

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

# **Modélisation et inversion en géophysique**

## **1 - Modélisation : Introduction**

Bernard Giroux  
([bernard.giroux@ete.inrs.ca](mailto:bernard.giroux@ete.inrs.ca))

Institut national de la recherche scientifique  
Centre Eau Terre Environnement

Version 1.2.3  
Hiver 2019

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

# Motivation

## Motivation

Règles d'or en  
modélisation

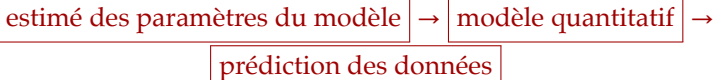
Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Dans ce cours, la modélisation fait référence au calcul de la réponse *géophysique* d'un modèle *numérique* donné.



- La modélisation directe peut servir à
  - 1 planifier et déterminer les paramètres de levés de terrain ;
  - 2 étudier la sensibilité des méthodes à une distribution donnée des propriétés du sous-sol, ou à une variation de ces propriétés dans le temps ;
  - 3 calculer la réponse des modèles lors de l'inversion.

## Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

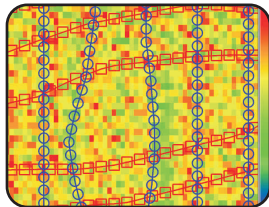
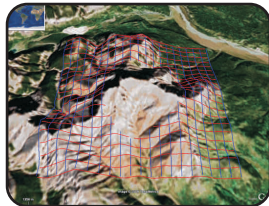
OMNI 3D® Workshop includes the following OMNI 3D®  
Layout Modules:



- Design & Edit Module
- Target Module
- Status Module
- 2D Ray Model Module
- Array Module
- 4D Module

## Advanced Analyses Module

- Assess 3D geometry effects on DMO, PSTM, multiples, and noise
- Analyze potential 3D geometry artifacts (footprints) using existing 2D seismic traces
- Interactive fold analysis
- Estimate PSTM illumination using Fresnel Zone binning
- Generate synthetic SEG Y data using survey geometry and a 3D model
- Build a depth cube of stack fold to analyze illumination at depth
- Analyze illumination on a subsurface horizon using any survey geometry



Source : <http://www.gedco.com>

## Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

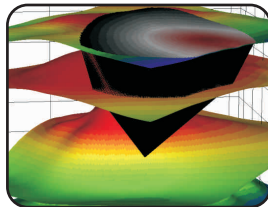
Discretisation

Programmation  
orientée objet

Références

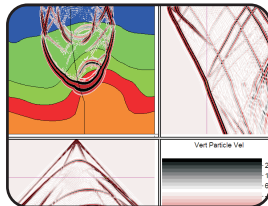
### 3D Ray Model Module

- Investigate parameters such as bin size, offsets, resolution, and imaging (migration) effects
- Build multi-layer 3D models including surface topography
- Create horizons using theoretical parameters or imported horizon data
- Model diffractions, reflections, and exploding horizons



### Elastic Wave Equation (EWE) Module

- Calculate Elastic or Acoustic Wave Equation response using a finite-difference solution
- Create full-waveform 2D synthetics using surface, VSP, OBC, and inter-well geometries
- Import model parameters from 2D Ray Models
- Add user-defined velocity gradients and heterogeneity
- Output real-time movies of shot wavefronts in Microsoft AVI format
- Monitor calculations interactively
- Will work on a multi-node cluster
- Built-in cluster manager to spread work across LANs



Source : <http://www.gedco.com>

# Motivation - Études de sensibilité

## Motivation

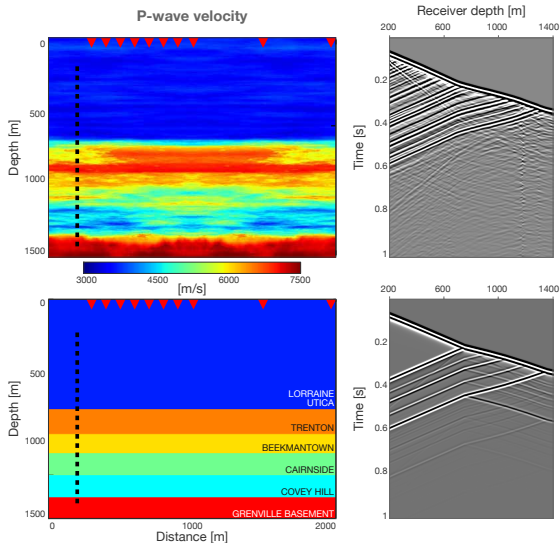
Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références



# Motivation - Études de sensibilité

## Motivation

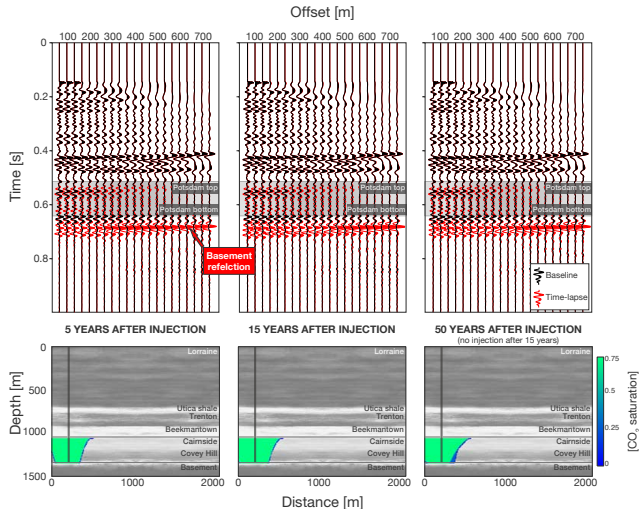
Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références



## Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

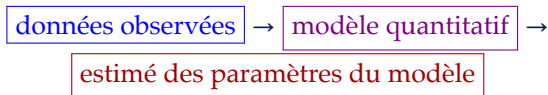
Programmation  
orientée objet

Références

- Problème direct



- Problème inverse





Motivation

**Règles d'or en  
modélisation**

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

# Règles d'or en modélisation

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Valider son code
  - par comparaison avec une solution analytique
  - par comparaison avec un autre code éprouvé
  - tester les cas extrêmes (forts contrastes de propriétés, position arbitraire des sources / récepteurs, limites du domaine de modélisation, ...)
- Connaître les limites des algorithmes et méthodes utilisés
- Connaître un débogueur pour le langage utilisé
  - Python : spyder est un IDE (*integrated development environment*) avec débogueur intégré
  - MATLAB : le débogueur est intégré à l'éditeur
- Optimiser la performance avec un outil de profilage
  - Python : modules `cProfile` ou `profile`
  - Dans MATLAB : `doc profile`

Motivation

Règles d'or en  
modélisation

**Notions utiles en  
calcul numérique**

Discrétisation

Programmation  
orientée objet

Références

# Notions utiles en calcul numérique

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Le fait de négliger la précision des calculs peut avoir des conséquences dramatiques.
- Trois cas tristement célèbres :
  - Des **erreurs d'arrondi** mal gérées auraient entraîné le mauvais fonctionnement d'un missile Patriot en 1991, causant la mort de 28 personnes ;
  - L'explosion de la fusée Ariane 5 en 1996 aurait été causée par une **erreur de dépassement** en assignant une valeur réelle à une variable déclarée comme un entier ;
  - Une modélisation numérique par éléments finis imprécise couplée à un mauvais encrage a causé le naufrage de la plate-forme pétrolière Sleipner A en 1991, causant des pertes de plus de 700 millions de dollars.

Source : <http://www.ima.umn.edu/~arnold/disasters/disasters.html>

# Précision des calculs

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Dans un ordinateur, les nombres sont représentés par un système *binaire*.
- Les entiers peuvent être représentés de façon exactes, en autant qu'ils se situent à l'intérieur d'une fourchette de valeurs données,
  - La largeur de cette fourchette dépend du nombre de *bits* utilisés pour représenter les nombres, e.g. pour 4 bits

$$0 = 0000_2 = 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

$$1 = 0001_2 = 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$4 = 0100_2 = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

$$15 = 1111_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Une **erreur de dépassement** surviendrait si on tentait s'assigner la valeur 16 à un entier 4 bits.

- Un bit est utilisé pour représenter le signe dans le cas d'entiers positifs et négatifs.
- Quelles sont les limites si on utilise 32 bits, 64 bits ?

- Les réels sont représentés en *virgule flottante*, en utilisant
  - un **signe**  $s$  ;
  - une **mantisse** (ou significande)  $m$  ;
  - un **exposant**  $e$  ;
 sous la forme

$$s \times m \times B^{e-E},$$

où  $B$  est la base de la représentation et  $E$  est le *biais* de l'exposant.

- La norme **IEEE 754** (la plus courante et celle utilisée par Python et MATLAB) définit  $B=2$ , et comporte deux formats : 32 bits et 64 bits, soit

Précision	$s$	$e$	$m$	$E$	Précision	Ch. significatifs
single (32 bits)	1 bit	8 bits	23 bits	127	24 bits	environ 7
double (64 bits)	1 bit	11 bits	52 bits	1023	53 bits	environ 16

- Avec la norme IEEE 754, la mantisse est “normalisée”, i.e. si les premiers bits du nombre en mantisse sont des zéros, on déplace vers la gauche (*left-shift*) les bits jusqu'au 1<sup>e</sup> bit égal à 1 (en ajustant l'exposant en conséquence)
  - comme ce bit est toujours 1, il n'est pas stocké et on augmente ainsi la précision d'un bit;
  - la mantisse a la valeur numérique  $1.f$ , où  $f$  est la fraction définie par les bits de  $m$ ;
    - pour la fraction  $f$ , la puissance de la base après le point est négative, i.e.  $1.1010_2$  est équivalent à  $1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4}$ , soit  $1 + 0.5 + 0.125 = 1.625$
- Quelques exemples de nombre représentés en double
  - 0 0111111111 0000 (+ 48 autres zéros) =  $+1 \times 2^{1023-1023} \times 1.0_2 = 1.$
  - 1 0111111111 0000 (+ 48 autres zéros) =  $-1 \times 2^{1023-1023} \times 1.0_2 = -1.$
  - 0 0111111111 1000 (+ 48 autres zéros) =  $+1 \times 2^{1023-1023} \times 1.1_2 = 1.5$
  - 0 1000000000 0000 (+ 48 autres zéros) =  $+1 \times 2^{1024-1023} \times 1.0_2 = 2.$
  - 0 1000000000 1010 (+ 48 autres zéros) =  $+1 \times 2^{1025-1023} \times 1.1010_2 = 6.5$

