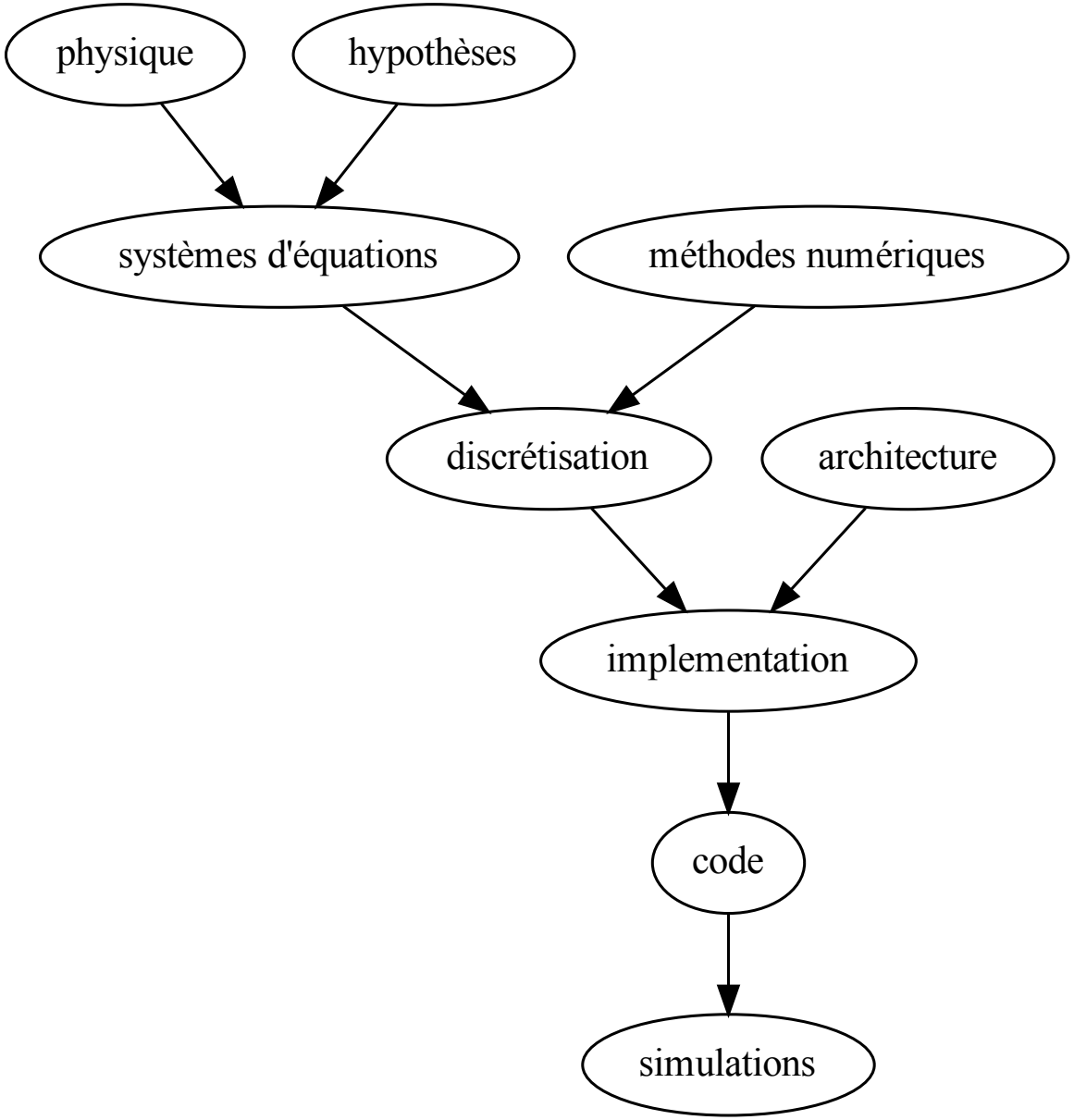
Des équations aux codes

G. Roullet
LOPS, Univ. Brest

24-28 novembre 2025


physique
hypothèses
systèmes d'équations
méthodes numériques
discrétisation
architecture
implementation
code
simulations

Introduction

Objectifs de ce cours

- équations et variables
- méthodes numériques
- architecture
- implémentation

Où comment passer des maths à un code.

- Ce cours sera illustré avec le code `Fluids2d`, 100% Python, $\sim 1\,500$ lignes de code
- ce code ne traite que des cas 2D, (x, y) ou (x, z).
- les TP utiliseront ce code.

Equations

Diversité des systèmes d'équations

- équations d'Euler incompressibles,
- équations Boussinesq,
- équations primitives (PE),
- équations couche mince (rotating shallow water) (RSW),
- équations quasi-géostrophiques (QG),
- ...

Hypothèses

Diversité des hypothèses

- incompressible,
- Boussinesq,
- hydrostatique,
- Coriolis: hypothèse traditionnelle, plan beta,
- géoïde sphérique,
- etc.

