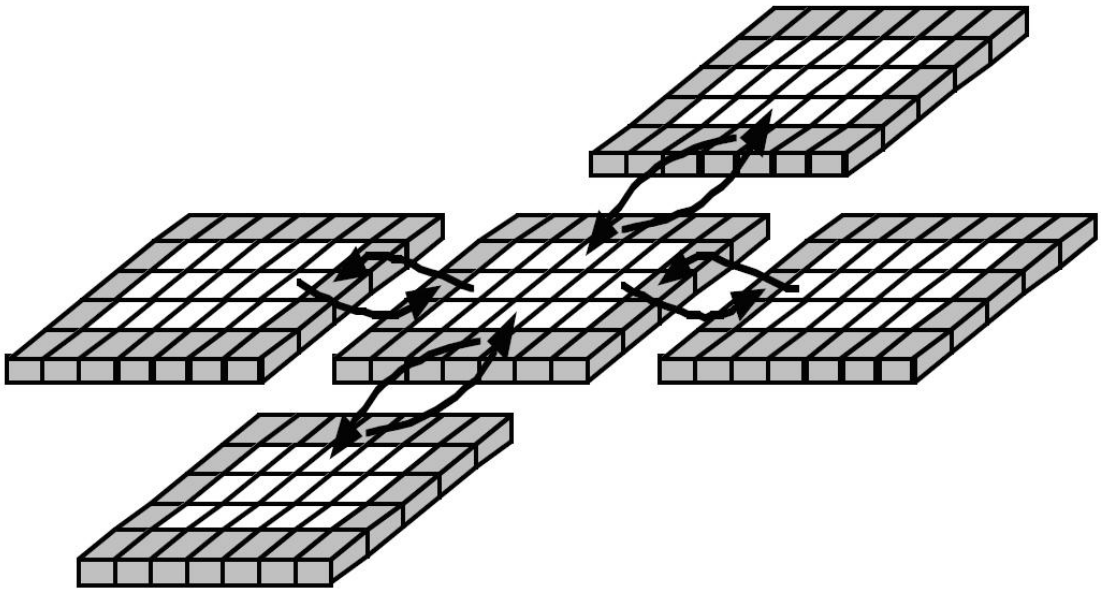


FIGURE 14 – *Découpage du maillage*