# Précision des calculs – Erreurs d'arrondi

## Motivation

## Règles d'or en modélisation

## Notions utiles en calcul numérique

## Discrétisation

## Programmation orientée objet

## Références

- Les réels ne sont pas toujours représentés de façon exacte,
  - e.g. évaluez le nombre 123456789 en simple précision.
  - Python : [https://github.com/bernard-giroux/geo1302/blob/master/p\\_erreur\\_single.ipynb](https://github.com/bernard-giroux/geo1302/blob/master/p_erreur_single.ipynb)
  - MATLAB : [https://github.com/bernard-giroux/geo1302/blob/master/m\\_erreur\\_single.ipynb](https://github.com/bernard-giroux/geo1302/blob/master/m_erreur_single.ipynb)
- Les opérations arithmétiques en virgule flottante ne sont pas exactes, mêmes si les nombres sont représentés exactement.
  - Par exemple, pour additionner deux entiers, les bits de la mantisse du plus petit nombre sont décalés vers la droite jusqu'à ce que l'exposant soit le même que celui du plus grand nombre, et on perd alors de la précision car des bits sont "perdus" à cause du décalage vers la droite.
- La **précision matérielle** (*machine accuracy*)  $\epsilon_m$  est définie comme le plus petit nombre qui, lorsqu'additionné à 1.0, donne une valeur différente de 1.0
  - Pour les float IEEE 754,  $\epsilon_m$  vaut environ  $1.19 \times 10^{-7}$
  - Pour les double IEEE 754,  $\epsilon_m$  vaut environ  $2.22 \times 10^{-16}$



# Précision – Erreurs d'arrondi

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- On peut voir  $\epsilon_m$  comme la précision de la partie fractionnelle, i.e. correspondant au changement causé par le bit le moins significatif de la mantisse.
- La quasi totalité des opérations arithmétiques entraîne une erreur d'au moins  $\epsilon_m$ .
- Ce type d'erreur est appelé **erreur d'arrondi** (*roundoff error*).
- Les erreurs d'arrondi s'accumulent au cours des calculs.
  - En étant chanceux,  $N$  opérations entraînent une erreur de  $\sqrt{N}\epsilon_m$ , si le signe des erreurs est aléatoire
  - Très fréquemment, il y a un biais de signe et on a alors  $N\epsilon_m$
  - Certaines opérations peuvent entraîner une erreur d'arrondi beaucoup plus grande que  $\epsilon_m$ , e.g. la soustraction de deux nombres très rapprochés donne un résultat avec peu de bits significatifs

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- En calcul numérique, il est fréquent de calculer une approximation “discrète” d’une quantité “continue” ;
  - e.g. une intégrale est évaluée numériquement en calculant une fonction à un nombre fini de points, plutôt qu’en “tout” point.
- Les **erreurs de troncation** surviennent lorsque le résultat obtenu par calcul numérique diffère de la réponse vraie.
- Contrairement aux erreur d’arrondi, il est possible de contrôler la magnitude des erreurs de troncation
  - e.g. en choisissant avec soin le nombre de points pour évaluer une intégrale.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Une méthode est dite instable lorsqu'une erreur d'arrondi introduite au début des calculs est progressivement amplifiée, au point de corrompre complètement la solution.
- Sur un ordinateur "parfait" une telle méthode serait utilisable.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Lors de l'exécution d'un code, la création de variables accapare une certaine quantité de mémoire vive (RAM).
  - Si le nombre de variables est élevé, toute la mémoire physique disponible peut être consommée;
  - Pour éviter le plantage du programme, le système d'exploitation utilise alors un espace disque (2 à 3 ordres de grandeur plus *lent* que la RAM) pour alouer plus de mémoire (*swap space*).
- La quantité de mémoire se mesure en octet (*byte*), où un octet vaut 8 bits.
- Par convention et abus de langage,
  - un kilooctet vaut 1024 ( $2^{10}$ ) octets (on devrait utiliser un kibioctet...);
  - un mégaoctet (mébioctet) vaut 1 048 576 ( $2^{20}$ ) octets;
  - un gigaoctet (gibioctet) vaut 1 073 741 824 ( $2^{30}$ ) octets;
  - un téraoctet (tébioctet) vaut 1 099 511 627 776 ( $2^{40}$ ) octets;
  - etc...

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Contrairement au C/C++ ou au FORTRAN, Python et MATLAB gèrent l'allocation de la mémoire de façon transparente.
- Avantage : simplifie la gestion des variables et accélère le développement ;
- Désavantage : peut affecter la performance car les opérations de gestion de mémoire sont "lentes"
  - Lorsque la taille des tableaux est connue, important de créer ces tableaux à la bonne taille ;
  - MATLAB : lors d'appels de fonctions, les variables en arguments sont passées par *référence*. **Si la variable est modifiée, une copie est créée automatiquement** (inutile si on retourne le résultat à la variable initiale).
- Désavantage : peut entraîner des bogues
  - Comme les variables ne sont pas *déclarées* avant d'être *initialisées*, on peut écraser le contenu d'une variable existante en créant une nouvelle variable sous le nom d'une variable existante (e.g. à l'intérieur d'une boucle).