Variables

Diversité des variables

A chaque jeu d'équations correspond un jeu de variables

modèle		prognostiques	diagnostiques
Euler		$\mathbf{u}$	p
PE	Primitive Equations	$(\mathbf{u}_h, T, S)$	$(\rho, p, w, K, \boldsymbol{\omega})$
RSW	Rotating Shallow Water	$(\mathbf{u}_h, h)$	(p, K, ω)
QG	Quasi-geostrophic	q	$(\mathbf{u}_h, p, h)$

Deux groupes de variables

prognostiques : variables qui possèdent une loi d'évolution temporelle explicite

diagnostiques : variables qui se déduisent des variables prognostiques

Diagnostique vs. prognostique

Exemples

- $\mathbf{u}_h$, **quantité de mouvement horizontale** $\rightarrow$ prognostique dans presque tous les modèles, sauf pour QG $\rightarrow$ diagnostique.
- w, **vitesse verticale** $\rightarrow$ prognostique en non-hydrostatique, diagnostique dans les modèles PE, RSW etc
- K, **énergie cinétique** $\rightarrow$ diagnostique
- p, **pression** $\rightarrow$ diagnostique
- T, **température** $\rightarrow$ prognostique ou diagnostique
- ρ, **masse volumique** $\rightarrow$ presque toujours diagnostique, sauf dans les modèles simples, à une composante, viz. $\rho(T)$.

Structure des équations

Systèmes d'EDP

3 grandes **classes de problèmes** (imbriqués)

- transport
- ondes
- diffusion

et souvent, de manière plus ou moins cachée: des **problèmes elliptiques**

Liens avec le numérique

- Les méthodes numériques diffèrent largement selon la classe du problème.
- La difficulté numérique est bien plus grande sur le transport que sur la diffusion.
- Les problèmes elliptiques font peur à la communauté OA.

Equations d'Euler incompressibles

Conservation quantité de mouvement + incompressibilité

$$\partial_t \mathbf{u} = -\omega \times \mathbf{U} - \boldsymbol{\nabla}(p + K) \tag{1}$$

$$\boldsymbol{\nabla} \cdot \mathbf{U} = 0 \tag{2}$$

Quantités diagnostiques

$$\omega = \boldsymbol{\nabla} \times \mathbf{u}$$

$$K = \frac{1}{2}\mathbf{u} \cdot \mathbf{U}$$

$$\mathbf{U} = g^{-1}\mathbf{u}$$

$\mathbf{u}$: vitesse **transportée**
$\mathbf{U}$: vitesse **transportante**
g : tenseur de métrique

$$g = \begin{pmatrix} dx^2 & 0 \\ 0 & dy^2 \end{pmatrix}$$

À noter: sous cette forme, les équations sont **invariantes par changement de coordonnées**.

Calcul de la pression (équations d'Euler)

Equation de Poisson pour la pression (cas $g = I$)

$$\boldsymbol{\nabla} \cdot \mathbf{U} = 0 \quad \rightarrow \quad \boldsymbol{\nabla}^2 p = -\boldsymbol{\nabla} \cdot (\omega \times \mathbf{U}) - \boldsymbol{\nabla}^2 K$$

En pratique on utilise plutôt une **méthode de projection**

$$\begin{aligned} \mathbf{u}^* &= \mathbf{u}^n - \Delta t \, (\omega \times \mathbf{U} + \boldsymbol{\nabla} K) \\ \mathbf{u}^{n+1} &= \mathbf{u}^* - \boldsymbol{\nabla} \tilde{p} \end{aligned}$$

avec $\tilde{p} = p\Delta t$ et l'**équation de Poisson**

$$\boldsymbol{\nabla}^2 \tilde{p} = \boldsymbol{\nabla} \cdot \mathbf{u}^* \, . \tag{3}$$

Cette technique permet de garantir $\boldsymbol{\nabla} \cdot \mathbf{u}^n \sim 0$ pour tout n.
Résolution de (3) avec un algo itératif → solution approchée pour p.

Inteprétation de la pression

La pression est une grandeur clef en OA, et dans la vraie vie: 1028 mb, il fait beau ...

3 interprétations

- p est le **multiplicateur de Lagrange** qui s'ajuste pour que la force $-\boldsymbol{\nabla} p$ maintienne l'écoulement incompressible
- p est le **poids de la colonne** (hydrostatique)
- p est une **grandeur thermodynamique**, $p = -(\partial U/\partial V)_{(S,\mu)}$

Laquelle est la bonne ?

Réponse: toutes ! ça dépend du modèle = du jeu d'équations.

Autre technique

Pour éviter d'affronter la résolution de l'équation de Poisson: **relaxation de la compressibilité** $\rightarrow$ présence d'ondes "sonores"

$$\partial_t \mathbf{u} = -\omega \times \mathbf{U} - \boldsymbol{\nabla}(p + K)$$
$$\partial_t s = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, s)$$
$$p = c^2\, s$$

avec s un traceur virtuel et c un paramètre contrôlant la vitesse des ondes.

- Avec $c \rightarrow \infty$ on rend le système "raide" et la solution tend vers l'incompressibilité.
- On remplace un pb elliptique par une intégration temporelle de ces ondes
- cf. CROCO NBQ pour gérer le non-hydrostatique.

Equations de Boussinesq

On ajoute le traceur actif b, **flottabilité** (buoyancy)

$$\partial_t \mathbf{u} = -\omega \times \mathbf{U} - \boldsymbol{\nabla}(p + K) + b\boldsymbol{\nabla}\phi$$
$$\partial_t b = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, b)$$
$$\boldsymbol{\nabla} \cdot \mathbf{U} = 0$$

avec $\phi = g\, z$, le géopotentiel, g l'accélération de la gravité.

buoyancy

$$b = -g \frac{\rho_0 - \rho}{\rho_0}$$

La flottabilité

- est une accélération
- varie comme la température (b grand=léger)

Quizz

1. Comment est calculée la pression dans ce modèle ?

$$\partial_t \mathbf{u} = -\omega \times \mathbf{U} - \boldsymbol{\nabla}(p + K) + b\boldsymbol{\nabla}\phi$$
$$\partial_t b = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, b)$$
$$\boldsymbol{\nabla} \cdot \mathbf{U} = 0$$

2. Comment interpréter la pression ?

Modèle Boussinesq hydrostatique

Dans le modèle précédent, la pression n'est pas calculée comme étant le poids de la colonne, viz. elle n'est pas hydrostatique.

