T2DInverse: Towards calibration and sensitivity analysis into Telemac2D using automatic differentiation

Youssef Loukili, Marc Honnorat, Jerome Monnier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

T2DInverse: Towards calibration and sensitivity analysis into Telemac2D using automatic differentiation

Youssef Loukili — Marc Honnorat — Jérôme Monnier

N° 5618

Juillet 2005

Thème NUM

ISSN 0249-6399

INRIA
RHÔNE-ALPES

T2DInverse: Towards calibration and sensitivity analysis into Telemac2D using automatic differentiation

Youssef Loukili *, Marc Honnorat †, Jérôme Monnier ‡

Thème NUM — Systèmes numériques
Projet Idopt

Rapport de recherche n° 5618 — Juillet 2005 — 52 pages

Abstract: The industrial system Telemac2D, developed by LNHE-EDF and commercialized by Sogreah Co., is an software dedicated to the hydrodynamical and environmental modelling of maritime and river flows. It is based on finite element method and is written in Fortran 90. Our final objective is to include a full variational data assimilation process in the system. This requires the development of an adjoint model. We plan to obtain it using the Automatic Differentiation (AD) tool Tapenade v2.0. In this study, we prepare and validate a reduced version of Telemac2D that is both meant for river flows and suitable to Tapenade v2.0. Then, we establish a strategy of AD according to limitations of the current release of Tapenade (v2.0). Also, the optimization chain is achieved thanks to the unconstrained minimizer N1QN3 based on a quasi-Newton type method. The final objective is to exploit the fluvial measurements and observations in an optimal manner in Telemac2D, in order to identify the partially known or lacking parameters (bathymetry, bed friction, inflow discharge and initial state).

Key-words: Telemac2D, River Hydraulics, Flood modelling, automatic differenciation, Tapenade v2.0, variational data assimilation, sensitivity analysis, parameter identification

* Youssef.Loukili@imag.fr
† Marc.Honnorat@imag.fr
‡ Jerome.Monnier@imag.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

T2DInverse: en vue de l'assimilation de données et d'analyses de sensibilité avec Telemac2D à l'aide de la différenciation automatique

Résumé : Le logiciel industriel Telemac2D, dévellopé par LNHE-EDF et commercialisé par la société Sogreah, est dédié à la modélisation numérique d'écoulements marins et d'hydraulique fluviale. Le code est basé sur la méthode des éléments finis et est développé en Fortran 90. Notre objectif est d'y implémenter une méthode d'assimilation variationnelle de données. Cela nécessite le développement de son code adjoint. Nous prévoyons de l'obtenir à l'aide de l'outil de différenciation automatique Tapenade v2.0 dévellopé au sein du projet INRIA-Tropics. Dans le présent travail, nous préparons et validons une version réduite du code Telemac2D. Cette version réduite est dédiée à l'hydraulique fluviale et à sa différenciation automatique par Tapenade v2.0.

Notre objectif final est d'une part l'analyse de sensibilité et d'autre part l'assimilation de données pour les modèles de crues. Les paramètres à identifier sont généralement la topographie, les coefficents de frottement, les débits amonts et l'état initial. Nous portons une attention toute particulière aux incompatibilités entre les fonctionnalites Fortran 90 utilisées dans le code source initial de Telemac2D et les limitations actuelles du logiciel Tapenade (v 2.0). Nous dressons une liste exhaustive des points à résoudre en vue d'obtenir le code adjoint de manière automatique. Par ailleurs, nous présentons la chaîne usuelle d'optimisation code direct - code adjoint - optimiseur implémentée et qui permettra a terme l'assimilation de données une fois le code adjoint obtenu en intégralité.

Mots-clés : Telemac2D, Hydraulique fluviale, modèles de crues, différenciation automatique, Tapenade v2.0, assimilation variationnelle de données, analyse de sensibilité, identification de paramètres

1 Introduction

Les équations de Saint-Venant ("Shallow-Water") sont bien adaptées pour la description de certains écoulements géophysiques à surface libre tels que les inondations fluviales, les circulations estuaires et côtières par exemple.

Telemac2D est un logiciel de simulation numérique basé sur les équations de St-Venant et pouvant considérer des configurations réelles. Il fait partie du système global Telemac développé par le Laboratoire National d'Hydraulique et Environnement -LNHE- d'EDF et commercialisé par la société SOGREAH. Son codage a commencé il y a de nombreuses années, et sa version actuelle est écrite en Fortran 90. Pour faire fonctionner Telemac2D sur un cas réel d'écoulement d'inondations, on a besoin d'un minimum d'informations de terrain qui représentent les entrées du modèle: la bathymétrie de la rivière avec la topographie des zones inondables, la friction du lit correspondante à une loi de frottement convenable, l'état initial de l'écoulement et les débits ou variables d'état (hauteur et vitesses) sur les limites amont et aval de la rivière.

Ces données de terrain font partie du modèle global conçu pour simuler la réalité de l'écoulement. Mais malheureusement elles sont souvent manquantes ou partielles, car coûteuses ou difficiles à mesurer. De ce fait, les expertises de prévention comptent notamment sur l'analyse des résultats de nombreuses simulations utilisant notamment des valeurs extrêmes pour certaines données mal connues. Le problème pourrait se poser ainsi: comment peut-on estimer les variables d'entrées à partir d'observations (réelles) de l'écoulement? Les méthodes d'identification, calage et plus globalement d'assimilation de données sont de bons candidats pour résoudre ce type de problème. L'idée des méthodes d'assimilation de données étant de donner autant d'importance aux observations qu'au modèle considéré, et ceci en vue de les confronter de manière "consistante". Ces méthodes connaissent un grand succès dans de nombreuses applications industrielles et environnementales, typiquement prévisions météorologiques et océanographiques.

Mathématiquement, ces méthodes reviennent à définir des variables de contrôle optimale qui minimisent l'écart entre les observations et les simulations.
Les algorithmes de minimisation employés nécessitent le calcul du gradient de la fonction coût, et ce calcul requiert la solution d'un modèle dit adjoint (l'état adjoint du système).

Une première étude autour de l'identification de paramètres pour le système "Shallow-Water" implémenté dans Telemac2D a déja été mené, voir [2]. Cette étude était, tout comme dans le présent travail, basé sur les méthodes de contrôle optimal et de l'adjoint. L'adjoint du code ayant été obtenu par différenciation des équations puis implémentation des équations.

Le but de notre travail est d'incorporer un processus complet d'assimilation de données au logiciel de modélisation hydrodynamique Telemac2D, et ceci en générant le code adjoint

nécéssaire à l'aide des outils de différenciation automatique. Telemac2D présente une base importante de cas réels qui servira à la validation de cette approche dans le contexte de l'hydraulique fluviale et de la prévention numérique de crues.

C'est en effet pour l'obtention du code adjoint que la *Différentiation Automatique* (DA) peut être d'une aide précieuse. L'étude du modèle adjoint avec sa discrétisation et son implémentation peut s'avérer très coûteuse en temps. Aussi, dés que le modèle direct est modifié, l'adjoint correspondant doit être également modifié. La DA permet de produire à partir des instructions du code direct et sous certaines conditions de programmation, un codage qui correspond aux instructions adjointes du code direct. Afin de réaliser cette opération pour Telemac2D, nous avons opté pour le différentiateur automatique **Tapenade**, dédié aux codes de calcul écrits en Fortan. Tapenade est un outil logiciel développé et distribué gratuitement par l'équipe Tropics de l'INRIA - Sophia-Antipolis.

Ce rapport de recherche est organisé comme suit :

- dans la section 2, nous rappelons brièvement le modèle de Saint-Venant et sa discrétisation considérés par le code Telemac2D, nous nous référons à [3].
- dans la section 3, nous décrivons la méthode d'assimilation variationnelle employée et appliquée au cas de l'écoulement fluvial.
- dans la section 4, nous expliquons quelques bases pratiques de la DA, et nous montrons à travers un exemple comment on peut utiliser Tapenade v2.0 pour le calcul de l'adjoint.
- dans la section 5, nous rapportons les principales tâches de codage effectuées sur Telemac2D pour le préparer à la DA et l'accommoder à l'outil Tapenade v2.0. Ce qui a donné naissance à une version réduite recherche: Telemac2D-v5.4-simple.
- dans la section 6, nous proposons une stratégie globale de DA de Telemac2D-v5.4-simple par étapes, et nous illustrons la DA de la routine `VECTOR` avec son arborescence.
- dans la section 7, nous implémentons la chaîne d'optimisation dans le code, qui utilise une méthode de type quasi-Newton réalisée grâce à N1QN3 de MODULOPT développé à l'INRIA-Rocquencourt.

2 Modèle mathématique

En hydraulique maritime ou fluviale, les équations de Saint-Venant sont utilisées pour représenter les écoulements à surface libre en eaux peu profondes. Ces équations sont obtenues en intégrant sur la verticale les équations de Navier-Stokes, après avoir présumé une pression hydrostatique, des vitesses quasi-uniformes sur la verticale, et un fond et une surface libre imperméables. La forme non conservative de la formulation hauteur-vitesse comporte une équation de continuité et une équation de quantité de mouvement

$$\frac{\partial h}{\partial t} + U \,.\, \nabla h + h \; div U = 0$$

$$\frac{\partial U}{\partial t} + (U \,.\, \nabla)\, U + g\, \nabla z - \frac{1}{h}\, div(k\, h\, \nabla U) = F$$

h : hauteur d'eau
$U = (u, v)^\top$: vitesse
k : coefficient pour la dispersion due à la turbulence et aux hétérogénéités selon la verticale
g : pesanteur
$z = h + z_f$: cote de la surface libre
z_f : cote du fond
F : termes sources

Dans les deux équations, les seconds et troisièmes termes modélisent respectivement le transport par le courant et la propagation des ondes longues.
En négligeant la force de Coriolis, l'influence du vent, la pression atmosphérique et le potentiel astral, le terme source F se réduit au frottement du fond. Selon la loi de Strickler, il s'écrit

$$F = -\frac{g\, U\, |U|}{h^{4/3}\, S^2}$$

Dans le code Telemac2D, la résolution des équations est implémentée pour deux étapes principales, voir [3]. :

- Discrétisation des termes de convection (transport de h et U) par la méthode des caractéristiques (en option).

- Discrétisation de la propagation, de la diffusion et du terme de frottement (plus la convection si on n'utilise pas les caractéristiques) par un schéma semi-implicite en temps (θ-schéma) et par des éléments finis.

Pour une description détaillée des modèles ainsi que leurs discrétisations en temps et en espace, nous renvoyons à [3].

Pour chaque pas de temps, on obtient un système linéaire à résoudre : $A\, X = B$

$$A = \begin{pmatrix} AM1 & BM1 & BM2 \\ CM1^\top & AM2 & 0 \\ CM2^\top & 0 & AM3 \end{pmatrix} \quad , \quad X = \begin{pmatrix} h_j^{n+1} \\ u_j^{n+1} \\ v_j^{n+1} \end{pmatrix} \quad , \quad B = \begin{pmatrix} CV1 \\ CV2 \\ CV3 \end{pmatrix}$$

qui est résolu par des sous-itérations et grâce à la bibliothèque BIEF du L.N.H.E. par un solveur au choix.

3 Assimilation de données variationnelle en écoulement fluvial

L'assimilation de données (AD) peut être définie comme étant la combinaison la plus cohérente possible - dans un certain sens d'optimalité à préciser - des observations et mesures effectuées avec l'approximation numérique d'un modèle physique-mathématique, en vue d'améliorer les prédictions de ce modèle. Le développement des techniques d'AD doit son essor ces deux dernières décennies notamment à la recherche effectuée dans les domaines de la de la météorologie et de l'océanographie. L'AD permet aussi l'estimation des paramètres, des conditions initiales (CI) ou des conditions aux limites (CL). On distingues deux grandes familles de méthodes d'AD :

- Méthodes variationnelles, basées sur la théorie du contrôle optimal. L'optimisation est réalisée par une méthode de gradient dont la convergence peut nécessiter un grand nombre d'itérations. Le gradient étant obtenu dans le contexte de cette étude par la différentiation automatique (DA) du code direct qui fournit un code adjoint.

- Méthodes stochastiques, fondées sur la théorie de l'estimation stockastique optimale. Typiquement, la méthode du filtre de Kalman, qui elle requière une capacité de stockage prohibitive des matrices de covariance.

Ainsi, l'AD variationnelle se traduit par la minimisation d'une fonctionnelle mesurant l'écart entre la prévision et les observations

$$J = \frac{1}{2} \; (\; Y - Y_{obs} \; , \; Y - Y_{obs} \;)$$

où $Y \in \mathrm{R}^m$ représente les valeurs calculées par le modèle, correspondantes à l'ensemble des observations $Y_{obs} \in \mathrm{R}^m$.
Dans le modèle décrivant le système physique dynamique, on suppose que la dépendance de Y par rapport à un ensemble de variables de contrôle $X \in R^n$ peut être décrite par une certaine application assez régulière

$$\begin{array}{c} P : \mathrm{R}^n \longrightarrow \mathrm{R}^m \\ X \longmapsto Y \end{array}$$

Alors $\quad J(X) = \frac{1}{2} \; (\; P(X) - Y_{obs} \; , \; P(X) - Y_{obs} \;) \quad ,$
et le problème est de déterminer l'ensemble X qui minimise J; ce qui nécessite le calcul du gradient $\nabla_X J(X_o)$ de J en un point X_o donné. Le développement de Taylor de J au premier ordre donne

$$\begin{array}{lll} \delta J & = < \nabla_X J(X_o), \delta X > & , \quad \delta X = X - X_o \\ \delta J & = < P(X_o) - Y_{obs}, A(X_o) \; \delta X > & , \quad A(X_o) = \nabla P(X_o) \\ \delta J & = < A(X_o)^*(P(X_o) - Y_{obs}), \delta X > & \end{array}$$

D'où

$$\nabla_X J(X_o) = A(X_o)^*(P(X_o) - Y_{obs})$$

Les opérateurs linéaires $A(X_o)$ et $A(X_o)^*$ représentent respectivement les modèles linéaire tangent et adjoint; et le terme $P(X_o)-Y_{obs}$ peut être interprété comme un forçage du modèle adjoint.