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Désavantage : structures potentiellement inefficaces
  - MATLAB crée une entête pour chaque tableau, pour y stocker la taille du tableau et la classe de variable;
  - Pour des structures, MATLAB génère une entête pour la structure, pour *chaque* champ de la structure et pour *chaque* variable correspondant aux champs;
  - Soient deux variables contenant la même information

S1.R(1 : 100, 1 : 50)

S1.G(1 : 100, 1 : 50)

S1.B(1 : 100, 1 : 50)

et

S2(1 : 100, 1 : 50).R

S2(1 : 100, 1 : 50).G

S2(1 : 100, 1 : 50).B,

**S1 contient 7 entêtes et S2 contient 15 004 entêtes.**

# Gestion de la mémoire

Motivation

Règles d'or en  
modélisation

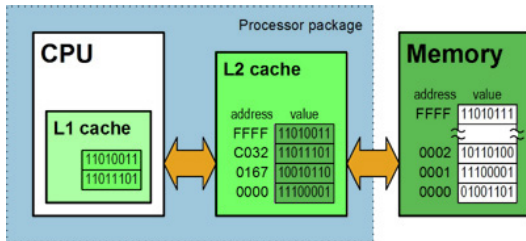
Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Désavantage (MATLAB) : pas de contrôle sur l'emploi de la mémoire *cache*
  - l'allocation manuelle de la mémoire permet de construire des codes *cache-friendly*;
  - En calcul haute performance, beaucoup d'efforts sont déployés pour optimiser l'utilisation de la mémoire cache.
- Python : quelques modules disponibles (cachetools), @functools.lru\_cache.



# Calcul sériel et calcul parallèle

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- **Calcul sériel** : les opérations sont effectuées les unes à la suite des autres ;
  - l'ordre dans lequel les opérations sont effectuées est prédictible.
- **Calcul parallèle** : des opérations sont effectuées simultanément, sur plusieurs coeurs, processeurs, ou noeuds de calcul ;
  - l'ordre des opérations ne peut pas être prédit.
- **Taxonomie de Flynn**
  - **SISD** (Single Instruction, Single Data) : PC jusqu'à la fin des années 90 ;
  - **SIMD** (Single Instruction, Multiple Data) : unités de calcul spécialisé (traitement du signal ou d'image) ;
  - **MIMD** (Multiple Instructions, Multiple Data) : processeurs multi-coeur ;
  - **MISD** (Multiple Instructions, Single Data) : beaucoup plus rarement utilisé.



Motivation

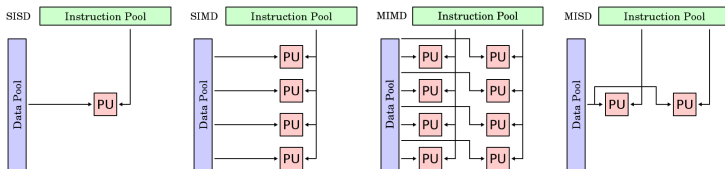
Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références



Source : C.M.L. Burnett

## ● Légende

- PU : Processeur (acronyme anglais de *Processing Unit*)
- *Instruction Pool* : ensemble des instructions disponibles pour le ou les PU
- *Data Pool* : ensemble des données nécessaires aux calculs

# Calcul sériel et calcul parallèle

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Tout les algorithmes ne sont parallélisables
- Différents schémas (*pattern*) de parallélisation existent
  - Décomposition par tâches (*task decomposition*);
  - Décomposition des données (*data decomposition*);
  - et plusieurs autres (voir Mattson *et al.*, 2004).
- Certains problèmes scientifiques sont propices au schéma de décomposition des données (les données sont séparées en groupes sur lesquels les opérations sont effectuées indépendamment) :
  - multiplication de matrices;
  - différences-finies.
- Le calcul scientifique parallèle n'est cependant pas limité au schéma de décomposition des données ;
  - Modélisation en parallèle de plusieurs tirs sismiques :  
décomposition par tâches.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Toutes les plateformes récentes permettent un certain degré de parallélisme (même vos téléphones intelligents).
- Plateformes courantes pour le calcul scientifique :
  - **un ordinateur** avec 1 ou quelques CPUs multi-coeurs;
    - la mémoire RAM est dite *partagée*, i.e. accessible à tous les coeurs;
    - architecture MIMD.
  - **une grappe d'ordinateurs** avec 2 ou plusieurs CPUs multi-coeurs;
    - la mémoire RAM est dite *distribuée*, i.e. chaque noeud n'a accès qu'à sa RAM.
    - architecture MIMD.
  - **GPU(s)** (*graphics processing unit*), ajouté(s) aux plateformes précédentes.
    - chaque GPU a sa propre RAM qui est indépendante de la RAM sur la carte-mère;
    - architecture SIMD.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

**Discrétisation**

Programmation  
orientée objet

Références

# Discrétisation

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- En science de la Terre en général, les milieux étudiés sont fortement hétérogènes ou de géométrie complexe ;
- Il est généralement impossible de trouver une solution analytique pour résoudre les problèmes les plus communs ;
- L'approche privilégiée consiste à représenter le milieu par une distribution discrète des propriétés physiques d'intérêt, i.e. le milieu est *discrétisé* ;
- Le choix du type de discrétisation dépend
  - des algorithmes disponibles pour solutionner le système,
  - du coût des calculs (temps et mémoire),
  - et de la précision recherchée.