Version hydrostatique (2D vertical)

$$
\begin{aligned}
\partial_t u &= \omega\, W - \partial_x (p + K) \\
\partial_z p &= b \\
\partial_t b &= -\boldsymbol{\nabla} \cdot (\mathbf{U}\, b) \\
\boldsymbol{\nabla} \cdot \mathbf{U} &= 0
\end{aligned}
$$

avec $K = \frac{1}{2} u U$ et $\omega = -\partial_z u$

Distorsions hydrostatiques

- pas d'énergie cinétique verticale
- une vorticité (horizontale) sans la composante w
- $w = 0$ (**transportée**) mais $W \neq 0$ (**transportante**)

Calcul de la pression et de W

La pression est calculée par intégration verticale

$$p = -\int_z^0 b\, dz \tag{4}$$

ainsi que la vitesse verticale W (en utilisant $\boldsymbol{\nabla} \cdot \mathbf{U} = 0$)

$$W = -\int_{-H}^z \partial_x U\, dz \tag{5}$$

Attention $W|_{z=0} \neq 0$: la dynamique intérieure crée des vitesses verticales en surface $\rightarrow$ flux à travers la surface ?

Mode barotrope

Deux approches

1) Toit rigide / projection de pression

On introduit une pression de surface p^s telle que $W|_{z=0} = 0$. Comme la projection précédente mais uniquement en surface.

$$u^* = u^n + \Delta t\, R(u, U, W)$$
$$u^{n+1} = u^n - \partial_x p^s$$

avec

$$(\partial_x H \partial_x) p^s = \partial_x \left(\sum_{-H}^{0} u^*\, dz \right)$$

Ca ressemble au modèle d'Euler mais $p^s(x)$ là où $p(x, z)$.

Mode barotrope

2) Surface libre

On introduit η la hauteur de surface libre et on pose

$$\partial_t \eta = W|_{z=0}$$

- Equation prognostique supplémentaire $\rightarrow$ ondes de gravité barotropes
- Ondes rapides $c = \sqrt{gH} \rightarrow$ oblige un petit Δt
- Difficulté numérique, e.g. technique du time-splitting

Modèle RSW

Le modèle couche-mince tournant (1 couche)

$$\partial_t \mathbf{u} = -(f + \omega) \times \mathbf{U} - \boldsymbol{\nabla}(p + K)$$
$$\partial_t h = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, h)$$
$$p = g\, h$$

avec h hauteur d'eau, H épaisseur moyenne de h, g l'accélération de la gravité et f le paramètre de Coriolis

modèle jouet

- ondes de gravito-inertie $\omega^2 = f^2 + gHk^2$
- équilibre géostrophique
- dynamique de la **vorticité potentielle** $q = (f + \omega)/h$ (variété lente)
- **rayon de déformation de Rossby** $R = \sqrt{gH}/f$.

Modèle quasi-géostrophique

Obtenu à partir de RSW en **projetant la dynamique** $\partial_t s = R(s)$ **sur le mode géostrophique**

$$\partial_t s = P \circ R(s)$$

avec $s = (\mathbf{u}, h)$ et l'**opérateur de projection** [Thiry et al, 2024]

$$P = G \circ (Q \circ G)^{-1} \circ Q$$
$$Q(\mathbf{u}, h) = \boldsymbol{\nabla} \times \mathbf{u} - fh/H$$
$$G(p) = (\boldsymbol{\nabla}^{\perp} p/f, p/g)$$

Le modèle QG a filtré les ondes de gravito-inertie. On peut utiliser un pas de temps plus grand qu'avec RSW.

Le modèle du vent thermique

On reprend Boussinesq et on ajoute un jet $V(x,z)$ perpendiculaire au plan (x,z)

$$\partial_t \mathbf{u} = -\omega \times \mathbf{U} - \boldsymbol{\nabla}(p+K) + b\boldsymbol{\nabla}\phi + fV\boldsymbol{\nabla}x$$
$$\partial_t b = -\boldsymbol{\nabla}\cdot(\mathbf{U}\, b)$$
$$\partial_t V = -\boldsymbol{\nabla}\cdot(\mathbf{U}\, V) - fU$$
$$\boldsymbol{\nabla}\cdot\mathbf{U} = 0$$

avec f le paramètre de Coriolis.

- Equilibre du vent thermique ($\mathbf{U}=0$): $-\boldsymbol{\nabla}p + b\boldsymbol{\nabla}z + fV\boldsymbol{\nabla}x = 0$
- Dans ce modèle $\mathbf{U}$ est la vitesse secondaire ($\perp$ jet).
- Attention comme tout modèle 2D, sa zone de validité est étroite.

Equations de transport

transport de traceur

forme flux

$$\partial_t q = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, q) \tag{6}$$

forme advective

$$\partial_t q = -\mathbf{U} \cdot \boldsymbol{\nabla} q \tag{7}$$

En pratique, on préfère (6) car elle garantit la **conservation du traceur**.