L'implémentation du modèle adjoint s'avère donc indispensable pour la réalisation de l'AD. Et quoiqu'elle pourrait constituer un travail assez important en volume et en temps, elle apporte un grand bénéfice, vu encore que l'adjoint obtenu ne dépend pas des variables de contrôle, et ainsi servira aux différentes identifications envisagées.

Cas de l'hydraulique fluviale

Pour l'hydraulique fluviale, les variables de contrôle qui déterminent l'état de l'écoulement sont essentiellement les CL amont (débit entrant) et aval CL_N et CL_D, le coefficient de friction S, la cote du fond z_f ou encore la CI ou l'ébauche Y_o (voir par exemple [1]). Et on peut aussi paramétrer le modèle via l'incorporation d'une erreur systématique $B\,\eta$, cf par exemple [5]. Effectivement, on ne peut réaliser des simulations réelles sans une connaissance relativement précise de ces données. Toutefois, elles sont souvent acquises d'une manière partielle et incertaine (levés de bathymétrie sur des coupes ou encore des plaines d'inondation par télédétection). De plus, d'autres données ne peuvent pas être mesurées ou approchées correctement, comme la rugosité du lit. D'autres observations de l'écoulement peuvent être partiellement disponibles dans l'espace et dans le temps (hauteur, surface, vitesse par exemple). Il conviendrait alors de les confronter en tant que réalité hydraulique avec le modèle numérique de description.

Modèle adjoint

Pour un modèle général discrétisé en espace et résolu sur l'intervalle de temps $[0,T]$

$$\begin{cases} \dfrac{\partial Y}{\partial t} + F(Y,S,z_f) + B_D.CL_D + B_N.CL_N + B\,\eta = 0 \\ Y(0) = Y_o \end{cases}$$

Y : variable d'état
Y_o : ébauche
F : opérateur discret d'équations aux dérivées partielles (non linéaire)
S : coefficient de friction
z_f : cote du fond
$B_D.CL_D$: CL avale de type Direchlet
$B_N.CL_N$: CL amont de type Neumann
$B\,\eta$: erreur systématique du modèle

$X = (Y_o, S, z_f, CL_D, CL_N, \eta)^\top$ regroupe les variables de contrôle.

On introduit l'état adjoint $Y*$ défini comme solution du modèle adjoint

$$\left\{ \begin{array}{l} \dfrac{\partial Y^*}{\partial t} - [\dfrac{\partial F}{\partial Y}]^\top \; Y^* = C^t \; (C \; Y - Y_{obs}) \\ Y^*(T) = 0 \end{array} \right.$$

où C est un opérateur d'observation.

Alors on peut calculer le gradient de chacune des fonctions coûts ci-dessous, grâce à la méthode adjointe :

Minimisation par rapport à l'ébauche

$J(Y_o) = \frac{1}{2} \; \|CY(X) - Y_{obs}\|^2 + \frac{1}{2} \; \|Y_o - Y_o^g\|^2$

$\nabla_{Y_o} J = -Y^*(0)$

Minimisation par rapport aux CL

$J(CL) = \frac{1}{2} \; \|CY(X) - Y_{obs}\|^2 + \frac{1}{2} \; \|CL_D - CL_D^g\|^2 + \frac{1}{2} \; \|CL_N - CL_N^g\|^2$

$\nabla_{CL_D} J = B_D^\top . Y^* \quad , \quad \nabla_{CL_N} J = B_N^\top . Y^*$

Minimisation par rapport à la friction ou le fond

$J(K) = \frac{1}{2} \; \|CY(X) - Y_{obs}\|^2 + \frac{1}{2} \; \|K - K^g\|^2 \quad , \quad K = S \text{ ou } z_f$

$\nabla_K J = [\frac{\partial F}{\partial K}]^\top \; Y^*$

Minimisation par rapport à une erreur modèle

$J(\eta) = \frac{1}{2} \; \|CY(X) - Y_{obs}\|^2 + \frac{1}{2} < N\eta, \eta >$

$\nabla_\eta J = B^\top Y^* + N\eta$

Le superscript ^{g} indiquant une valeur d'initialisation, et N une pondération.

4 Différentiation automatique par Tapenade

La différentiation automatique (DA) propose l'informatisation de l'écriture de la dérivée et de l'adjoint d'un code de calcul. Cela représente une alternative à la différenciation mathématique pour ensuite implémenter leurs discrétisations. A l'aide de la DA, il est maintenant possible sous certaines conditions de produire le code dit tangent ou adjoint à partir du code direct.

Une formalisation de la DA d'un code $P_\#$
peut être traduite de la manière suivante :

Soit $X_\# = (Y_{o\#}, S_\#, z_{f\#}, CL_{D\#}, CL_{N\#}, \eta_\#)^\top$ l'ensemble des variables de contrôle discrètes, et $Y_\# = (h_\#, u_\#, v_\#)^\top$ la variable d'état discrète. Alors on peut schématiser les codes :

direct $P_\#$,
linéaire tangent $P_\#\_d = (\partial P_\#/\partial X_\#)$ et **adjoint** $P_\#\_b = (\partial P_\#/\partial X_\#)^*$ par

$$X_\# \longrightarrow P_\# \longrightarrow (Y_\#, cost)^\top$$

$$d\ X_\# \longrightarrow \frac{\partial P_\#}{\partial X_\#} \longrightarrow d\ (Y_\#, cost)^\top$$

$$d\ (Y_\#^*, cost^*)^\top \longrightarrow (\frac{\partial P_\#}{\partial X_\#})^* \longrightarrow d\ X_\#^*$$

et on a : $< d\ (Y_\#^*, cost^*)^\top, d\ (Y_\#, cost)^\top > = < d\ X_\#^*, d\ X_\# >$

Alors pour $d\ (Y_\#^*, cost^*)^\top = (0,1)^\top$, on obtient les dérivées partielles du coût *cost* par rapport aux variables de contrôle $X_\#$

$$\frac{\partial\ cost}{\partial (X_\#)_i} = (d\ X_\#^*)_i$$

Notre but principal est donc l'obtention du code adjoint représentant $(\partial P_\#/\partial X_\#)^*$ avec l'aide primordiale de Tapenade.

Tapenade est un logiciel de DA de programmes Fortran 77 et Fortran 95, développé par l'équipe Tropics de l'INRIA Sophia Antipolis. Il permet la génération des codes tangent (effet du Jacobien sur un vecteur) et adjoint (effet du transposé du Jacobien sur un vecteur). Si on présume que chaque séquence ou instruction I_k implémente une fonction vectorielle différentiable f_k, alors on peut identifier le programme $P_\#$ à

$$F = f_p\ o\ f_{p-1}\ o \ldots o\ f_1$$

Formellement, la DA applique une règle de chaîne pour calculer les dérivées de F : si on note X_k les valeurs de toutes les variables après l'instruction I_k ($X_k = f_k(X_{k-1})$), alors la règle de chaîne fournit le Jacobien de F:

$$F'(X) = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times \ldots \times f'_1(X_o)$$

ce qui peut être retraduit à une suite d'instructions I'_k qui, insérées dans le programme direct, fournissent le linéaire tangent $P_\#\_d$. En terme de mémoire, l'évaluation et le stockage du Jacobien peuvent s'avérer très chers. Mais heureusement, en pratique, on n'a besoin que de certaines sensibilités ou dérivées directionnelles

$$F'(X)\ X' = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times \ldots \times f'_1(X_o)\ X'$$

qui est calculée de droite à gauche par des produits matrice-vecteur.
Quant aux formulations adjointes, on a souvent besoin du transposé du Jacobien

$$F'^*(X)\ Y = f_1'^*(X_o) \times f_2'^*(X_1) \times \ldots \times f_p'^*(X_{p-1})\ Y$$

Or l'évaluation de l'adjoint $P_{\#}\_b$ ici requiert les X_k dans l'ordre inverse de leur calcul; d'où la nécessité de leur stockage. Tapenade utilise pour ce faire la technique de tout-stocker "Store-All", et l'ensemble des variables stockées est dit trajectoire.

On illustre ci-dessous la différentiation en modes directe et inverse
d'une instruction $\mathbf{I_k}$: $a(i) = x * b(j) + cos(a(i))$
Pour $X = (a(i), b(j), x)^\top$,

$$f'_k(X) = \begin{pmatrix} -sin(a(i)) & x & b(j) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad , \quad f_k'^*(X) = \begin{pmatrix} -sin(a(i)) & 0 & 0 \\ x & 1 & 0 \\ b(j) & 0 & 1 \end{pmatrix}$$

$\mathbf{I'_k}$: $ad(i) = xd * b(j) + x * bd(j) - ad(i) * sin(a(i))$

$\mathbf{I'^*_k}$: $xb = xb + b(j) * ab(i)$
$bb(j) = bb(j) + x * ab(i)$
$ab(i) = -sin(a(i)) * ab(i)$

Pour plus de détails sur les fondements et la pratique de Tapenade, nous nous référons à [4].

Actuellement, les outils de DA n'ont pas encore atteint le niveau de développement qui permet une automatisation complète de la différentiation. En effet, il y a des limitations concernant le traitement de certaines extensions des languages utilisés, comme ANSI-C et Fortran 77. Pourtant, les logiciels de DA disponibles arrivent à réduire l'effort de différentiation manuelle d'une manière significative. De plus, le progrès dans la couverture des nouvelles fonctionnalités de programmation devient de plus en plus rapide, grâce à la communication quotidienne entre les développeurs et les utilisateurs potentiels. Tapenade figure parmi les logiciels de DA les plus utilisés. Tapenade peut être utilisé soit localement après téléchargement, soit via le serveur de l'INRIA. C'est pratiquement la ressource de DA la plus accessible et sans restrictions pour des buts de recherche. A cela s'ajoute la flexibilité de manipulation de son interface graphique web.

Exemple pratique

Ici, nous donnons un exemple de DA en formulant une abstraction du genre de problème que nous voulons différentier par Tapenade. Soit alors le système simple suivant de solution présumée $Y = (h, u)^\top \in \mathrm{R}^2 \times [0, T]$

$$\begin{cases} \dfrac{\partial h}{\partial t} + 2\ h\ u = 0 \\ \dfrac{\partial u}{\partial t} + (1 + \dfrac{g}{S^2})\ u^2 + g\ (1 + z_f)\ h = 0 \end{cases}$$

qui peut être écrit sous forme

$$\frac{\partial Y}{\partial t} + A(Y)\ Y = 0$$

avec

$$A(Y) = \begin{pmatrix} 2\,u & 0 \\ g\,(1+z_f) & (1+\frac{g}{S^2})\,u \end{pmatrix}$$

On fait une discrétisation en temps de type Euler, et on suppose que la non linéarité peut être résolue par un schéma de substitution. On serait donc amener à résoudre numériquement le problème par une routine `run_test` qui appelle une routine `solve`. On renvoit à l'annexe 0 pour le listing de ces routines avec celles différentiées par Tapenade dans les deux modes direct et inverse. Le `Makefile` utilisé pour la DA, avec $h0$, $u0$, S et zf comme variables indépendantes, et h et u comme variables dépendantes, est le suivant

```
b:
    tapenade -b -head run_test \
    -inputfiletype fortran95 -outputlanguage fortran95 \
    -outvars "h u" -vars "h0 u0 S zf" \
    -O backward -html run_test.f90
d:
    tapenade -d -head run_test \
    -inputfiletype fortran95 -outputlanguage fortran95 \
    -outvars "h u" -vars "h0 u0 S zf" \
    -O forward -html run_test.f90
```

* Dans `run_test`, pour les deux instructions :

```
a = 1 + 2 * dt * upm1
d = 1 + dt * ( 1 + g / (S*S) ) * upm1
```

si on note $X = (a, d, u_p^{-1}, S)^\top$, on a

$$f'(X) = \begin{pmatrix} 0 & 0 & 2\,dt & 0 \\ 0 & 0 & (1+g/S^2)\,dt & -2\,dt\,g\,u_p^{-1}\,/\,S^3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$f'(X)^* = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2\,dt & (1+g/S^2)\,dt & 1 & 0 \\ 0 & -2\,dt\,g\,u_p^{-1}\,/\,S^3 & 0 & 1 \end{pmatrix}$$

Ce qui correspond bien aux instructions adjointes fournies par Tapenade :

```
temp0 = s**2
temp = g/temp0
sb = sb - dt*upm1*temp*2*s*db/temp0
upm1b = upm1b + dt*2*ab + dt*(1+temp)*db
```

* Dans `solve`, pour les trois instructions :

```
det = a * d
hp = d * hpm1 / det
up = ( -c * hpm1 + a * upm1 ) / det
```

si on note $X = (h_p, u_p, d, u_p^{-1}, a, det, h_p^{-1}, c)^\top$, on a

$$f'(X) = \begin{pmatrix} 0 & 0 & h_p^{-1}/det & 0 & 0 & -d\,h_p^{-1}/det^2 & d/det & 0 \\ 0 & 0 & 0 & a/det & u_p^{-1}/det & (c\,h_p^{-1} - a\,u_p^{-1})/det^2 & -c/det & -h_p^{-1}/det \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & a & 0 & d & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$f'(X)^* = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_p^{-1}/det & 0 & 1 & 0 & 0 & a & 0 & 0 \\ 0 & a/det & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & u_p^{-1}/det & 0 & 0 & 1 & d & 0 & 0 \\ -d\,h_p^{-1}/det^2 & (c\,h_p^{-1} - a\,u_p^{-1})/det^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ d/det & -c/det & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -h_p^{-1}/det & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Ce qui correspond bien aux instructions adjointes fournies par Tapenade :

```
tempb = upb/det
cb = cb - hpm1*tempb
tempb0 = hpb/det
hpm1b = d*tempb0 - c*tempb
detb = -(d*hpm1*tempb0/det) - (-(c*hpm1)+a*upm1)*tempb/det
ab = d*detb + upm1*tempb
upm1b = a*tempb
db = a*detb + hpm1*tempb0
```

* Une révision des `PUSH` et `POP` permet d'enlever le `CALL PUSHREAL8(c)` et `CALL POPREAL8(c)` car `c` est une constante.