Motivation

Règles d'or en  
modélisation

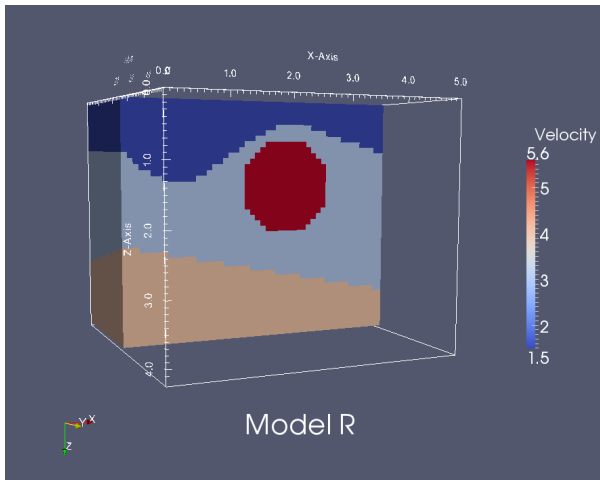
Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

## Grilles régulières



Motivation

Règles d'or en  
modélisation

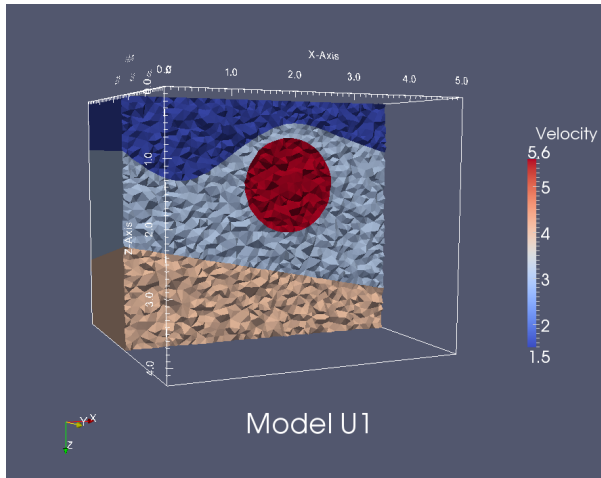
Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

## Maillages non structurés



Motivation

Règles d'or en  
modélisation

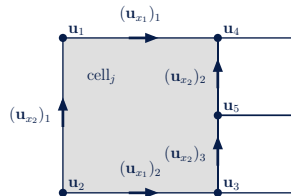
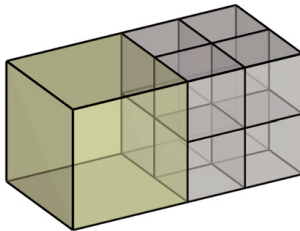
Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

## *Multi-grid*



Source : Haber *et al.*, 2007



Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- La génération de maillages non structurés est une opération pouvant s'avérer complexe.
- Plusieurs mailleurs ont été conçus pour accomplir cette tâche, dont
  - gmsh, tetgen (C++)
  - pygmsh, MeshPy, frentos (Python)
  - iso2mesh, DistMesh (MATLAB)et plusieurs autres dans la liste de la page <http://www.robertschneiders.de/meshgeneration/software.html>.
- En science de la Terre, des logiciels spécialisés comme
  - GOCAD
  - Petrelpeuvent également produire des maillages non structurés.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Les phénomènes géophysiques peuvent souvent être décrits par des équations aux dérivées partielles ;
- Les *différences finies* sont une façon d'approximer numériquement des dérivées ;
- La méthode des différences finies (MDF) constitue une approche conceptuellement simple et intuitive car le formalisme est proche de l'équation aux dérivées partielles ;
- Également, la mise en oeuvre est généralement simple car la MDF est implémentée sur des grilles régulières.

# Différences finies

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

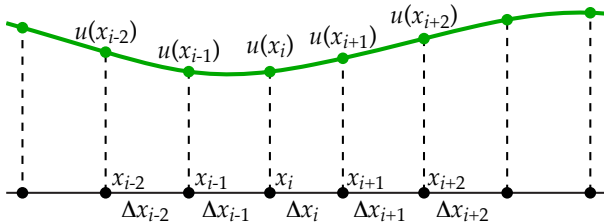
Programmation  
orientée objet

Références

- La définition classique de la dérivée d'une fonction  $u(x)$  est

$$u'(x) = \lim_{\Delta x \rightarrow 0} \frac{u(x + \Delta x) - u(x)}{\Delta x} \quad (1)$$

- Un ordinateur ne peut gérer la limite  $\Delta x \rightarrow 0$ ;
- L'idée est de définir un ensemble de points discrets  $x_i$ , où la fonction  $u$  est évaluée ( $u_i = u(x_i)$ );
- La distance entre les points est  $\Delta x_i = x_{i+1} - x_i$ , ou simplement  $\Delta x$  si la distance est constante partout.