Equations de transport

transport de traceur

forme flux

$$\partial_t q = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, q) \tag{6}$$

forme advective

$$\partial_t q = -\mathbf{U} \cdot \boldsymbol{\nabla} q \tag{7}$$

En pratique, on préfère (6) car elle garantit la **conservation du traceur**.

Voyez-vous que (6) est la **version volumes finis (VF)** et (7) la **version différences finies (DF)** ?

Equations de transport

transport de traceur

forme flux

$$\partial_t q = -\boldsymbol{\nabla} \cdot (\mathbf{U}\, q) \tag{6}$$

forme advective

$$\partial_t q = -\mathbf{U} \cdot \boldsymbol{\nabla} q \tag{7}$$

En pratique, on préfère (6) car elle garantit la **conservation du traceur**.

Voyez-vous que (6) est la **version volumes finis (VF)** et (7) la **version différences finies (DF)** ?

transport de vecteur

$$\partial_t \mathbf{v} = -(\boldsymbol{\nabla} \times \mathbf{v}) \times \mathbf{U} - \boldsymbol{\nabla}(\mathbf{U} \cdot \mathbf{v})$$

à noter: pas de coefficient $1/2$ dans $\mathbf{U} \cdot \mathbf{v}$.

Synthèse sur les équations

- Grande diversité des systèmes d'équations
- Des ingrédients génériques: transport, projection, etc.,
- qui s'assemblent à la manière d'un jeu de LEGO.
- Une famille de processus essentiels impactant la physique et le choix des méthodes numériques: les **ondes**.
  - ondes sonores
  - ondes de gravité: de surface et interne
  - ondes de vorticité (Rossby)

Discrétisations

Discrétisations en temps

- Leap-Frog + filtre d'Asselin (LFRA)
- méthodes de Runge-Kutta, eg. RK3, RK4
- méthodes Adams Bashforth
- méthodes semi-implicites
- etc.

Discrétisations en espace

- grilles logiquement rectangulaires (i, j, k)
- grilles destructurées (éléments finis)
- grilles hexagonales/icosahédriques
- grilles spectrales (FFT ou harmoniques sphériques)
- méthodes de Galerkin discontinues

Discrétisation en espace

Choix des coordonnées

- coordonnées horizontales: (x, y), (λ, ϕ), généralisées
- coordonnées verticales: z, ρ, p, σ (terrain-following)

Grilles décalées

Les grilles sont presque toujours **décalées** (staggered)

- les variables sont discrétisées à différents endroits en fonction de leur nature
- éléments géométriques: centers, edges, faces, volumes
- grilles A, B, C, etc. (nomenclature de Arakawa)
- **la grille C identifie le placement géométrique avec la nature de la variable** (scalaire, vecteur, pseudo-vecteur etc).

Grille C

Grille primale

points

segments flux $\mathbf{U}\,q$

surfaces traceurs b, q

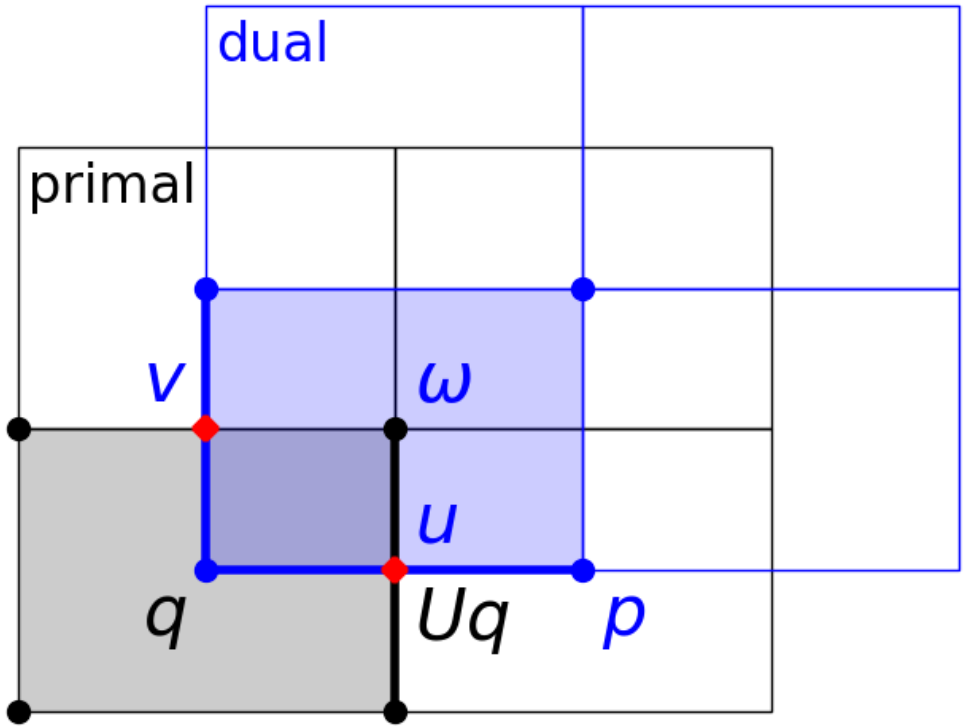
Grille **duale**

surfaces vorticité ω

segments vitesse $\mathbf{u}$

points pression p

en 3D, on ajoute les volumes & les traceurs deviennent des volumes



L'opérateur $\boldsymbol{\nabla}$ fait passer de
points $\rightarrow$ segments $\rightarrow$ surfaces (2D)
points $\rightarrow$ segments $\rightarrow$ surfaces $\rightarrow$ volumes (3D)
[Desbrun et al 2006]