* On enlève aussi `CALL PUSHINTEGER4(p - 1)` et `CALL PUSHINTEGER4(k - 1)` avec les deux `CALL POPINTEGER4(ad_to)` correspondants, et on remplace les `ad_to` par `pit` et `nit`.

Test du produit scalaire

Pour prouver la validité numérique des codes linéaire tangent et adjoint fournis par Tapenade, nous utilisons le test du produit scalaire. Pratiquement, il est souhaitable de créer un sous répertoire, y copier toute l'arborescence des routines différentiées en modes direct et adjoint, et le considérer comme répertoire de test après l'opération de DA. Ci-dessous, nous illustrons l'algorithme du test du produit scalaire pour `run_test`.

initialiser : $h0$, $u0$, S , zf

initialiser : $h0d$, $u0d$, Sd , zfd

initialiser : h , u

calculer hd , ud par $run\_test\_d(h0, h0d, u0, u0d, s, sd, zf, zfd, h, hd, u, ud)$

affecter : $hb = hd$, $ub = ud$

calculer $h0b$, $u0b$, Sb , zfb par $run\_test\_b(h0, h0b, u0, u0b, s, sb, zf, zfb, h, hb, u, ub)$

vérifier l'égalité : $< (h0d, u0d, Sd, zfd), (h0b, u0b, Sb, zfb) >=< (hd, ud), (hd, ud) >$

5 Préparation du code direct: Telemac2D-v5.4-simple

Le module Telemac2D fait partie du système global de modélisation fluviale et maritime Telemac développé par le LNHE-EDF et commercialisé par Sogreah. Il traite essentiellement de la simulation des écoulements surfaciques des eaux et transports de polluants, plus une possibilité de couplage avec le module de sédimentation Sisyphe. En principe, le codage d'un logiciel industriel destiné à la DA doit respecter un cahier de normes de programmation, ce qui n'est pas le cas de Telemac2D. De plus Telemac2D utilise une programmation par pointeurs non encore compris par Tapenade v2.0. De plus, le code est volumineux: plus de 108 000 lignes code Fortran 90. C'est dans ce contexte que nous tentons de produire une version exploitable dans le cadre de la DA par Tapenade v2.0. Nous présentons ici le code Telemac2D-v5.4-simple, qui d'un côté réduit considérablement la taille mémoire pour le traitement par Tapenade, et d'un autre côté garde tous les aspects physiques et numériques utiles à la modélisation en hydraulique fluviale.

Les simplifications physiques et numériques principales ont concerné :

- Le couplage avec le module de sédimentation Sisyphe.
- Les traceurs et les points de rejets.
- Les CL maritimes concernant la météo et la force de Coriolis.
- Les seuils, siphons et flotteurs.
- Le modèle de turbulence $K - \epsilon$.
- La parallélisation.

- La discrétisation par volumes finis et de l'équation de Boussinesq.
- L'utilisation des coordonnées sphériques.
- Quelques schémas de traitement de la forme de convection.
- Quelques solveurs des systèmes linéaires.

Ceci a impliqué une réduction du nombre des lignes à seulement 63300 et de l'arborescence à environ 600 appels non répétés au lieu de 1000. Les aspects de simulation et d'utilisation retenus dans Telemac2D-v5.4-simple, et qui se rapportent à l'hydrodynamique de l'écoulement bidimensionnel fluvial, sont :

- Equations de Saint-Venant non conservatives.
- Discrétisation par éléments finis P1 et/ou P1 bulles pour h, u et v.
- Friction du fond suivant la loi de Strickler.
- Modélisation de la turbulence par une viscosité constante.
- Simulation des bancs découvrants et plaines inondables.
- Quelques routines utilisateurs concernant les CL et CI.
- Suite de simulation à partir d'un état calculé.
- Contrôle d'état stationnaire et de vraissemblance.
- Calcul des bilans de masse.
- Schémas de convection : caractéristiques, SUPG et conservatif.
- Solveurs linéaires : GMRES et Gradient Conjugué Carré Stabilisé.

Restructuration en vue de la DA

Nous avons commencé par incorporer quelques pré-traiements généraux au code qui aiderons à la DA, tout en conservant une compilation et exécution Fortran efficace.

a - Déclaration des variables :

Les variables globales sont réparties sur deux nouveaux modules. Dans le premier on garde toutes celles qui restent logiquement inactives dans le processus de DA (variables entières, logiques, chaînes de caractères, variables liées au maillage et à la configuration du solveur). Et dans le second on déclare toutes les autres variables restantes, dont la plupart sont *a priori* jugées devenir actives au cours de la DA.

b - Variables de contrôle :

Nous avons du informer correctement les entrées en arguments des variables de contrôle dans la routine principale à différentier. Les conditions aux limites sont les plus délicates à préparer, vu qu'on a besoin de leurs valeurs en chaque pas de temps avant la simulation.

Ainsi, nous avons implémenté une routine qui s'occupe de remplir les valeurs du débit imposé en amont et de la cote imposée en aval en chaque pas de temps.

c - Mode libre avec écriture d'observations :

Nous avons mis l'épine dorsale de calcul de Telemac2D (quelques appels plus la boucle en temps entière) dans chacune des des routines principales se rapportant au mode libre ou imposé. Ici le mode libre signifie un simple calcul direct avec création d'un jeu d'observations si souhaitée.

d - Mode imposé avec lecture d'observations et calcul du coût :

La routines principale du mode imposé est celle effectivement concernée par la DA, et utilisée dans l'organigramme d'optimisation. Elle comporte presque les mêmes appels que celle du mode libre, sauf pour la lecture des observations au lieu de leur écriture, et le calcul d'une fonction coût.

Validation

Afin de valider cette version simplifiée du code original, nous avons effectué plusieurs tests numériques tirés du document de validation de Telemac2D, dont nous présentons les résultats ci-dessous. Ces tests ont été concluants. Pour plus de détails, nous renvoyons au manuel d'utilisation de Telemac2D-v5.4-simple fourni en annexe.

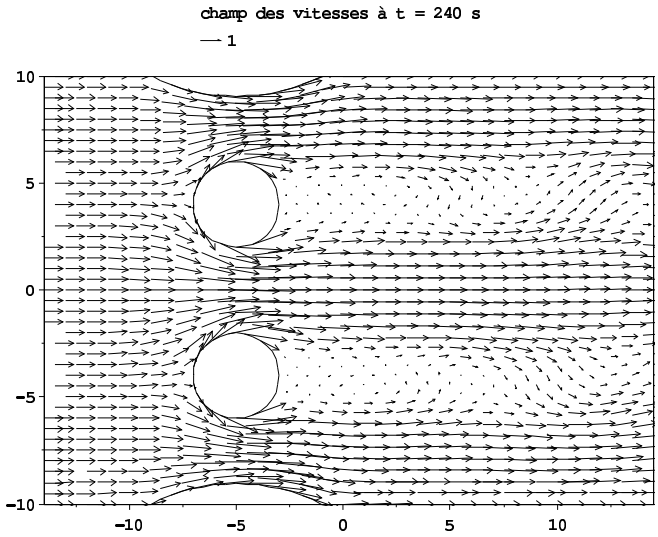


Figure 1: Champs de vitesses et tourbillons de Karmann autour de deux piles de pont

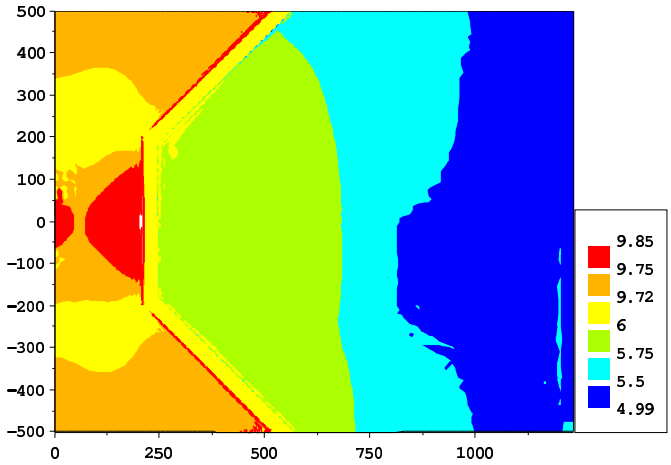


Figure 2: Surface de l'eau après débordement d'une digue (t = 2000 s)

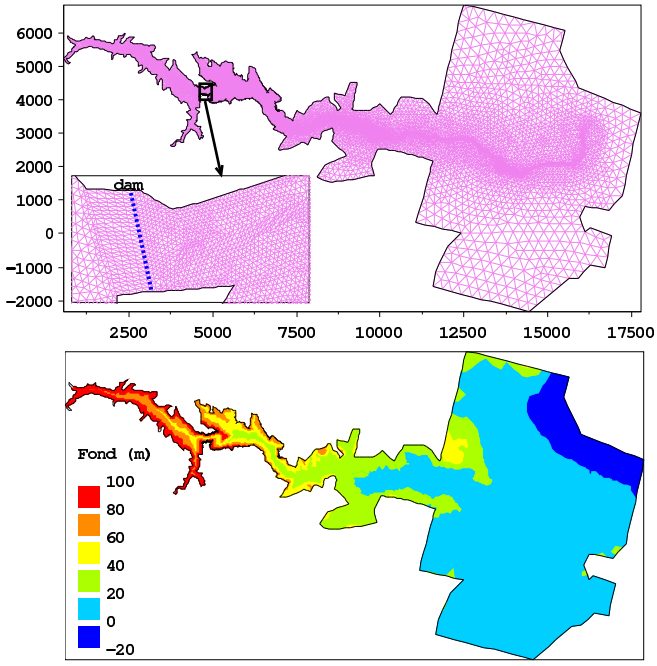


Figure 3: maillage et fond du modèle Malpasset

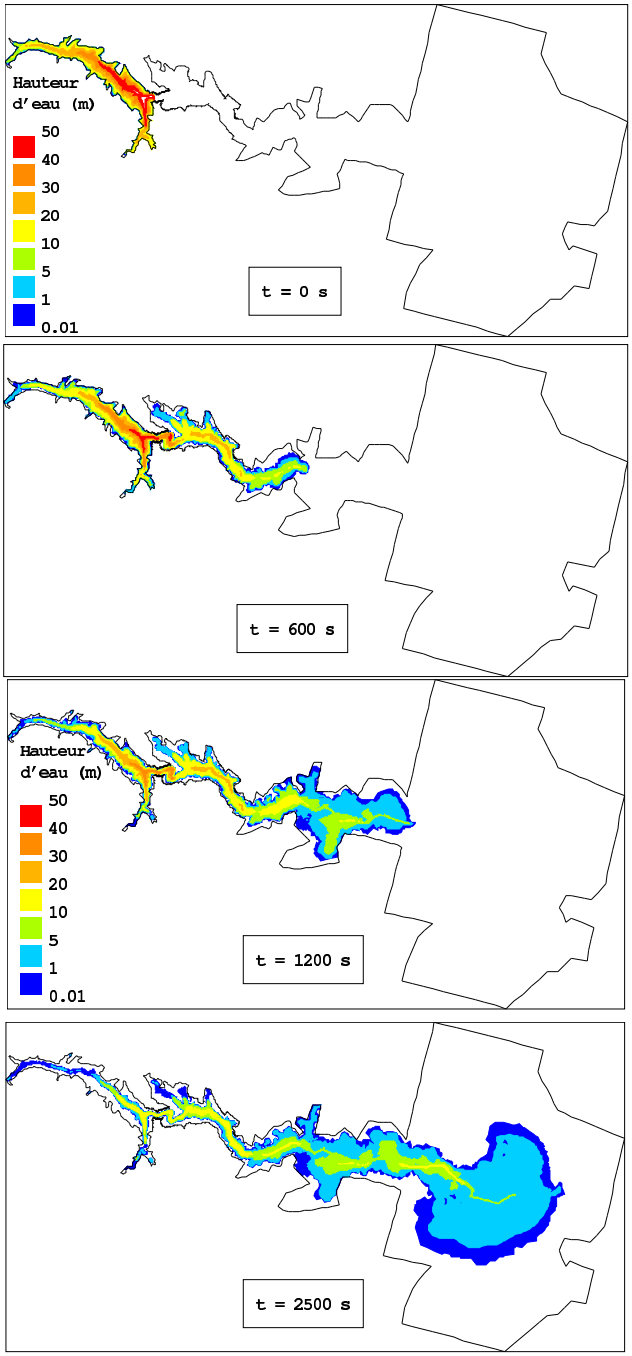


Figure 4: Simulation de la catastrophe après le bris du barrage

6 Différenciation automatique de Telemac2D-v5.4-simple

6.1 Problématique de DA par Tapenade

Comme mentionné précédemment, la taille du code Telemac2D-v5.4-simple est réduite au mieux mais reste volumineuse et avec une arborescence assez complexe et des appels multiples de certaines routines de calcul de base. Ceci évoque déjà des réflexions sur l'infaisabilité de DA du code entier due à la taille mémoire prohibitive exigée, mais aussi aux confusions prévues d'optimisation et de validation du code adjoint généré.

De plus, il y a occurrence de certains aspects de programmation Fortran 90 qui restent non complètement traités par Tapenade :

Pointeurs :

```
TYPE(BIEF_OBJ) ,  POINTER  ::  T1 , T2 , T3
```

Cibles :

```
TYPE(BIEF_OBJ) ,  INTENT(IN) , TARGET  ::  IFAMAS
```

Pointages :

```
T1    =>  TB%ADR( 1)%P
IFA   =>  MESH%IFABOR%I
```

Allocations dynamiques :

```
C      ALLOCATION DU TABLEAU DE POINTEURS ADR
C
       BLO%MAXBLOCK = 128
       ALLOCATE ( BLO%ADR(BLO%MAXBLOCK) , STAT=ERR )
```

Types structurés récursifs :

```
TYPE BIEF_OBJ
     INTEGER KEY
     INTEGER TYPE
     . . .
     . . .
     TYPE(BIEF_OBJ), POINTER :: D
     . . .
     . . .
END TYPE BIEF_OBJ
```

Arguments optionnels :

```
LOGICAL, INTENT(IN), OPTIONAL :: TRANS
```

Contrôle de visibilité :

```
USE BIEF, EX_CPSTMT => CPSTMT
```

Max et min :

Tapenade ne traite pas plus de deux arguments.