# Différences finies

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Dans le cas discret, on a ainsi

$$u'(x_i) \approx \frac{u(x_i + \Delta x) - u(x_i)}{\Delta x} = \frac{u_{i+1} - u_i}{\Delta x} \quad (2)$$

qui est nommé **opérateur de différence avant**;

- Similairement, on a l'**opérateur de différence arrière** :

$$u'(x_i) \approx \frac{u(x_i) - u(x_i - \Delta x)}{\Delta x} = \frac{u_i - u_{i-1}}{\Delta x} \quad (3)$$

et l'**opérateur de différence centré** :

$$u'(x_i) \approx \frac{u(x_i + \Delta x) - u(x_i - \Delta x)}{2\Delta x} = \frac{u_{i+1} - u_{i-1}}{2\Delta x}. \quad (4)$$

- Dans le cas discret, ces expressions donnent un résultat différent : comment quantifier l'erreur ?
  - L'analyse par séries de Taylor peut nous donner la réponse.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Partons de l'identité

$$u(x) = u(x_i) + \int_{x_i}^x u'(s) ds \quad (5)$$

- Cette relation tient après dérivation, i.e.

$$u'(x) = u'(x_i) + \int_{x_i}^x u''(s) ds \quad (6)$$

- En insérant (6) dans (5) et en intégrant, on trouve

$$u(x) = u(x_i) + (x - x_i)u'(x_i) + \int_{x_i}^x \int_{x_i}^x u''(s) ds ds. \quad (7)$$

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- En répétant l'opération  $n$  fois, on trouve

$$u(x) = u(x_i) + (x - x_i)u'(x_i) + \frac{(x - x_i)^2}{2!}u''(x_i) + \cdots + \frac{(x - x_i)^n}{n!}u^{(n)}(x_i) + R_{n+1} \quad (8)$$

où

$$R_{n+1} = \int_{x_i}^x \cdots \int_{x_i}^x u^{(n+1)}(s) (ds)^{n+1}. \quad (9)$$

- L'équation (8) est la série de Taylor de la fonction  $u(x)$  en  $x_i$ .

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

Évaluons la précision de l'opérateur de différence avant.

- Développons la fonction  $u$  à  $x_{i+1}$  à partir de la série en  $x_i$ , soit

$$u(x_i + \Delta x_i) = u(x_i) + \Delta x_i \left. \frac{\partial u}{\partial x} \right|_{x_i} + \frac{\Delta x_i^2}{2!} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} + \frac{\Delta x_i^3}{3!} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \dots \quad (10)$$

- En réarrangeant on trouve

$$\frac{u(x_i + \Delta x_i) - u(x_i)}{\Delta x_i} - \left. \frac{\partial u}{\partial x} \right|_{x_i} = \underbrace{\frac{\Delta x_i}{2!} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} + \frac{\Delta x_i^2}{3!} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \dots}_{\text{erreur de troncation}} \quad (11)$$

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- L'erreur de troncation est la différence entre la dérivée exacte et sa représentation discrète
  - Pour des fonctions lisses et un pas  $\Delta x_i$  petit, le 1<sup>e</sup> terme est utilisé pour caractériser l'ordre de grandeur de l'erreur ;
  - Ce terme dépend de  $\frac{\partial^2 u}{\partial x^2}$  et de  $\Delta x$ , ce dernier seulement sur lequel il est possible d'agir pour réduire l'erreur ;
  - La notation  $O(\Delta x_i)$  est utilisée pour représenter le 1<sup>e</sup> terme de l'erreur de troncation, i.e.

$$\left. \frac{\partial u}{\partial x} \right|_{x_i} = \frac{u(x_i + \Delta x_i) - u(x_i)}{\Delta x_i} + O(\Delta x). \quad (12)$$

- Pour l'opérateur de différence arrière, on trouve également une erreur en  $O(\Delta x)$ , soit

$$\frac{u(x_i) - u(x_i - \Delta x_{i-1})}{\Delta x_{i-1}} - \left. \frac{\partial u}{\partial x} \right|_{x_i} = \underbrace{-\frac{\Delta x_{i-1}}{2!} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} - \frac{\Delta x_{i-1}^2}{3!} \left. \frac{\partial^3 u}{\partial x^3} \right|_{x_i} + \dots}_{\text{erreur de troncation}} \quad (13)$$



Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Pour l'opérateur de différence centré, l'erreur de troncation est obtenu en combinant les équations (10) et (13), ce qui donne (si  $\Delta x_{i-1} = \Delta x_i = \Delta x$ )

$$\frac{u_{i+1} - u_{i-1}}{2\Delta x} - \frac{\partial u}{\partial x} \Big|_{x_i} = \frac{\Delta x^2}{3!} \frac{\partial^3 u}{\partial x^3} \Big|_{x_i} + \dots \quad (14)$$

- On a alors une convergence quadratique :

$$\frac{\partial u}{\partial x} \Big|_{x_i} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O(\Delta x^2). \quad (15)$$

- Puisqu'on examine la convergence pour  $\Delta x \rightarrow 0$ , on a que  $O(\Delta x^2) < O(\Delta x)$  ;
  - l'erreur est plus *faible* en utilisant un opérateur centré qu'en utilisant un opérateur avant ou arrière.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

## Exercice sous Python/MATLAB

- Évaluez la fonction sinus dans l'intervalle  $[0, 4\pi]$  avec un pas de  $\pi/10$ .
- Calculez la dérivée
  - avec l'opérateur avant;
  - avec l'opérateur centré.
- Pour les deux dérivées discrètes, calculez l'erreur avec la solution analytique.
- Répétez avec un pas de  $\pi/20$ .
- Comment varie l'erreur en fonction de l'opérateur et en fonction du pas?