Notion d'ordre

Ordre d'une discrétisation

- L'**ordre** n d'une discrétisation est l'**exposant** de son erreur de troncature $\sim \mathcal{O}(h^n)$, avec h le pas de discrétisation (temps ou espace).
- L'ordre détermine la vitesse de convergence lorsqu'on raffine le pas h
- Plus l'ordre est grand, plus la discrétisation est précise
- Pour monter l'ordre, il faut utiliser plus de points (stencils plus étendus) et faire plus d'opérations.

Exemple

$\partial_x \phi = (\phi_i - \phi_{i-1})/h$ est d'ordre 1 si le résultat est évalué en i ou en $i-1$ mais d'ordre 2 si il est évalué en $i - 1/2$.

Ordre 2 et au delà

- Il est facile d'avoir des discrétisations d'ordre 2. Ce fut longtemps le défaut dans les codes.
- Certains opérateurs peuvent être montés à des ordres supérieurs: 3, 4 ou 5 pour le transport par exemple, 3 pour le temps (RK3).
- La montée en ordre permet d'avoir des solutions plus précises, pour une résolution donnée.
- Attention cependant, monter en ordre le transport des traceurs doit aller de pair avec un champ de vitesse d'ordre supérieur aussi (cf. TP) + question du transport de vecteur.

Subtilités liées à la montée en ordre

- la variable discrétisée est-elle en **différences finies** ou en **volume finis** ?
- **DF**: ϕ_i est la valeur au point x_i
- **VF**: ϕ_i est la valeur moyenne sur la maille i
- discrétisation sur volumes: VF, sur vertex: DF
- A l'ordre 2 $\rightarrow$ pas de différence, aux ordres supérieurs, c'est différent!
- Exemple: interpolation 4e ordre au point $x_{i+1/2}$

$$\text{DF} \quad \phi^*_{i+1/2} = (-\phi_{i-1} + 9\phi_i + 9\phi_{i+1} - \phi_{i+2})/16$$
$$\text{VF} \quad \phi^*_{i+1/2} = (-\phi_{i-1} + 7\phi_i + 7\phi_{i+1} - \phi_{i+2})/12$$

Discrétisation et interpolation

Comment (re-)trouver les poids d'une discrétisation ?

Soit ϕ_i un discrétisation FD de ϕ, S un stencil = un ensemble d'indices, e.g. $(i-1, i, i+1)$. Une **reconstruction** $\phi(x)$ sur l'intervale $[x_{i-1}, x_{i+1}]$ est

$$\phi(x) = \sum_{j \in S} P_j(x)\, \phi_j$$

où $P_j(x)$ est le **polynôme de Lagrange**

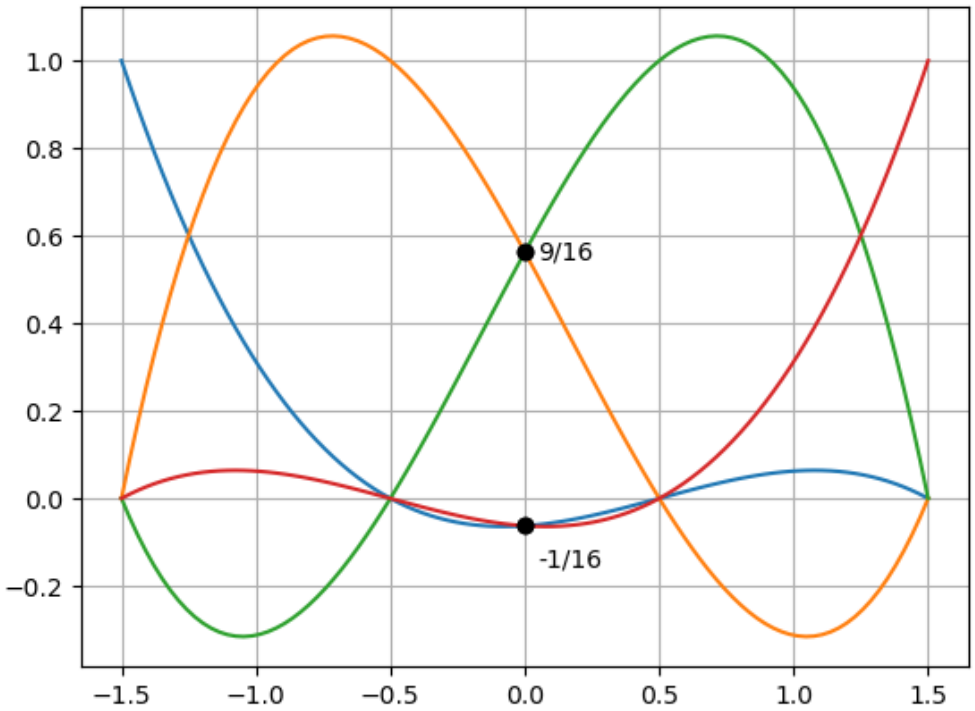
$$P_j(x) = \prod_{k \in S,\ k \neq j} \frac{x - x_j}{x_k - x_j}$$