Certains Alias :

Tapenade peut générer des problèmes d'aliasing après la DA.

Variables globales actives :

Dans ses premières versions, Tapenade ne les gèrent pas correctement.

Tableaux en arguments :

Leurs tailles restent à spécifier après la DA.

Instructions d'affichage :

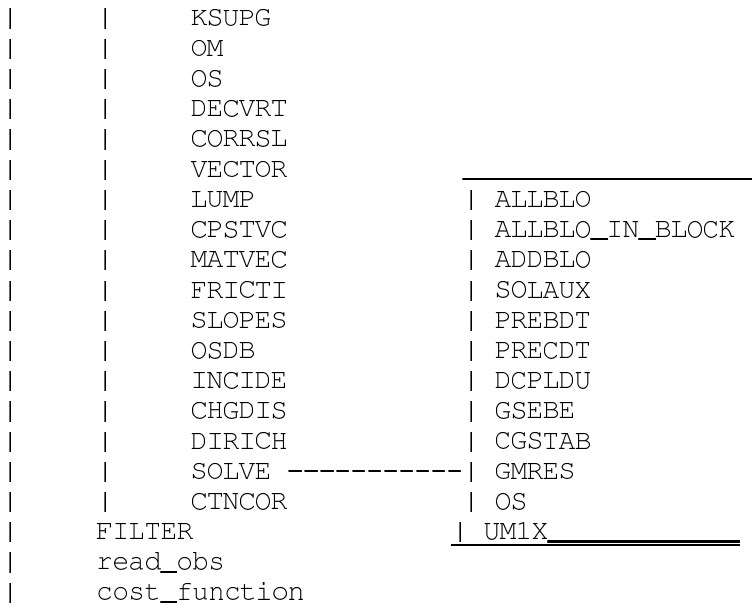
Ne sont pas différentiées.

6. 2 Stratégie générale de DA

Dans cette section, nous éclairons les différentes étapes de notre plan de DA de Telemac2D-5.4-simple. Nous rappelons que le code a subi une restructuration pour la DA, qui est expliquée dans son guide de programmation. Ainsi, Telemac2D-5.4-simple comporte déjà deux modes principaux (libre et imposé), qui lui permettent respectivement, soit de faire des simulations ordinaires d'écoulement, soit de passer pour un simulateur dans un processus d'optimisation, qui calcule aussi naturellement une fonction coût. C'est donc la routine `run_imposed` implémentant le mode imposé qui va être concernée par la DA.

Arborescence simplifiée de `run_imposed`

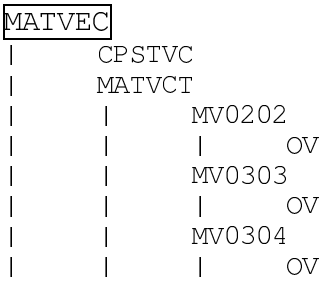
```
run_imposed
|     OS
|     CELERITE
|     CHGDIS
|     CLIP
|     PROPIN_TELEMAC2D
|     COEFRO
|     ENTETE
|     HPROPA
|     CHPCON
|     BORD
|     |     DEBIMP
|     |     |     VECTOR
|     CHARAC
|     GESTIO
|     CPSTVC
|     PROPAG
|     |     OSBD
|     |     MATRIX
```



Evidemment, il y a des difficultés diverses qui pourront accompagner toute tentative de DA globale de `run_imposed`, et qui se rapportent à la taille du code et aux limitations de traitement du Fortran 90 par Tapenade, listées dans la section précédente. Mais d'autres problèmes plus sérieux qui sont subtiles d'en résulter, se rapportent à la complexité de post-traitement, optimisation et validation du code adjoint. De ce fait, nous avons opter pour la DA par morceaux de code. Et justement, il y a quelques routines qui assurent des calculs élémentaires ou de base pour le code. Ces routines sont souvent appelées plusieurs fois , et leurs propres arborescences sont assez importantes. La DA de chacune d'entre elles indépendamment reste relativement moins fastidieuse dans toutes les étapes, du pré-traitement à la validation.
Néanmoins, même si Tapenade prévoit la compilation de routines en *boites noires*, les types structurés en arguments ne sont pas traités. Or c'est le cas pour les routines que nous projetons différentier. Mais cela ne pose pas une vraie barrière à notre démarche, et nous en remédions en construisant des routines jumelles qui ont les mêmes arguments et qui n'ont pas d'arborescence. Nous les appelons *boites blanches* et nous les utiliserons dans la DA des routines mères. Une fois ces dernières différentiées, nous n'aurons qu'à substituer les boites blanches et leurs adjoints par les routines effectives proprement différentiées auparavant.

Exemples de routines de base :

`OS` : assure différentes opérations sur les vecteurs.
`OM` : assure différentes opérations sur les matrices.
`VECTOR` : assure des calculs de vecteurs.
`MATRIX` : assure des calculs de matrices.
`MATVEC` : assure différentes opérations matrice vecteur.
`CGSTAB` : résolution du système linéaire par gradient conjugué stabilisé.
`GMRES` : résolution du système linéaire par GMRES.



```
|      |      MV0404
|      |      |     OV
|      |      MV0403
|      |      |     OV
|      |      ASSVEC
|      |      |     OV
|      |      |     ASSVE1
|      |      MW0303
|      |      |     OPASS
|      |      |     OV
|      |      MVSEG
|      |      |     OV
|      OS
```

Etape de pré-traitement

Nous avons mené un pré-traitement global sur tout le code Telemac2D-v5.4-simple qui réalise principalement des contournements de Tapenade en ce qui concerne ses limitations pour le Fortran 90. Ensuite, pour chaque routine de base, nous faisons un pré-traitement propre à la partie du code correspondante, pour qu'on puisse réussir la DA par Tapenade. Il est à noter alors que tous les codes résultant ne peuvent pas être directement compilés pour le Fortran, et donc requièrent une autre étape de post-traitement.

Pointeurs :

Nous commentons les lignes du code déclarant les pointeurs et nous les remplaçons en effaçant le mot clé `POINTER` :

```
C     TYPE(BIEF_OBJ) ,  POINTER  ::  T1 , T2 , T3
      TYPE(BIEF_OBJ)             ::  T1 , T2 , T3
```

Cibles :

Nous commentons les lignes du code déclarant les cibles et nous les remplaçons en effaçant le mot clé `TARGET` :

```
C     TYPE(BIEF_OBJ) ,  INTENT(IN) , TARGET  ::  IFAMAS
      TYPE(BIEF_OBJ) ,  INTENT(IN)           ::  IFAMAS
```

Pointages :

Nous commentons les lignes du code implémentant les pointages et nous les remplaçons par des simples affectations :

```
C     T1    =>  TB%ADR( 1)%P
      T1    =   TB%ADR( 1)%P
```

Allocations dynamiques :

Nous avons déjà éviter la plupart des allocations de mémoire qui sont heureusement antérieures à la partie du code à différentier, i.e. `run_imposed`. Toutefois, il reste encore des déclarations allocate dans la routine solve. Evidemment, c'est à commenter ; mais aussi à pré-traiter les dimensions des tableaux et blocks concernés.

Type structuré `bief_obj` :

Ce type constitue en fait la structure de base de la bibliothèque d'éléments finis du LNH-EDF. Les variables `bief_obj` peuvent être des vecteurs réels ou entiers à une ou deux dimensions, des matrices, des blocks de vecteurs et de matrices, ou aussi des blocks de blocks. Il y a des champs qui précisent le type, la nature ou le nom de l'objet ; et d'autres qui assurent son stockage.

```
----------------------------------------------------------------------
      TYPE BIEF_OBJ
      SEQUENCE
            INTEGER KEY
            INTEGER TYPE
            CHARACTER(LEN=6) NAME
C     FOR VECTORS
            INTEGER NAT
            INTEGER ELM
            INTEGER DIM1
            INTEGER MAXDIM1
            INTEGER DIM2
            INTEGER MAXDIM2
            INTEGER DIMDISC
            INTEGER STATUS
            DOUBLE PRECISION, pointer, DIMENSION(:)::R
            INTEGER, POINTER,DIMENSION(:)::I
C     FOR MATRICES
            INTEGER STO
            INTEGER ELMLIN
            INTEGER ELMCOL
            CHARACTER*1 TYPDIA
            CHARACTER*1 TYPEXT
            TYPE(BIEF_OBJ), POINTER :: D
            TYPE(BIEF_OBJ), POINTER :: X
C     FOR BLOCKS
            INTEGER N
            INTEGER MAXBLOCK
            TYPE(POINTER_TO_BIEF_OBJ),POINTER,DIMENSION(:) :: ADR
      END TYPE BIEF_OBJ
C -------------------------------------------------------------------
      TYPE POINTER_TO_BIEF_OBJ
      SEQUENCE
            TYPE(BIEF_OBJ), POINTER :: P
      END TYPE POINTER_TO_BIEF_OBJ
----------------------------------------------------------------------
```

Le problème ici est que une fois les pointeurs éliminés, on se trouve avec des imbrications du type `bief_obj`. Pour régler cette situation, nous déclarons deux autres types structurés jumeaux `bief_obj_2` et `bief_obj_3` et nous redéfinissons `bief_obj` et `pointer_to_bief_obj` :

```
      TYPE BIEF_OBJ_2
      SEQUENCE
            INTEGER KEY
            INTEGER TYPE
            CHARACTER(LEN=6) NAME
C     FOR VECTORS
            INTEGER NAT
            INTEGER ELM
            INTEGER DIM1
            INTEGER MAXDIM1
            INTEGER DIM2
            INTEGER MAXDIM2
            INTEGER DIMDISC
            INTEGER STATUS
            DOUBLE PRECISION, DIMENSION(:)::R
            INTEGER, DIMENSION(:)::I
```

```
      END TYPE BIEF_OBJ_2
      TYPE BIEF_OBJ_3
      SEQUENCE
            INTEGER KEY
            INTEGER TYPE
            CHARACTER(LEN=6) NAME
C     FOR VECTORS
            INTEGER NAT
            INTEGER ELM
            INTEGER DIM1
            INTEGER MAXDIM1
            INTEGER DIM2
            INTEGER MAXDIM2
            INTEGER DIMDISC
            INTEGER STATUS
            DOUBLE PRECISION, DIMENSION(:)::R
            INTEGER, DIMENSION(:)::I
C     FOR MATRICES
            INTEGER STO
            INTEGER ELMLIN
            INTEGER ELMCOL
            CHARACTER*1 TYPDIA
            CHARACTER*1 TYPEXT
            TYPE(BIEF_OBJ_2) :: D
            TYPE(BIEF_OBJ_2) :: X
      END TYPE BIEF_OBJ_3
C -----------------------------------------------------------------------
      TYPE POINTER_TO_BIEF_OBJ
      SEQUENCE
            TYPE(BIEF_OBJ_3) :: P
      END TYPE POINTER_TO_BIEF_OBJ
C -----------------------------------------------------------------------
      TYPE BIEF_OBJ
      SEQUENCE
            INTEGER KEY
            INTEGER TYPE
            CHARACTER(LEN=6) NAME
C     FOR VECTORS
            INTEGER NAT
            INTEGER ELM
            INTEGER DIM1
            INTEGER MAXDIM1
            INTEGER DIM2
            INTEGER MAXDIM2
            INTEGER DIMDISC
            INTEGER STATUS
            DOUBLE PRECISION, DIMENSION(:)::R
            INTEGER, DIMENSION(:)::I
C     FOR MATRICES
            INTEGER STO
            INTEGER ELMLIN
            INTEGER ELMCOL
            CHARACTER*1 TYPDIA
            CHARACTER*1 TYPEXT
            TYPE(BIEF_OBJ_2), POINTER :: D
            TYPE(BIEF_OBJ_2), POINTER :: X
C     FOR BLOCKS
            INTEGER N
            INTEGER MAXBLOCK
            TYPE(POINTER_TO_BIEF_OBJ), DIMENSION(:) :: ADR
      END TYPE BIEF_OBJ
C -----------------------------------------------------------------------
```

Arguments optionnels :

Nous éliminons les mots clés `OPTIONAL` de façon que toutes les variables arguments des routines soient présentes dans tous les appels. Ainsi, des retouches logiques sont nécessaires cas par cas, surtout dans les routines elles mêmes.

```
----------------------- Exemple ----------------------------
SUBROUTINE MATVEC( OP , X , A , Y , C , MESH , LEGO )
C      LOGICAL          , INTENT(IN), OPTIONAL :: LEGO
      LOGICAL          , INTENT(IN)           :: LEGO
C
C       IF(PRESENT(LEGO)) THEN
C         LEGO2 = LEGO
C       ELSE
C         LEGO2 = .TRUE.
C       ENDIF
C
      LEGO2 = LEGO
------------------------------------------------------------
```

Puis dans tous les appels sans l'argument optionnel `LEGO` on ajoute `.TRUE.` :

```
C      CALL MATVEC('X=AY    ', WORK2, S, WORK1, C, MESH )
      CALL MATVEC('X=AY    ', WORK2, S, WORK1, C, MESH, .TRUE. )
```

Contrôle de visibilité :

On écarte les renommages des ressources de modules lors de leurs utilisation :

```
C      USE BIEF, EX_CPSTMT => CPSTMT
      USE BIEF
```

Max et min :

On réécrit les appels de max et min avec seulement deux arguments par appel :

```
C      H = MAX(H1,H2,H3)
      H = MAX(MAX(H1,H2),H3)
```

Certains Alias :

En général, Tapenade affiche des messages d'avertissement lorsqu'il rencontre des cas d'alias non évidents. Dans notre démarche, nous restons conscient des problèmes qu'ils peuvent générer, et nous les traitons cas par cas.