# Différences finies - Ordres supérieurs

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Les opérateurs précédents sont définis avec deux points, on dit qu'ils sont d'ordre 2.
- Que se produit-il si on utilise 4 points ?
- Partons de l'équation (14)

$$\frac{u_{i+1} - u_{i-1}}{2\Delta x} = \frac{\partial u}{\partial x} + \frac{\Delta x^2}{3!} \frac{\partial^3 u}{\partial x^3} + \frac{\Delta x^4}{5!} \frac{\partial^5 u}{\partial x^5} + \frac{\Delta x^6}{7!} \frac{\partial^7 u}{\partial x^7} + \dots + \frac{\Delta x^{2m}}{(2m+1)!} \frac{\partial^{(2m+1)} u}{\partial x^{(2m+1)}} + \dots \quad (16)$$

- Utilisons maintenant les points  $x_{i-2}$  et  $x_{i+2}$ , et remplaçons  $\Delta x$  par  $2\Delta x$  :

$$\frac{u_{i+2} - u_{i-2}}{4\Delta x} = \frac{\partial u}{\partial x} + \frac{(2\Delta x)^2}{3!} \frac{\partial^3 u}{\partial x^3} + \frac{(2\Delta x)^4}{5!} \frac{\partial^5 u}{\partial x^5} + \frac{(2\Delta x)^6}{7!} \frac{\partial^7 u}{\partial x^7} + \dots \quad (17)$$

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- En multipliant (16) par  $2^2$  et en soustrayant le résultat à (17), on trouve

$$\frac{8(u_{i+1} - u_{i-1}) - (u_{i+2} - u_{i-2})}{12\Delta x} = \frac{\partial u}{\partial x} - \frac{4\Delta x^4}{5!} \frac{\partial^5 u}{\partial x^5} - \frac{20\Delta x^6}{7!} \frac{\partial^7 u}{\partial x^7} + \dots \quad (18)$$

- On remarque que l'erreur de troncation est maintenant d'ordre 4 :

$$\frac{\partial u}{\partial x} = \frac{u_{i-2} - 8u_{i-1} + 8u_{i+1} - u_{i+2}}{12\Delta x} + O(\Delta x^4) \quad (19)$$

- De façon générale, **plus l'ordre de l'opérateur est élevé, plus faible est l'erreur** (Fornberg, 1998).

# Différences finies - Exemple

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Considérons la distribution verticale de la température dans la croûte terrestre.
- En l'absence de variations latérales, l'équation de la chaleur en régime permanent est

$$\frac{d}{dz} \left( \lambda(z) \frac{dT}{dz} \right) = -A(z), \quad (20)$$

où

- $\lambda$  est la conductivité thermique (W/m/K);
- $T$  est la température (K);
- $A$  est la production de chaleur (W/m<sup>3</sup>);
- $z$  est la profondeur (m).

# Différences finies - Exemple

Motivation

Règles d'or en  
modélisation

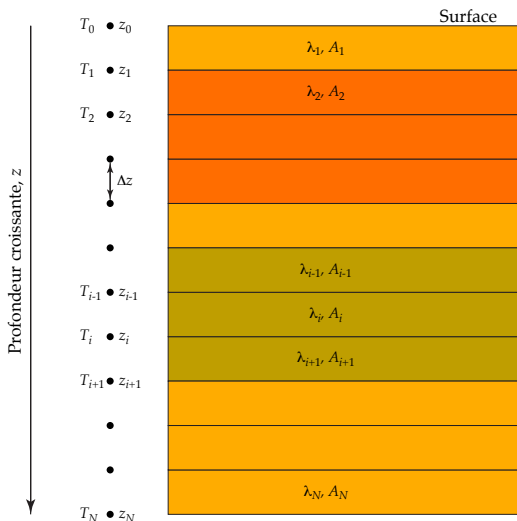
Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

Discrétisation du  
milieu en  $N$   
couches  
homogènes



# Différences finies - Exemple

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Pour solutionner le problème, il est nécessaire de poser des conditions aux limites;
- Considérons
  - une température connue en surface,  $T_0$ ;
  - un flux de chaleur  $Q$  dans la  $N^e$  couche.
- L'objectif est de **déterminer les valeurs de température aux noeuds de la grille.**

# Différences finies - Exemple

Motivation

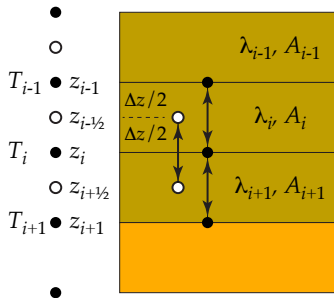
Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références