On peut alors trouver les poids de n'importe quelle dérivée en n'importe quel point. Par exemple, la discrétisation de la dérivée première au point $i + 1/2$ est

$$\phi'_{i+1/2} = \sum_{j \in S} w_j\, \phi_j \quad \text{avec} \quad w_j = P'_j(x_{i+1/2})$$

Polynômes de Lagrange

Les 4 polynômes de Lagrange du stencil $S = \{-3/2, -1/2, 1/2, 3/2\}$

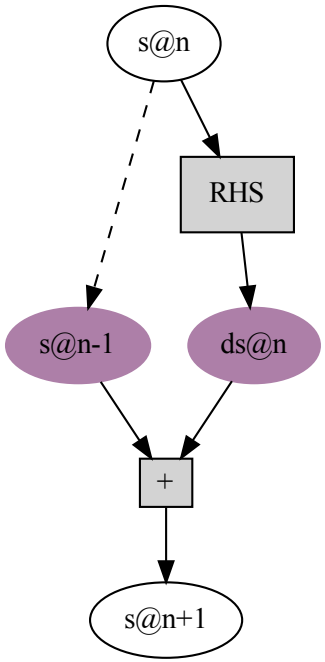


et l'interpolation en $x = 0$ associée (ordre 4)

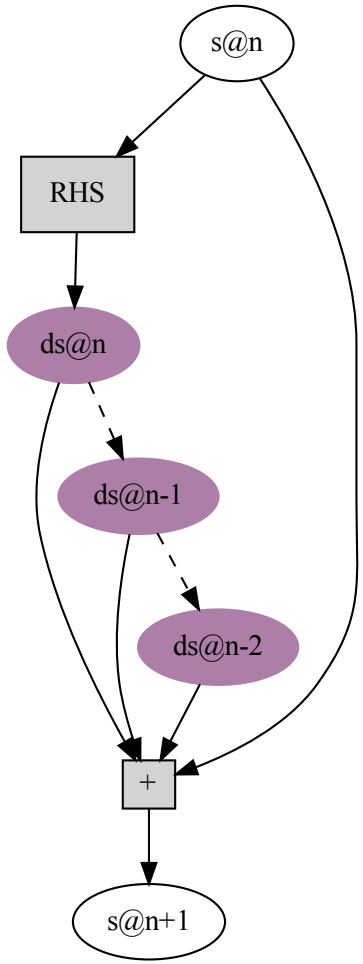
$$\phi_i^* = (-\phi_{i-3/2} + 9\phi_{i-1/2} + 9\phi_{i+1/2} - \phi_{i+3/2})/16$$

Quelques schémas d'intégration en temps

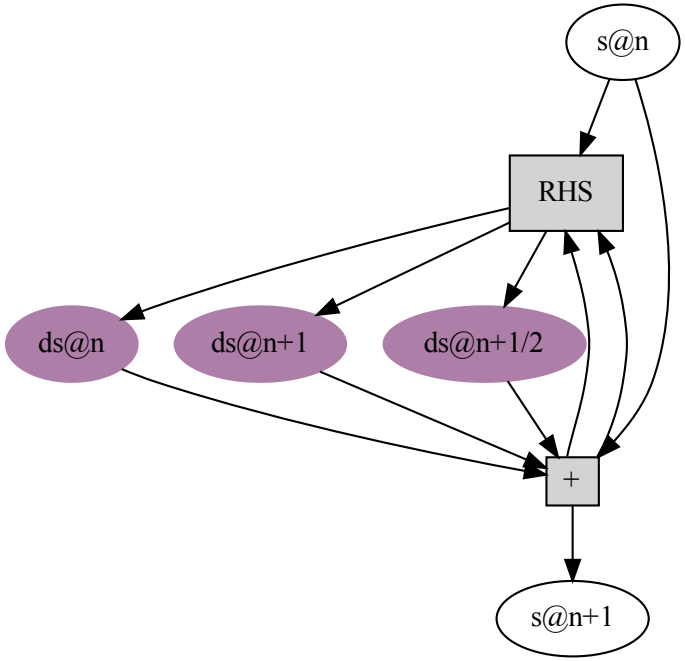
Leap-Frog, ordre 2



Adams Bashforth, ordre 3



RK3, ordre 3



En violet: les états intermédiaires à stocker.

Ingrédients d'un coeur dynamique

Un coeur dynamique de modèle c'est

des **données**
&
des **fonctions**

- L'organisation des données est cruciale.
- Les données devraient être regroupées en ensembles cohérents.
- Ensemble cohérent $\rightarrow$ définition de **types**.
- La création de types adaptés devrait être la règle.
- Exemple: le type `state` pour contenir tous les tableaux de l'état d'un modèle.