Variables globales actives :

Lorsqu'on a entamé ce projet de DA, Tapenade posait encore des problèmes concernant la reconnaissance de variables actives déclarées dans les modules. Donc nous avons du passer toutes les variables assujetties d'être actives en arguments des routines qui les utilisent. Le point de départ ici était de partager les variables du module `declarations_telemac2d` sur deux nouveaux modules :

`* declarations_telemac2d_a` : Dans ce module, on garde toutes les variables globales qui restent logiquement inactives dans le processus de DA :

- # variables entières
- # variables logiques
- # chaînes de caractère

variable `mesh` de type `bief_mesh` (structure du maillage)

variables de type `slvcfg` (configuration du solveur)

`* declarations_telemac2d_b :` Ce deuxième module comporte toutes les autres variables restantes, dont a plupart sont *a priori* jugées devenir actives au cours de la DA.

variables réelles (double precision)
variables de type `bief_obj`

Ainsi, nous avons du restructurer la plupart des routines qui utilisent `declarations_telemac2d`, en y substituant par `declarations_telemac2d_a`, et en passant en paramètres toutes les variables actives utilisées parmi les arguments.

---------- **Exemple :** les ajouts sont exhibées en gras encadré ----------

```
SUBROUTINE  CELERITE  ( c0,h,grav )
C
      USE BIEF
      USE DECLARATIONS_TELEMAC2D_A
C
      IMPLICIT NONE
      INTEGER LNG,LU
      type(bief_obj)  ::  c0 , h
       double precision  ::  grav
     COMMON/INFO/LNG,LU
C-----------------------------------------------------------------
      CALL OSBD( 'X=CY    ' , C0 , H  , H , GRAV , MESH )
      CALL OS  ( 'X=+(Y,C)' , C0 , C0 , H , 0.D0        )
      CALL OS  ( 'X=SQR(Y)' , C0 , C0 , H , 0.D0        )
      RETURN
      END
-----------------------------------------------------------------
```

Tableaux en arguments :

Il s'avère qu'il faut réserver une attention particulière à la taille des tableaux passés en paramètres, surtout si cette taille et donc son allocation dépend d'une instruction conditionnelle. Un exemple de pré-traitement dans tel cas est de spécifier à l'avance une dimension du tableau suivant les conditions de calcul.
Un autre genre de pré-traitement consiste à remplacer le passage d'un tableau en paramètre en assimilant son premier élément à son adresse, par le tableau lui même :

```
C      CALL OV(OP, X%R(1), YY%R(1), ZZ%R(1), C, X%DIM1)
      CALL OV(OP, X%R, YY%R, ZZ%R, C, X%DIM1)

C      CALL OV( OP, X(1,DIMX), Y(1,DIMY), Z(1,DIMZ), C, NPOIN )
      CALL OV( OP, X(:,DIMX), Y(:,DIMY), Z(:,DIMZ), C, NPOIN )
```

Instructions d'affichage :

Nous les éliminons avec les appels à la routine de plantage `PLANTE` et les instructions `STOP`.

Etape de DA

Ici, nous décrivons la manipulation de l'outil de DA Tapenade qui demeure au cœur de notre stratégie globale, et donc vitale pour la génération des routines adjointes.

Nous utilisons une routine générique joha avec une arborescence quelconque comportant certains modules et routines : joha1 , joha2 , joha3 , joha4 , joha5 .
Nous voulons différentier joha(vd1,vd2,vi1,vi2,vi3,v1,v2) avec vd1 et vd2 sont les variables de sortie dépendantes, et vi1, vi2 et vi3 celles d'entrées indépendantes.
Pour compiler Tapenade sur joha en modes direct et adjoint, nous réalisons le makefile suivant :

```
------------------------ Makefile ----------------------

files = joha1  joha2  joha3  joha4  joha5  joha

b:
     tapenade -b -head joha \
          -inputfiletype fortran95 \
          -outputlanguage fortran95 \
          -outvars "vd1 vd2"
          -vars "vi1 vi2 vi3" \
          -O backward -html $(files)
d:
     tapenade -d -head joha \
          -inputfiletype fortran95 \
          -outputlanguage fortran95 \
          -outvars "vd1 vd2"
          -vars "vi1 vi2 vi3" \
          -O forward -html $(files)

-------------------------------------------------------
```

Ainsi, make d fournit le code représentant le linéaire tangent **joha_d** avec **joha1_d** , **joha2_d** , **joha3_d** , **joha4_d** et **joha5_d** ;

et make b réalise la routine de calcul de l'adjoint **joha_b** avec son arborescence **joha1_b** , **joha2_b** , **joha3_b** , **joha4_b** et **joha5_b** .

L'option **-O** permet de spécifier le répertoire d'enregistrement du code différentié.

Etape de post-traitement

Après la génération des routines différentiées en modes direct et adjoint par Tapenade, une autre tâche consiste à vérifier et corriger si nécessaire leur codage automatique. Mais tout d'abord, nous devons remettre les fonctionnalités du Fortran 90 commentées lors de l'étape de pré-traitement. Et pour les variables différentiées correspondant à des variables pointeurs ou cibles, il faut les redéclarer avec précaution. Les pointages des variables différentiées restent aussi à traiter cas par cas, tout en évitant les problèmes d'alias. Le but du post-traitement est d'apporter les retouches indispensables à la compilation des tests de validation, tout en tenant compte de l'environnement de programmation de Telemac2D. Evidemment, plusieurs exemples vont se manifester plus loin lors de la DA des routines de base. Sur un autre plan, Les retouches dépendent vraisemblablement des versions de Tapenade ; et nous restons en interaction continue avec l'équipe TROPICS qui le développe, afin d'automatiser au mieux cette tâche.

Pointeurs :

```
     TYPE(BIEF_OBJ) ,  POINTER  ::  T1 , T2 , T3
```

Cibles :

```
TYPE(BIEF_OBJ) ,  INTENT(IN) , TARGET  ::  IFAMAS
```

Pointages :

```
T1    =>  TB%ADR( 1)%P
```

Allocations dynamiques

Les remettre correctement avec celles des nouveaux tableaux différentiés.

Type structuré `bief_obj`

A remettre.

Contrôle de visibilité :

```
USE BIEF, EX_CPSTMT => CPSTMT
```

Tableaux en arguments :

```
CALL OV(OP, X%R(1), YY%R(1), ZZ%R(1), C, X%DIM1)
CALL OV( OP, X(1,DIMX), Y(1,DIMY), Z(1,DIMZ), C, NPOIN )
```

Instructions d'affichage :

Les remettre en cas de nécessité.

Etape d'optimisation

L'optimisation des routines différentiées vise en général la réduction de la taille mémoire réservée à la compilation de l'adjoint. Essentiellement, il faut bien étudier les empilements codés par Tapenade, pour pouvoir ensuite procéder à leur simplification avec les dépilements correspondants.

Exemple :

Extrait de code directement généré par Tapenade

Les empilements et dépilements éliminés sont bandés en gris, et les rectifications sont les lignes de code en gras. Nous notons ici que le tableau de la variable d'état n'est pas modifié dans la boucle sur les éléments ; ce qui permet de remplacer ses ''pushs'' et ''pops'' à l'extérieur de cette boucle. Ceci permet de réserver une taille mémoire de seulement `8*ntmax*3*nelt` au lieu de `8*ntmax*3*nelt**2` .

```
-----------------------------------------------------------------
DO WHILE (ts - tc .GT. eps)
      nt = nt + 1
      dof = doftm1
      DO kvol=1,nelt
            CALL PUSHREAL8ARRAY(dof, nelt*3)
            CALL CALC_FLUX(kvol, qin, hout, dof, resc, visco)
            CALL PUSHREAL8ARRAY(dof, nelt*3)
            CALL CALC_SOURCE(kvol, dof, gradz, man, so, sf)
            CALL PUSHREAL8(result1)
            result1 = surf(kvol)
```

```
              res(kvol, :) = (so+sf-resc+visco)/ result1 surf(kvol)
       END DO
       CALL PUSHREAL8ARRAY(dof, nelt*3)
       CALL PUSHINTEGER4(kvol - 1)
       dof = doftm1 + dt*res
       doftm1 = dof
       CALL READ_OBS(nt)
       CALL PUSHREAL8ARRAY(dof, nelt*3)
       CALL PUSHINTEGER4(nt)
       CALL COST_FUNCTION(nt, dof, tcost)
       cost = cost + tcost
       tc = tc + dt
       ad_count = ad_count + 1
END DO
CALL PUSHINTEGER4(ad_count)
!
ntmax = nt          !-----------------------------------------------------
!
CALL POPINTEGER4(ad_count)
DO i=1,ad_count
DO nt=ntmax,1,-1
       tcostb = costb
       CALL POPINTEGER4(nt)
       CALL POPREAL8ARRAY(dof, nelt*3)
       CALL COST_FUNCTION_B(nt, dof, dofb, tcost, tcostb)
       dofb = dofb + doftm1b
       doftm1b(1:nelt, 1:3) = 0.0
       doftm1b = dofb
       resb = resb + dt*dofb
       dofb(1:nelt, 1:3) = 0.0
       CALL POPINTEGER4(ad_to)
       CALL POPREAL8ARRAY(dof, nelt*3)
       DO kvol=ad_to,1,-1
       DO kvol=nelt,1,-1
              tempb = resb(kvol, :)/result1 surf(kvol)
              sob = sob + tempb
              sfb = sfb + tempb
              rescb = rescb - tempb
              viscob = viscob + tempb
              resb(kvol, :) = 0.0
              CALL POPREAL8(result1)
              CALL POPREAL8ARRAY(dof, nelt*3)
              CALL CALC_SOURCE_B(kvol,dof,dofb,gradz,man,so,sob,sf,sfb)
              CALL POPREAL8ARRAY(dof, nelt*3)
              CALLCALC_FLUX_B(kvol,qin,hout,dof,dofb,resc,rescb,visco,viscob)
       END DO
       doftm1b = doftm1b + dofb
       dofb(1:nelt, 1:3) = 0.0
END DO
-----------------------------------------------------------------------
```

Etape de validation

Maintenant, après toutes les étapes précédentes qui ont contribué à la réalisation de l'adjoint, il reste à prouver sa validité numérique. Nous utilisons alors le test du produit scalaire expliqué ci-dessous. Pratiquement, il est souhaitable de créer un sous répertoire `validation` pour y copier toutes les routines différentiées en modes direct et adjoint, et le considérer comme répertoire de travail juste après l'étape de DA.
Comme exemple générique, le programme de validation des codes joha_d et joha_b (voir étape de DA) implémente l'algorithme suivant :

pour `X = ( vi1 , vi2 , vi3 )` variables indépendantes

et `Y = ( vd1 , vd2 )` variables dépendantes

initialiser `X , Xd , Y`

calculer `Yd` par `joha_d(X,Xd,Y,Yd, . . . )`

affecter `Yd` à `Yb` : `Yb = Yd`

calculer `Xb` par `joha_b(X,Xb,Y,Yb, . . . )`

comparer les deux produits scalaires : `( Xd , Xb )` et `( Yd , Yd )`

qui doivent être égaux.

6.3 - Travail effectué

Dans le cadre de la stratégie globale de DA établie dans la section précédente, nous avons donc pu compléter l'essentiel des pré-traitements généraux sur le code :

* **Pointeurs, cibles et pointages :** pré-traités, mais à bien observer pour le post-traitement des parties de code non encore différentiées.
* **Allocations dynamiques :** pré-traitement par commentation réussi, et à suivre une fois rencontrées.
* **Type structuré** `bief_obj` **:** accommodé pour la DA grâce à deux types jumeaux intégrés.
* **Arguments optionnels :** pré-traités dans tout le code.
* **Contrôle de visibilité :** dans l'étape de validation, parfois il est nécessaire de le remettre comme post-traitement. Pensez aussi à le faire dans les routines différentiées pour assurer une bonne compilation.
* **Max et min :** pré-traités dans tout le code.
* **Certains alias :** à régler cas par cas si rencontrés. Mais il vaut mieux les détecter et les pré-traiter bien avant l'étape de différentiation automatique.
* **Variables globales actives :** pré-traitées pour assurer une DA correcte à travers la partition du module `declarations_telemac2d`.
* **Tableaux en arguments :** à pré-traiter au fur et à mesure, suivant la complexité des appels.

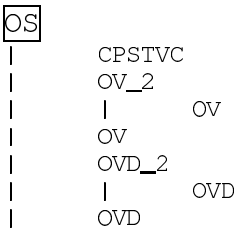
Ensuite, nous avons entamé la DA qui a réussi pour `os` et `vector`. Pour `om` , nous avons réalisé l'essentiel du travail de pré-traitement, et nous avons préparé le makefile et les fichiers pour la validation.

Pratiquement, nous avons organisé l'opération de DA de `run_imposed` dans un répertoire, à versions futures évolutives, nommé : `telemac_tap_v0p2/`
Et c'est dans ce répertoire que sont mises toutes les routines concernées par la DA (arborescence de `run_imposed`), plus un répertoire `whitebox/` qui regroupe d'autres sous-répertoires consacrés à la DA des routines de base (`da_os/` , `da_vector/` , `da_om/`, . . .etc). A chaque fois on entame la DA d'une partie du code `rout`, on transfert les fichiers arborescents concernés de `telemac_tap_v0p2/` vers `da_rout/` , on fait les pré-traitements supplémentaires nécessaires (à suivre le développement et mises à jour de Tapenade). Evidemment, c'est dans chaque `da_rout/` qu'on crée les répertoires `forward/` , `backward/` et `validation/` comme expliqué avant.

DA de `os.f`

Arborescence

```
OS ( OP , X , Y , Z , C , IOPT , INFINI , ZERO )
```



Pré-traitements

* Elimination de l'optionalité des arguments `X` , `Y` , `Z` , `C` , `IOPT` , `INFINI` et `ZERO` . Ainsi, rectification dans tous les appels à `OS` avec utilisation de certaines variables de travail appelées `bidon` pour `X` , `Y` ou `Z` de type `bief_obj` .

* Elimination des pointeurs. Exemple :

```
      IF (yay) THEN
            IF (present) THEN
! pré             YY => Y
                  yy%r = y%r
            END IF
      ELSE
! pré             YY => X
                  yy%r = x%r
      END IF
```

* Traitement pour l'appel de certains tableaux. Exemples :

* Appel à `OV` par `OV_2` :

```
!pré  CALL OV( OP , X(1,DIMX) , Y(1,DIMY) , Z(1,DIMZ) , C , NPOIN )
      CALL OV( OP , X(:,DIMX) , Y(:,DIMY) , Z(:,DIMZ) , C , NPOIN )
```

* Traitement du même ordre pour l'appel à `OVD` par `OVD_2` .

* Appel à `OV` par `OS` :

```
!pré  CALL OV( OP , X%R(1) , YY%R(1) , ZZ%R(1) , C , X%DIM1 )
      CALL OV( OP , X%R    , yy%R    , zz%R    , C , X%DIM1 )
```

Makefile de DA

```
b:
      tapenade -b -head os \
      -inputfiletype fortran95 -outputlanguage fortran95 \
      -outvars "x" -vars "y z c" -O backward -html \
      os.f ov.f ov_2.f ovd.f ovd_2.f cpstvc.f  bief.f bief_def.f cmpobj.f
d:
      tapenade -d -head os \
      -inputfiletype fortran95 -outputlanguage fortran95 \
      -outvars "x" -vars "y z c" -O forward -html \
      os.f ov.f ov_2.f ovd.f ovd_2.f cpstvc.f  bief.f bief_def.f cmpobj.f
```

Post-traitement et optimisation

* Reprise des pointeurs et pointage des variables différentiées. Exemple :

```
      IF (yay) THEN
            IF (present) THEN
                  YY  => Y
                  yyb => yb
                  CALL PUSHINTEGER4(0)
            END IF
      ELSE
            YY  => X
            yyb => yb
            CALL PUSHINTEGER4(1)
      END IF
```

Remarquez que les pointages des variables différentiées ne suis pas nécessairement la même logique de pointage des variables correspondantes.

* Taille des empilements. Exemple :

```
!     CALL PUSHREAL8ARRAY(zz%r(1), dim1*ISIZE1OFzz)
      CALL PUSHREAL8ARRAY(zz%r, zz%dim1*zz%dim2)
```

* Simplification des initialisations. Exemple :

```
      zzb%r(:) = 0.D0
      yyb%r(:) = 0.D0
```

au lieu des initialisations de tous les champs de `zzb` et `yyb`.

Validation

```
program test_ps
  use bief_def
  use decl
  IMPLICIT NONE
!
  print*,'******************************************************'
  print*,'*                tests sur les vecteurs a 1 dim        '
  print*,'******************************************************'
!
  dim1 = 10
  dim2 = 1
  call alloc
!
  iopt = 0
  infini = 10**10
  zero = 10**(-20)
  x%r = 1.d0
  y%r = (/(i*5.369874,i=1,dim1*dim2)/)
  z%r = 1.36987d0
  yd%r = 7.18846d0
  zd%r = 3.1547d0
  c=0.36
  cd=4.7154
!
  call os( 'X=Y     ' , xs , x , xs , c , iopt , infini , zero )
!
  .
  .
  call test('X=X+YZ  ')
  .
  .
  print*,'******************************************************'
  print*,'*                tests sur les vecteurs a 2 dim        '
  print*,'******************************************************'
```

```
!
  dim1 = 5
  dim2 = 3
  call alloc
!
  x%r = 1.d0
  y%r = 5.12354d0
  z%r = 1.36987d0
  yd%r = 7.18846d0
  zd%r = 3.1547d0
  c=0.36
  cd=4.7154
!
  call os( 'X=Y     ' , xs , x , xs , c , iopt , infini , zero )
!
  .
  .
  call test('X=CXY   ')
  .
  .
  print*,'*****************************************************'
  print*,'*                      tests sur les blocs          *'
  print*,'*****************************************************'
!
  dim1 = 5
  dim2=1
  call alloc_bloc
!
  c=0.36
  cd=4.7154
!
  call os( 'X=Y     ', xs , x , xs , c , iopt , infini , zero )
!
  .
  .
  call test_bloc('X=Y**C  ')
  .
  .
!
end program test_ps


subroutine test(op)
  use decl
  use bief_def
!
  CHARACTER(LEN=8) :: OP
!
  print*
  print*,op
!
  call os('X=Y     ',xr,xs,xs,c,0,infini,zero )
!
  call os('X=Y     ',x,xs,xs,c,0,infini,zero)
!
  call os( OP , XR , Y , Z , C , iopt,infini,zero )
!
  call os_d(op,x,xd,y,yd,z,zd,c,cd,iopt,infini,zero)
!
  call os('X=Y     ',x,xs,xs,c,0,infini,zero)
!
```

```
  call os('X=Y     ',xb,xd,xd,c,0,infini,zero)
!
  call os_b(op,x,xb,y,yb,z,zb,c,cb,iopt,infini,zero)
!
  ps1 = 0 ; ps2 = 0
!
  do i=1,dim1*dim2
     ps1 = ps1 + xd%r(i) * xd%r(i)
     ps2 = ps2 + yd%r(i) * yb%r(i) + zd%r(i) * zb%r(i)
  end do
!
  ps2 = ps2 + cd * cb
!
  erreur = abs(ps1-ps2)/abs(ps1)
!
  print*,'erreur=',erreur
!
end subroutine test
```

```
****************************************
*     tests sur les vecteurs a 1 dim     *
****************************************
X=X+YZ
erreur =   4.48862236921975D-018
****************************************
*     tests sur les vecteurs a 2 dim     *
****************************************
X=CXY
erreur =   3.01678080764694D-016
****************************************
*     tests sur les blocs                *
****************************************
X=Y**C
erreur =   1.64298183786182D-017
```

DA de `vector.f`

Arborescence

VECTOR (VEC,OP,FORMUL,IELM1,XMUL,F,G,H,U,V,W,MESH,MSK,MASKEL)

```
VECTOR
|     VECTOS
|     |     VC00AA
```

```
                |     |     VC00BB
                |     |     VC01AA
                |     |     VC01BB
                |     |     VC01OO
                |     |     VC03AA
                |     |     VC03BB
                |     |     VC04AA
                |     |     VC05AA
                |     |     VC08AA
                |     |     VC08BB
                |     |     VC09AA
                |     |     VC10OO
                |     |     VC11AA2
                |     |     VC11AA
                |     |     VC11BB
                |     |     VC13AA
                |     |     VC13BB
                |     |     VC14AA
                |     |     VC15AA
                |     |     VC16AA
                |     |     VC17AA
                |     |     ASSVEC
                |     |     |     OV
                |     |     |     ASSVE1
```

Pré-traitements

En plus de quelques pré-traitements citées précédemment, il y a principalement :

* Remplacements des `T(:,i)` au début de `VECTOS` :

```
      IF(IELM1.EQ.1) THEN
            T1 = T(:,1)
            T2 = T(:,2)
      ELSEIF(IELM1.EQ.11) THEN
            T1 = T(:,1)
            T2 = T(:,2)
            T3 = T(:,3)
      ELSEIF(IELM1.EQ.12) THEN
            T1 = T(:,1)
            T2 = T(:,2)
            T3 = T(:,3)
            T4 = T(:,4)
      ENDIF
C     CALL VC00AA(XMUL,SURFAC,NELEM,NELMAX,T(:,1),T(:,2),T(:,3))
      CALL VC00AA(XMUL,SURFAC,NELEM,NELMAX,T1,T2,T3)
```

* Remplissage des `T(:,i)` à la fin de `VECTOS` :

```
      IF(IELM1.EQ.1) THEN
            T(:,1) = T1
            T(:,2) = T2
      ELSEIF(IELM1.EQ.11) THEN
            T(:,1) = T1
            T(:,2) = T2
            T(:,3) = T3
      ELSEIF(IELM1.EQ.12) THEN
            T(:,1) = T1
            T(:,2) = T2
            T(:,3) = T3
            T(:,4) = T4
      ENDIF
```

Makefile de DA

```
files   =   bief.f  bief_def.f  vector.f  vectos.f  nbpts.f  dimens.f  \
             vc00aa.f  vc00bb.f  vc01aa.f  vc01bb.f  vc01oo.f  vc03aa.f \
             vc03bb.f  vc04aa.f  vc05aa.f  vc08aa.f  vc08bb.f  vc09aa.f \
             vc10oo.f  vc11aa2.f vc11aa.f  vc11bb.f  vc13aa.f  vc13bb.f \
             vc14aa.f  vc15aa.f  vc16aa.f  vc17aa.f  assvec.f  assve1.f
b:
      tapenade -b -head vector \
      -inputfiletype fortran90 -outputlanguage fortran90 \
      -outvars "VEC" -vars "XMUL F G H U V W" \
      -O backward -html $(files)
d:
      tapenade -d -head vector \
      -inputfiletype fortran90 -outputlanguage fortran90 \
      -outvars "VEC" -vars "XMUL F G H U V W" \
      -O forward -html $(files)
```

Post-traitement et optimisation

En plus de quelques post-traitements citées précédemment, il y a principalement :

* Affectation de nbppe suivant le cas au début de `VECTOR_b` (nécessaire pour les tailles des empilements et dépilements) :

```
      if (ielm1 .eq. 1) then
           nbppe = 2
      elseif (ielm1 .eq. 11) then
           nbppe = 3
      elseif (ielm1 .eq. 12) then
           nbppe = 4
      endif
```

* Correction des tailles d'empilements dans `VECTOR_b` suivant les cas :

```
!     CALL PUSHREAL8ARRAY(mesh%w%r(1), nelmax*ISIZE1OFmesh)
!     CALL PUSHREAL8ARRAY(vec%r(1), ISIZE0OFvec)

IF (result1 .EQ. mesh%dim) THEN
      CALL PUSHREAL8ARRAY(mesh%w%r(1), mesh%nelmax*nbppe)
      CALL PUSHREAL8ARRAY(vec%r(1), npt)
ELSE
      CALL PUSHREAL8ARRAY(mesh%w%r(1), mesh%nelebx*nbppe)
      CALL PUSHREAL8ARRAY(vec%r(1), npt)
ENDIF
```

* Elimination de `vecb%r(:) = 0.D0` à la fin de `VECTOR_B` .

* Précision de certaines déclarations dans `VECOS_B` du genre :

```
!     DOUBLE PRECISION :: f(*), fb(*)
      DOUBLE PRECISION :: f(npt), fb(npt)
```

* Information de `nbppe` dans `VECOS_B` suivant les cas (nécessaire pour les tailles et quelques appels) :

```
      IF (ielm1 .EQ. 1) THEN
           nbppe = 2
           t1 = t(:, 1)
           t2 = t(:, 2)
```

```
            CALL PUSHINTEGER4(0)
      ELSE IF (ielm1 .EQ. 11) THEN
            nbppe = 3
            t1 = t(:, 1)
            t2 = t(:, 2)
            t3 = t(:, 3)
            CALL PUSHINTEGER4(1)
      ELSE IF (ielm1 .EQ. 12) THEN
            nbppe = 4
            t1 = t(:, 1)
            t2 = t(:, 2)
            t3 = t(:, 3)
            t4 = t(:, 4)
            CALL PUSHINTEGER4(3)
      ELSE
            CALL PUSHINTEGER4(2)
      END IF
```

* Tailles :

```
!     CALL PUSHREAL8ARRAY(t, nelmax*ISIZE1OFt)
!     CALL PUSHREAL8ARRAY(vec, ISIZE0OFvec)
      CALL PUSHREAL8ARRAY(t, nelmax*nbppe)
      CALL PUSHREAL8ARRAY(vec, npt)
```

* Rectifications du genre :

```
fb = 0.D0        !     fb(1:) = 0.D0
```

* Ajout des arguments `nbppe` et `npoin` nécessaires pour les tailles dans `ASSVEC_B` et `ASSVE1_B` :

```
      CALL ASSVEC_B(vec, vecb, ikle, npt, nelem, nelmax, &
     &              ielm1, t, tb, init, lv, msk, maskel, nbppe)

!     CALL POPREAL8ARRAY(x, ISIZE0OFx)
!     CALL ASSVE1_B(x, xb, ikle(1, idp), w(1, idp), &
!    &      wb(1, idp), nelem, nelmax, lv, msk, maskel)
      CALL POPREAL8ARRAY(x, npoin)
      CALL ASSVE1_B(x, xb, ikle(1, idp), w(1, idp), &
     &      wb(1, idp), nelem, nelmax, lv, msk, maskel,npoin)
```

Validation

```
program test_ps
!
  use bief_def
  use decl
  IMPLICIT NONE
!
  print*,'*******************************************************'
  print*,'*               tests sur vector                      *'
  print*,'*******************************************************'
!
  npt = 13
  dim1 = npt
  nelem = 6
```

```
  nptfr = 6
!
  call alloc
!
!  op = '='
  op = '+'
!
  formul = 'MASBAS          '
!  formul = 'MASVEC          '
!  formul = 'SUPG            '
!  formul = 'VGRADP          '
!  formul = 'VGRADP2         '
!  formul = 'FLUBOR          '
!  formul = 'FLUBOR2         '
!  formul = 'VGRADF          '
!  formul = 'QGRADF          '
!  formul = 'FLUBDF          '
!  formul = 'GGRADF         X'
!  formul = 'GGRADF         Y'
!  formul = 'GGRADF         Z'
!  formul = 'GRADF          X'
!  formul = 'GRADF          Y'
!  formul = 'GRADF          Z'
!  formul = 'PRODF           '
!  formul = 'DIVQ            '
!  formul = 'SUPGDIVU        '
!  formul = 'FLUDIF          '
!
!  ielm1 = 1
!  nbppe = 2
  ielm1 = 11
  nbppe = 3
!  ielm1 = 12
!  nbppe = 4
!
  xmul  = -9.81
  xmuld = -9.81
!  xmul  = -1.0
!  xmuld = -1.0
!
!  msk = .true.
  msk = .false.
!
  maskel%r = 1.0
!
  do i=1,npt
      f%r(i)  = 1.0
      fd%r(i) = 1.5
      g%r(i)  = 2.0
      gd%r(i) = 2.5
      h%r(i)  = 3.0
      hd%r(i) = 3.5
      u%r(i)  = 4.0
      ud%r(i) = 4.5
      v%r(i)  = 5.0
      vd%r(i) = 5.5
      w%r(i)  = 6.0
      wd%r(i) = 6.5
  enddo
!
  vec%r  = 0.0
```

```
ielm1  = 11
nbppe  = 3
op     = '+'
formul = 'MASBAS'
msk    = .false.

ps1 = 10748.290346371
ps2 = 10748.290346371

ielm1  = 12
nbppe  = 4
op     = '='
formul = 'MASBAS'
msk    = .true.

ps1 = 62.502964119970
ps2 = 62.502964119970
```