Ingrédients d'un coeur dynamique

Données

param	ensemble des paramètres utilisateurs
state	collections de tableaux
mesh	données concernant la grille, e.g. dx, dy, le masque, la bathymétrie, le paramètre de Coriolis, etc.
time	calendrier, t, indice d'itération etc.
integrator	collections de tableaux intermédiaires pour passer de n à $n+1$

```
class Model:
    def __init__(self, param):
        self.param = param
        self.mesh = Mesh(param)
        self.state = State(param, self.mesh)
        self.time = Time(param)
        self.integrator = Integrator(param, self.mesh, self.state)
```

Ingrédients d'un coeur dynamique

Fonctions

- qui opèrent sur l'état et retournent
- une **tendance** `ds = R(s)`
- ou qui modifient cet état `D(s)` [avec `s` "mutable"]
- chaque fonction combinant elle-même le résultat d'autres fonctions

Construction de briques

- structure modulaire
- voie pour avoir un continuum d'**abstraction**
- depuis des **fonctions bas niveau**, optimisées pour une architecture matérielle
- jusqu'à des **fonctions haut niveau**, très abstraites, e.g. `divflux`

Traduction en code du modèle d'Euler

Etat: **namedtuple** dont chaque clef est un *array*

```
    s = namedtuple("u", "U", "omega", "ke", "p")
```

Fonction pour la partie **prognostique**

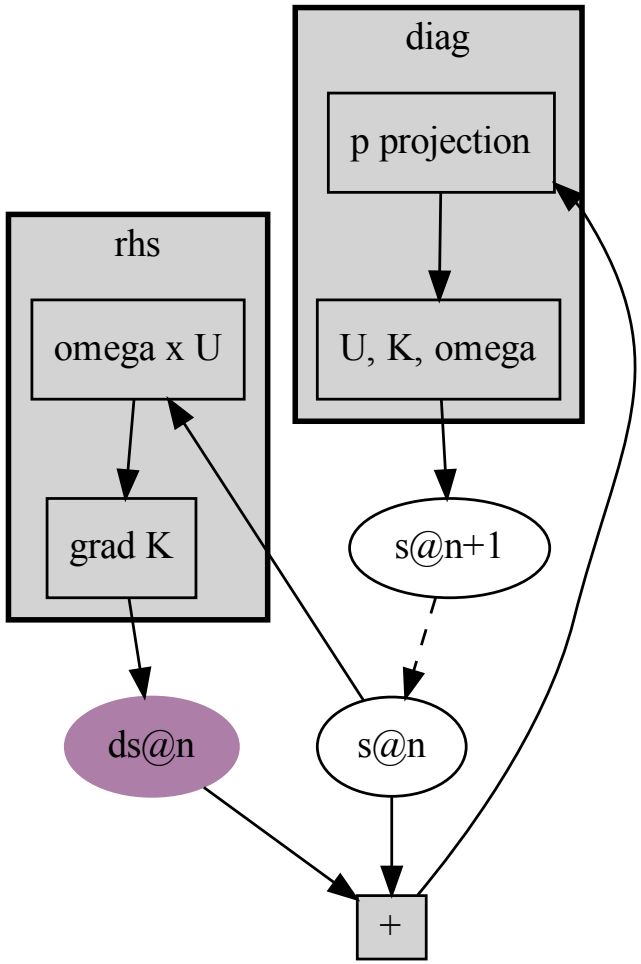
```
def rhs(s, ds):
    """ RHS for Euler model in momentum-pressure"""
    addvortexforce(param, mesh, s.U, s.omega, ds.u)
    addgrad(mesh, s.ke, ds.u)
```

Fonction pour la partie **diagnostique**

```
def diag(s):
    pressure_projection(mesh, s.U, s.div, s.p, s.u)
    sharp(mesh, s.u, s.U)
    compute_vorticity(mesh, s.u, s.omega)
    compute_kinetic_energy(param, mesh, s.u, s.U, s.ke)
```

Data flow

Flux des données pour le **modèle Euler**, avec un schéma . . . Euler avant ;-)



Le coeur des données est l'état s^n qu'on veut itérer en temps

généricité

cette structure permet le changement de

modèle via les blocks `rhs` et `diag`

schéma en temps via l'agencement des flèches & l'ajout d'autres `ds`

Variables globales ou locales ?

Comment `param` et `mesh` sont connus dans `rhs` ?

```
    def rhs(s, ds):
        """ RHS for Euler model in momentum-pressure"""
        addvortexforce(param, mesh, s.U, s.omega, ds.u)
        addgrad(mesh, s.ke, ds.u)
```

Fortran

- en les mettant dans des **modules**
- en important ces modules
- e.g. dans NEMO:
  `USE oce`
  `USE dom_oce`

Python

En utilisant une **coupure** (*closure*)

```
def get_euler(param, mesh):

    def rhs(s, ds):
        """ RHS for Euler model"""
        addvortexforce(param, mesh, s.U, s.omega, ds.u)
        addgrad(mesh, s.ke, ds.u)

    return rhs
```

Notez: fonction retournant une . . . fonction
Essayez en Fortran !

Adapation au modèle de Boussinesq

Ajout de “b” dans le namedtuple

```
    s = namedtuple("u", "b", "U", "omega", "ke", "p")
```

et ajout des deux termes dans la fonction prognostique

```
    def rhs(s, ds):
        """ RHS for Boussinesq model in momentum-pressure"""
        addvortexforce(param, mesh, s.U, s.omega, ds.u)
        addgrad(mesh, s.ke, ds.u)
        addbuoyancy(mesh, s.b, ds.u)
        divflux(param, mesh, s.flx, s.b, s.U, ds.b)
```

Et voilà ! . . . 3 lignes de code

Calcul du transport et niveaux d'abstraction

Terme $\boldsymbol{\nabla} \cdot (\mathbf{U}\, q) \rightarrow$ fonction `divflux`

Discrétisation en volumes finis

1. calcul des flux $(F_x, F_y) = (Uq, Vq)$
2. divergence

```
def divflux(param, mesh, flx, q, U, dq):
    method = param.compflux
    compflux(flx.x, U.x, q, mesh.oc.x, mesh.xshift, method)
    compflux(flx.y, U.y, q, mesh.oc.y, mesh.yshift, method)
    div(mesh, flx, dq)
```

Noter le passage du paramètre `param.compflux` qui permet de sélectionner la méthode numérique (upwind, centrée, WENO, etc.).