Préparation de DA de `om.f`

Arborescence

```
OM ( OP , M , N , D , C , MESH )
```

```
OM
|     CPSTMT
|     |     CPSTVC
|     OM0101
|     |     OV
|     OM1111
|     |     OV
|     OM1101
|     |     OVDB
|     OM2121
|     |     OV
|     OM1201
|     |     OVDB
|     OM5161
|     |     OVDB
|     OM5111
|     |     OV
|     OM1112
|     |     OV
|     OM1211
|     |     OV
|     OMSEG
|     |     OV
|     OMSEGBOR
|     |     OVDB
|     |     OV
```

Pré-traitements

* On écarte les renommages des ressources de modules :

```
cpre-      USE BIEF, EX_OM => OM
```

```
      USE BIEF
```

* Elimination provisoire des pointeurs. Exemple :

```
cpre- IKLE => MESH%IKLE%I
      IKLE = MESH%IKLE%I
```

* Réctification de `CHARACTER*n` par `CHARACTER(n)` . Exemple :

```
CHARACTER(1) TYPDIM , TYPEXM , TYPDIN , TYPEXN
```

* Suppression de plusieurs messages d'erreurs et instructions d'arrêt de calcul du genre :

```
      ELSE
            IF (LNG.EQ.1) WRITE(LU,100) M%NAME
            CALL PLANTE(1)
            STOP
```

* Suppression de l'optionalité de l'argument `TRANS` de la routine `CPSTMT` , et corrections :

- dans la routine :

```
            cpre- IF(PRESENT(TRANS)) THEN
                              TR = TRANS
            cpre- ELSE
            cpre-       TR = .FALSE.
            cpre- ENDIF
```

- dans les appels :

```
            cpre- CALL CPSTMT(N,M)
                  CALL CPSTMT(N,M,.FALSE.)
```

* Correction des tableaux dans les appels :
(suppression des `(1)` et remplacement des `1` par `:`) Exemples :

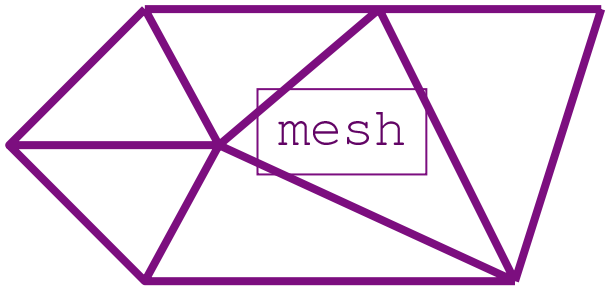
```
      CALL OM0101( OP , M%D%R(1) , TYPDIM , M%X%R(1) , TYPEXM ,
                  N%D%R(1) , TYPDIN , N%X%R(1) , TYPEXN , D%R(1) ,
                  C , IKLE , NELEM , NELMAX , NDIAGM )

      CALL OV( 'X=CY    ' , XM(:,1) , XN(:,1) , Z , C , NELEM )
```

* Ajout de `USE BIEF_DEF` dans les déclarations de toutes les routines même si c'est inclus dans le module `BIEF` .

Makefile de DA

```
files   =   cpstvc.f  bief.f   bief_def.f   nbseg.f  dimens.f  inclus.f
            ov.f  ovdb.f  cpstmt.f  om0101.f  om1111.f  om1101.f  om2121.f
            om1201.f  om5161.f  om5111.f  om1112.f  om1211.f  omseg.f
            omsegbor.f  om.f
b:
      tapenade -b -head om \
      -inputfiletype fortran90 -outputlanguage fortran90 \
      -outvars "M" -vars "N D C" \
      -O backward -html $(files)
d:
      tapenade -d -head om \
      -inputfiletype fortran90 -outputlanguage fortran90 \
      -outvars "M" -vars "N D C" \
      -O forward -html $(files)
```

```
  vecd%r = 0.0
  vecb%r = 0.0
!
!-----------------------------------------------
!                                          maillage
  mesh%w%r = 0.d0
!
  mesh%nelem  = 6
  mesh%nelmax = 6
  mesh%lv     = 1
  mesh%neleb  = 6
  mesh%nelebx = 6
      .
      .
  mesh%dim = 2
!
!-----------------------------------------------
!
  call vector_d(vec, vecd, op, formul, ielm1, xmul, xmuld, f, fd, g&
&   , gd, h, hd, u, ud, v, vd, w, wd, mesh, msk, maskel)
!
  vecb%r = vecd%r
!
  call vector_b(vec, vecb, op, formul, ielm1, xmul, xmulb, f, fb, g&
&   , gb, h, hb, u, ub, v, vb, w, wb, mesh, msk, maskel)
!
  ps1 = 0.d0 ; ps2 = 0.d0
!
  do i=1,dim1
     ps1 = ps1 + vecd%r(i) * vecd%r(i)
     ps2 = ps2 + fd%r(i) * fb%r(i)
     ps2 = ps2 + gd%r(i) * gb%r(i)
     ps2 = ps2 + hd%r(i) * hb%r(i)
     ps2 = ps2 + ud%r(i) * ub%r(i)
     ps2 = ps2 + vd%r(i) * vb%r(i)
     ps2 = ps2 + wd%r(i) * wb%r(i)
  end do
!
  ps2 = ps2 + xmuld * xmulb
!
  erreur = abs(ps1-ps2) / abs(ps1)
!
  print*,'ps1=',ps1
  print*,'ps2=',ps2
  print*,'erreur=',erreur
!
end program test_ps
```




```
*************************************************
*                tests sur vector               *
*************************************************
ielm1  = 1
nbppe  = 2
op     = '='
formul = 'MASVEC'
msk    = .true.

ps1 = 432.31249849498
ps2 = 432.31249849498
```

Post-traitement et optimisation

* Remplacement de `bief_b` et `bief_def_b` :

```
      USE bief
      USE bief_def
```

* Rectification :

```
!     CHARACTER :: typdim*1, typdin*1, typexm*1, typexn*1
      CHARACTER(1) :: typdim, typdin, typexm, typexn
```

* Ré-intégration des pointeurs :

```
!     INTEGER :: IKLE(*)
      INTEGER, DIMENSION(:), POINTER :: IKLE
      IKLE => MESH%IKLE%I
```

* Taille des empilements :

```
!     CALL PUSHINTEGER4ARRAY(ikle, ISIZE0OFikle)
      CALL PUSHINTEGER4ARRAY(ikle, size(ikle) )

!     CALL PUSHREAL8ARRAY(m%d%r, ISIZE0OFm)
      CALL PUSHREAL8ARRAY(m%d%r, m%d%dim1*m%d%dim2)
```

* Empilement général mis hors une boucle `IF` :

```
      CALL PUSHINTEGER4(nseg1)
      CALL PUSHINTEGER4(nseg2)
      CALL PUSHINTEGER4(nelem)
      CALL PUSHINTEGER4(nptfr)
      CALL PUSHINTEGER4(netage)
      CALL PUSHINTEGER4(sizxn)
      CALL PUSHINTEGER4(ndiagm)
      CALL PUSHINTEGER4(ndiagn)
      CALL PUSHINTEGER4(ndiagx)
      CALL PUSHINTEGER4ARRAY(ikle, size(ikle) )
      CALL PUSHCHARACTERARRAY(typexm, 1)
      CALL PUSHCHARACTERARRAY(typexn, 1)
      CALL PUSHCHARACTERARRAY(typdim, 1)
      CALL PUSHCHARACTERARRAY(typdin, 1)
      CALL PUSHREAL8ARRAY(m%x%r, m%x%dim1*m%x%dim2)
      CALL PUSHREAL8ARRAY(m%d%r, m%d%dim1*m%d%dim2)
```

* Dépilement général mis hors une boucle `IF` :

```
      CALL POPREAL8ARRAY(m%d%r, m%d%dim1*m%d%dim2)
      CALL POPREAL8ARRAY(m%x%r, m%x%dim1*m%x%dim2)
      CALL POPCHARACTERARRAY(typdin, 1)
      CALL POPCHARACTERARRAY(typdim, 1)
      CALL POPCHARACTERARRAY(typexn, 1)
      CALL POPCHARACTERARRAY(typexm, 1)
      CALL POPINTEGER4ARRAY(ikle, size(ikle) )
      CALL POPINTEGER4(ndiagx)
      CALL POPINTEGER4(ndiagn)
      CALL POPINTEGER4(ndiagm)
      CALL POPINTEGER4(sizxn)
      CALL POPINTEGER4(netage)
```

```
      CALL POPINTEGER4(nptfr)
      CALL POPINTEGER4(nelem)
      CALL POPINTEGER4(nseg2)
      CALL POPINTEGER4(nseg1)
```

* Suppression de certains empilements et dépilements inutiles :

```
!     CALL PUSHINTEGER4(:)
!     CALL POPINTEGER4(:)
!     CALL POPINTEGER4(index1)
```

* Taille :

```
!     CALL PUSHREAL8ARRAY(xm, nelmax*ISIZE1OFxm)
      CALL PUSHREAL8ARRAY(xm(:, 1), nelem)
      CALL OV('X=Y     ', xm(:, 1), xn(:, 1), z, c, nelem)

      CALL POPINTEGER4(index1)
!     CALL POPINTEGER4(index2)
!     CALL POPREAL8ARRAY(dm(index2, index1+1), szmdn)
      CALL POPREAL8ARRAY(dm(:, index1+1), sizdn)

!     CALL OV_B('X=X+Y   ', dm(index2, index1+1), &
!    &dmb(index2, index1+1), dn, dnb, z, zb, c, cb, sizdn)
      CALL OV_B('X=X+Y   ', dm(:, index1+1), &
     &dmb(:, index1+1), dn, dnb, z, zb, c, cb, sizdn)
```

* Rectification :

```
!     xnb(1:nelmax, 1:) = 0.D0
      xnb(1:nelmax,  :) = 0.D0
```

* Taille :

```
!     CALL PUSHREAL8ARRAY(xm, MIN(nseg1, nseg2)*ISIZE1OFxm)
      CALL PUSHREAL8ARRAY(xm, nseg1)
      CALL OV('X=Y     ', xm, xn, z, c, nseg1)
```

* Faux empilement :

```
      arg1 = nseg1 + nseg2
!     CALL PUSHINTEGER4(arg1)
```

* Déplacement d'initialisations répétées hors boucle `IF` :

```
!     DO ii1=1,SIZE1OFx
!       xd(ii1) = 0.D0
!     END DO
!
       xd = 0.D0
```

Validation

Le répertoire de validation contient déjà le makefile, le programme principal du test du produit scalaire, et tous les fichiers nécessaires (arborescence de `om` et routines différentiées). On a commencé quelques essais de validation qui n'ont pas encore abouti à la réussite du test.

6.4 - Travail à compléter

Nous recommandons de suivre le plan de travail déjà expérimenté et réussi pour la DA de `os` et `vector` par exemple. Outre la compréhension de la démarche générale expliquée à travers ce document, une maîtrise de l'arborescence et de la gestion des fichiers dans le cadre prescrit est aussi primordiale pour la réussite de l'opération de DA de `run_imposed`.

DA de `om.f`

Il reste souhaitable de commencer la complétion du travail par la DA de cette routine et arborescence. Quoique le répertoire de validation et le test sont prêts, nous préférons une révision des pré-traitements. Essentiellement, suite à notre expérience avec `vector`, nous pensons que le pré-traitement des tableaux passés en paramètres du type ci-dessous aide beaucoup et évite des post-traitements difficiles (répétitifs et manque des tailles des tableaux, vu aussi la grande taille du code `om`). Cette note concerne au moins la version 2.0.11 de Tapenade qu'on utilise.

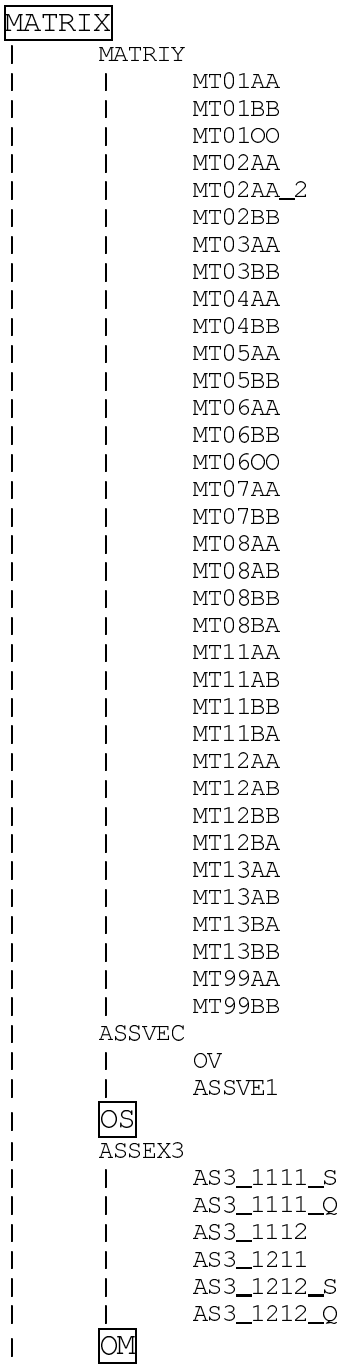
```
!     CALL OV( 'X=CY    ' , XM(:,1) , XN(:,1) , Z , C , NELEM )

      XM1 = XM(:,1)
      XN1 = XN(:,1)
```

et suivre leurs tailles dans les arguments des routines qui les utilisent :

```
      CALL OV( 'X=CY    ' , XM1 , XN1 , Z , C , NELEM,  &
     &         tailleXM1, tailleXN1 )
```

DA de `matrix.f`

Arborescence



Pré-traitements

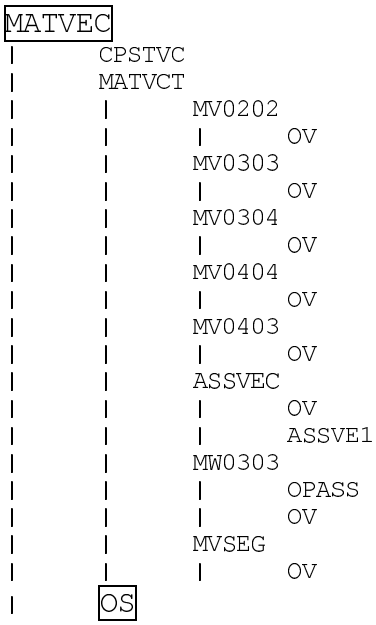
Makefile de DA

Post-traitement et optimisation

Validation

DA de `matvec.f`

Arborescence



Pré-traitements

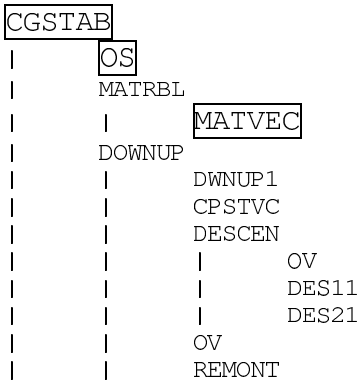
Makefile de DA

Post-traitement et optimisation

Validation

DA de `cgstab.f`

Arborescence



```
|       |       |       OV
|       |       |       REM11
|       |       |       REM21
```

Pré-traitements

Makefile de DA

Post-traitement et optimisation

Validation

6.5 DA de `gmres.f`

Arborescence

```
GMRES
|       GODOWN
|       |       GODWN1
|       |       |       CPSTVC
|       |       |       DESCEN
|       |       |       |       OV
|       |       |       |       DES11
|       |       |       |       DES21
|       MATRBL
|       |       MATVEC
|       PUOG
|       |       PUOG1
|       |       |       CPSTVC
|       |       |       TNOMER
|       |       |       |       OV
|       |       |       |       MER11
|       |       |       |       MER21
|       OS
|       GOUP
|       |       GOUP1
|       |       |       CPSTVC
|       |       |       REMONT
|       |       |       |       OV
|       |       |       |       REM11
|       |       |       |       REM21
```

Pré-traitements

Makefile de DA

Post-traitement et optimisation

Validation

7 Chaîne d'optimisation

Cette section est consacrée à l'intégration de la chaîne d'optimisation pour Telemac2D-v5.4-simple nécéssaire pour effectuer de l'assimilation de données. Cette chaîne couple le code direct, le code adjoint et l'algorithme de minimisation. Nous utilisons comme programme de minimisation N1QN3 du module d'optimisation MODULOPT, développé par J.-C. Gilbert et C. Lemaréchal à l'INRIA, et basée sur l'approche BFGS quasi-Newton à mémoire limitée.

Méthode du quasi-Newton

On cherche à résoudre le problème $\nabla_u J(u) = 0$, $u \in \mathrm{R}^n$, où $\nabla_u J$ est supposé assez régulière et de Jacobien inversible. L'algorithme du quasi-Newton s'écrit

$$u^{k+1} = u^k - \rho^k \; G_k \; \nabla_u J(u^k)$$

avec G_k une approximation de l'inverse du Hessien $H_u^{-1}(u^k)$ en u^k .
La formule BFGS de mise à jour de G_k se rapporte à une classe de méthodes quasi-Newton, dans laquelle on utilise l'information sur les gradients précédents pour composer G_k.

Algorithme N1QN3 de l'INRIA

L'approximation du Hessien est basée sur une formule BFGS par Nocedal, avec une option de préconditionnement due à Oren et Spedicato

$$G_{k+1} = (I - \frac{s^k (y^k)^\top}{(y^k)^\top s^k}) \; G_k \; (I - \frac{y^k (s^k)^\top}{(y^k)^\top s^k} + \frac{s^k (s^k)^\top}{(y^k)^\top s^k})$$

où $s^k = u^{k+1} - u^k$, $y^k = \nabla_u J(u^{k+1}) - \nabla_u J(u^k)$
La matrice G_k n'est pas stockée en mémoire, mais peut être représentée par $(2m+1)$ vecteurs, où m est un entier fourni par l'utilisateur. Ainsi, le produit $G_k \; \nabla_u J(u^k)$ est calculé directement à partir d'une matrice diagonale D_k et m paires de vecteurs $\{(y_j, s_j) \; , \; k-m \leq j \leq k-1\}$ si $k \geq m+1$, ou juste k paires sinon. Sans préconditionnement, $D_k = \delta_{k-1} I$, avec δ_k la fonction d'Oren-Spedicato $\delta_k = (y^k)^\top s^k \; / \; \|y^k\|^2$.
Le pas de déscente ρ_k est choisi de façon à satisfaire les conditions de Wolfe

$$\begin{aligned} & J(u^{k+1}) \; \leq \; J(u^k) + \alpha_1 \; \rho^k \; < \nabla_u J(u^k), G_k \; \nabla_u J(u^k) > \\ & < \nabla_u J(u^{k+1}), G_k \; \nabla_u J(u^k) > \geq \; \alpha_2 \; < \nabla_u J(u^k), G_k \; \nabla_u J(u^k) > \end{aligned}$$

où α_1 et α_2 sont des constantes satisfaisant $0 < \alpha_1 < 0.5$ et $\alpha_1 < \alpha_2 < 1$.
Dans l'algorithme N1QN3, on a fixé $\alpha_1 = 10^{-4}$ et $\alpha_2 = 0.9$.

Implémentation pour Telemac2D

En plus des modes libre et imposé codés dans Telemac2D-v5.4-simple, nous ajoutons un mode d'exécution appelé `run_twin` pour les expériences jumelles qui présume l'adjoint

entier réalisé, et qui est doté d'une chaîne d'optimisation complète. Nous baptisons cette première version destinée à l'AD Telemac2D-v5.4-opt-v1.0 .

Algorithme de `run_twin` :

1 : initialisation des variables de contrôle : $S_{\#}$, $z_{f\#}$, $CL_{N\#}$, $CL_{D\#}$, $h0_{\#}$, $u0_{\#}$, $v0_{\#}$.

2 : information du nombre total des variables de contrôle discrètes :
n = 5 * nombre des points
+ 2 * nombre des itérations en temps * nombre des frontières

3 : allocations : x(n) et gradf(n) .

4 : remplissage de x par les variables de contrôle discètes .

5 : initialisation à 0 de izs , rzs et dzs

6 : appel SIMUL(indic,n,x,f,gradf,izs,rzs,dzs) , avec indic = 4 : remplit
* la fonction coût f,
* la fonction coût totale dzs(4),
* et le gradient du coût gradf par les variables de contrôle adjointes,
obtenues par DA de la routine du mode imposé.

7 : initialisations :
dzs(1) = fonction coût initiale
dzs(2) = norme du gradient gradf initiale
dzs(3) = fonction coût totale initiale
dzs(4) = fonction coût totale

8 : préparation des affichages :
itération f $f/dzs(1)$ $||gradf||$ $||gradf||/dzs(2)$ $dzs(4)$ $dzs(4)/dzs(3)$

9 : autres données de minimisation :

dxmin = 1d-7	mode = 0
df1 = f	niter = 1000
epsg = 1d-5	nsim = 1500
impres = -1	m = 10
io = 6	ndz = 4*n+m*(2*n+1)

10: allocation : dz(ndz)

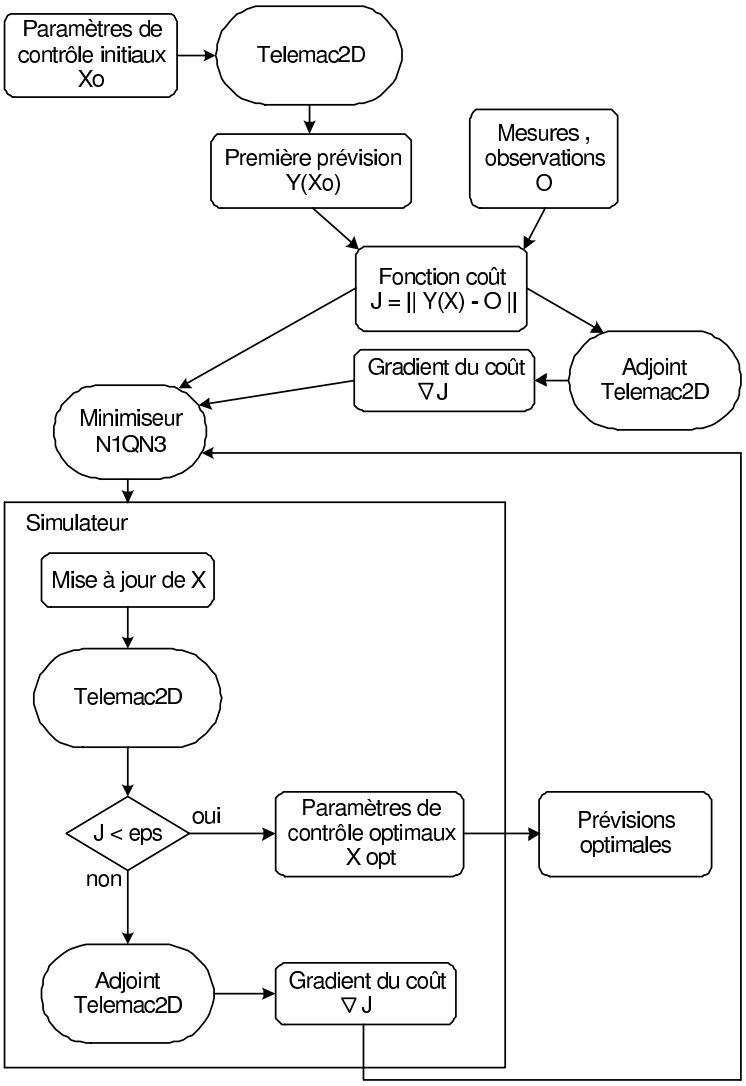
11: affichage des variables de contrôle initiales .

12: appel N1QN3(simul,duclid,dtonbe,dtcabe,n,x,f,gradf,dxmin,df1,
epsg,impres,io,mode,niter,nsim,iz,dz,ndz,izs,rzs,dzs) : minimisation

13: affichage des variables de contrôle optimales .

14: appel SIMUL(indic,n,x,f,gradf,izs,rzs,dzs) , avec indic = 2 :
simulation de l'écoulement avec les variables de contrôle optimales .

Organigramme d'optimisation


Paramètres de contrôle initiaux Xo
Telemac2D
Première prévision Y(Xo)
Mesures , observations O
Fonction coût J = || Y(X) - O ||
Gradient du coût ∇J
Adjoint Telemac2D
Minimiseur N1QN3
Simulateur
Mise à jour de X
Telemac2D
J < eps
oui
non
Paramètres de contrôle optimaux X opt
Prévisions optimales
Adjoint Telemac2D
Gradient du coût ∇J

8 Conclusion

Dans ce travail, nous avons établi des fondements de base numériques et informatiques de l'AD variationnelle dans le logiciel de modélisation hydrodynamique Telemac2D. Principalement :

- une version réduite du logiciel pour l'écoulement fluvial et adaptée pour la DA par Tapenade a été réalisée et validée: Telemac-v5.4-simple-v5.0 avec un guide de programmation et un manuel d'utilisation ;

- une stratégie de DA par morceaux du code a été proposée avec des étapes de prétraitement, différentiation, post-traitement et validation, expliquées dans un guide de DA ;

- et une chaîne d'optimisation par N1QN3 a été implémentée dans la version Telemac-v5.4-opt-v1.0 .

Une fois la DA terminée et validée, les processus d'assimilation de données, d'analyse de sensibilité, d'identification et de calibrage de modèle pourront être effectués avec ce module T2DInverse; ce qui permettrait d'améliorer les prédictions des modèles réels d'écoulements fluviaux, et d'assister de manière plus efficace les plans de prévention des inondations.

Remerciements

Ce travail a été effectué dans le cadre du séjour Post-Doctoral de Y. Loukili, financé par la région Rhône Alpes.
Les auteurs voudrait remercier L. Hascoet et V. Pascual du projet TROPICS de l'INRIA Sophia-Antipolis pour leur aide indispensable et leur expertise; et aussi les thésards du projet IDOPT pour la diffusion de leurs connaissances en différenciation automatique.

References

[1] Mazauric C. *Etude de l'assimilation de données variationnelle pour les modèles d'hydraulique fluviale.* PhD thesis, Université de Grenoble 1, 2003.

[2] Barros E. *Identification de paramètres dans les équations de Saint-Venant.* PhD thesis, Universite Paris VI, 1996. Publ. EDF-DER HE-43/96/034/A-1996.

[3] Hervouet J.-M. *Hydrodynamique des écoulements à surface libre, modélisation numérique avec la méthode des éléments finis.* Habilitation à diriger des recherches -hdr-, Université de Caen / Basse-Normandie, 2001.

[4] Inria, projet Tropics. *The* Tapenade *tutorial.* `<http://www-sop.inria.fr/tropics/>`.

[5] P. Vidard. *Vers une prise en compte de l'erreur modèle en assimilation de données 4D-variationnelle.* PhD thesis, Université Joseph Fourier, Décembre 2001.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399