Dans le code CROCO le numérique est sélectionné via des clefs CPP `#define WENO`. *Pour changer de numérique, il faut recompiler le code.*

Calcul du flux en un point

Les trois **discrétisations linéaires** de base, d'ordre 1, 2 et 3
grille décalée: la vitesse `U` se trouve entre `qm` et `qp`

```
def up1(U, qm, qp):
    return U*qm if U > 0 else U*qp

def ce2(U, qm, qp):
    return U*(qm+qp)/2

def up3(U, qmm, qm, qp, qpp):
    return (U*(5*qm+2*qp-qmm)/6
            if U > 0 else
            U*(5*qp+2*qm-qpp)/6
            )

def ce4(U, qmm, qm, qp, qpp):
    return U*(7*(qm+qp)-(qmm+qpp))/12
```

- Schémas d'**ordre impair**: ils sont **upwindés**, dans le sens de U.
- Schémas d'**ordre pair**: ils sont centrés
- le "stencil" (nombre de q utilisés) augmente avec l'ordre

Reconstruction non-linéaire

Les discrétisations précédentes sont **linéaires**, en q. Il existe aussi des discrétisations/**schémas non-linéaires**.
par ex: WENO, limiteurs de flux (minmod, Van Leer etc.)

```
def weno3z(qmm, qm, qp):
    """ WENO3z: reconstruct q at mid-point between qm and qp """
    eps = 1e-14

    q1 = (3*qm-qqm)/2
    q2 = (qm + qp)/2

    beta1 = (qm-qmm)**2
    beta2 = (qp-qm)**2
    tau = np.abs(beta2-beta1)

    g1, g2 = 1/3, 2/3
    w1 = g1 * (1 + tau / (beta1 + eps))
    w2 = g2 * (1 + tau / (beta2 + eps))

    return (w1*q1 + w2*q2) / (w1 + w2)
```

intérêt : ils évitent le phénomène de dispersion = création d'oscillations

désavantage : ils sont plus coûteux en cpu

Calcul du flux pour tout le domaine

Il nous manque finalement la fonction intermédiaire qui réalise la boucle sur toutes les faces

```
def compflux_on_interval(flx, U, q, o, s, i0, i1):
    for i in range(i0, i1):
        if o[i] > 4:
            flx[i] = flx5(U[i], q[i-3*s], q[i-2*s], q[i-s], q[i], q[i+s], q[i+2*s])
        elif o[i] > 2:
            flx[i] = flx3(U[i], q[i-2*s], q[i-s], q[i], q[i+s])
        elif o[i] > 0:
            flx[i] = flx1(U[i], q[i-s], q[i])
        else:
            flx[i] = 0
```

Particularités

- la fonction travaille avec des tableaux dépliés (longs vecteurs)
- les voisins sont distants de `q[i]` de `s`, avec `s in [1,nx]`
- fonction valable quelque soit la direction, x ou y
- le tableau `o[i]` contient la taille du stencil

Opérateurs de différentiation

Les opérateurs à base de ∇ s'écrivent simplement

```
def addgrad(mesh, phi, du):
    du.x[:, 1:] -= np.diff(phi, axis=1)*mesh.mskx[:, 1:]
    du.y[1:, :] -= np.diff(phi, axis=0)*mesh.msky[1:, :]
```

```
def div(mesh, U, delta):
    delta[:, :-1] = -np.diff(U.x, axis=1)
    delta[:-1, :] -= np.diff(U.y, axis=0)
    delta *= mesh.msk
```

```
def compute_vorticity(mesh, u, omega):
    omega[1:,:] = -np.diff(u.x, axis=0)
    omega[:,1:] += np.diff(u.y, axis=1)
    omega *= mesh.mskv
```

Cette simplicité vient de la grille décalée (type C) qui permet

$$\text{points} \xrightarrow{\nabla} \text{segments} \xrightarrow{\nabla} \text{surfaces} \xrightarrow{\nabla} \text{volumes}$$

Les grands absents de ce cours

Avez-vous vu LES grands absents de ce cours ?

Les grands absents de ce cours

Les termes de diffusion et de viscosité

- Aux échelles OA, la diffusion et la viscosité moléculaire sont négligeables
- Ils sont responsables de la dissipation
- La dissipation est un ingrédient essentiel de la turbulence
- **Diffusion et viscosité sont paramétrisés** par une fermeture
- fermeture explicite vs dissipation implicite (avec WENO par exemple).
- la dissipation est un phénomène sous-maille.

Conclusion

Equations

- Nombreux systèmes d'équations
- Décrivant chacun un extrait d'une physique plus complète

Discrétisation

- Notion de grille décalée
- Théorie de l'interpolation à la base des schémas

Architecture

- Structure des données: élément essentiel
- Organisation en fonctions: modularité, abstraction

Perspectives

- Les coeurs des codes communautaires sont le fruit d'une histoire
- Lecture difficile $\rightarrow$ côté boite noire
- Les architectures sont largement périmées
- Besoin émergent: codes différentiables (backward) & tournant sur GPU
- Il existe des tentatives pour développer des nouveaux codes: Oceananigans de ClimaMachine, Google, ClimFlows . . .