- Considérons le  $i^e$  noeud.
- En choisissant un opérateur centré de largeur  $\Delta z/2$ , la dérivée du terme entre parenthèses de l'équation (20) vaut

$$\left. \frac{d}{dz} \left( \lambda(z) \frac{dT}{dz} \right) \right|_{z_i} = \frac{\lambda_{i+1} \left. \frac{dT}{dz} \right|_{z_{i+1/2}} - \lambda_i \left. \frac{dT}{dz} \right|_{z_{i-1/2}}}{\Delta z} \quad (21)$$



# Différences finies - Exemple

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Avec également un opérateur centré de largeur  $\Delta z/2$  pour la dérivée de la température, on trouve

$$\frac{d}{dz} \left( \lambda(z) \frac{dT}{dz} \right) \Big|_{z_i} = \frac{\lambda_{i+1} \left( \frac{T_{i+1} - T_i}{\Delta z} \right) - \lambda_i \left( \frac{T_i - T_{i-1}}{\Delta z} \right)}{\Delta z} \quad (22)$$

- En réarrangeant le terme de droite, on trouve

$$\frac{\lambda_i}{\Delta z^2} T_{i-1} - \frac{\lambda_{i+1} + \lambda_i}{\Delta z^2} T_i + \frac{\lambda_{i+1}}{\Delta z^2} T_{i+1} \quad (23)$$

- L'équation (23) est égale à la production de chaleur à  $z_i$ , soit  $A(z)|_{z_i}$ 
  - Puisque  $A$  est défini pour les couches et non aux noeuds, on prends la moyenne entre deux couches, i.e.

$$A(z)|_{z_i} = \frac{A_i + A_{i+1}}{2} \quad (24)$$

- Les équations (23) et (24) vont nous permettre de construire un système matriciel de la forme  $\mathbf{Ax} = \mathbf{b}$  où le vecteur  $\mathbf{x}$  est le vecteur des températures.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

Mais

- L'équation (23) contient le terme  $T_{i-1}$ , ce qui pose un problème en surface
  - Heureusement, on sait que la température  $T_0$  est déjà connue étant donnée les conditions aux frontières posées;
  - C'est une condition dite *de Dirichlet*, i.e. la valeur que la solution doit vérifier sur les frontières est spécifiée.

# Différences finies - Exemple

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- En profondeur, le flux  $Q$  est connu.
  - Par définition, le flux de chaleur vaut

$$Q = -\lambda \frac{\partial T}{\partial z} \quad (25)$$

- Évalué dans la  $N^e$  couche avec un opérateur de différence arrière, on a

$$Q = -\lambda_N \frac{T_N - T_{N-1}}{\Delta z} \quad (26)$$

ou bien

$$\frac{\lambda_N}{\Delta z} T_{N-1} - \frac{\lambda_N}{\Delta z} T_N = Q. \quad (27)$$

- Lorsqu'on spécifie les valeurs des *dérivées* que la solution doit vérifier sur les frontières, on a une condition dite *de Neumann*.

# Différences finies - Exemple

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Le système matriciel peut maintenant se construire :

$$\begin{bmatrix}
 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
 \frac{\lambda_1}{\Delta z^2} & -\frac{\lambda_2 + \lambda_1}{\Delta z^2} & \frac{\lambda_2}{\Delta z^2} & 0 & \dots & 0 & 0 & 0 \\
 0 & \frac{\lambda_2}{\Delta z^2} & -\frac{\lambda_3 + \lambda_2}{\Delta z^2} & \frac{\lambda_3}{\Delta z^2} & \dots & 0 & 0 & 0 \\
 \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
 0 & 0 & 0 & 0 & \dots & \frac{\lambda_{N-1}}{\Delta z^2} & -\frac{\lambda_N + \lambda_{N-1}}{\Delta z^2} & \frac{\lambda_N}{\Delta z^2} \\
 0 & 0 & 0 & 0 & \dots & 0 & \frac{\lambda_N}{\Delta z} & -\frac{\lambda_N}{\Delta z}
 \end{bmatrix}
 \begin{bmatrix}
 T_0 \\
 T_1 \\
 T_2 \\
 T_3 \\
 \vdots \\
 T_{N-2} \\
 T_{N-1} \\
 T_N
 \end{bmatrix}
 =
 \begin{bmatrix}
 T_0 \\
 -\frac{A_1 + A_2}{2} \\
 -\frac{A_2 + A_3}{2} \\
 \vdots \\
 -\frac{A_{N-1} + A_N}{2} \\
 Q
 \end{bmatrix}
 \quad (28)$$

où les valeurs de production de chaleur  $A_i$  sont interpolées aux noeuds.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

## Exercice sous Python/MATLAB

- Partant des conditions

- $T_0 = 280 \text{ K}$ ;
- $Q = -55 \text{ mW/m}^2$ ;

et considérant un milieu à 20 couches de 100 m d'épaisseur chacune.

- Calculez le profil vertical de température si

$\lambda = \{1.8, 1.8, 1.8, 1.8, 1.8, 3.7, 3.7, 3.7, 3.7, 3.7, 2.4, 2.4, 2.4, 2.4, 2.4, 2.4, 3.5, 3.5, 3.5, 3.5\}$

et

$A = \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2.8 \times 10^{-6}, 2.8 \times 10^{-6}, 2.8 \times 10^{-6}, 2.8 \times 10^{-6}\}$

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

**Programmation  
orientée objet**

Références

# Programmation orientée objet

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Le développement de logiciels d'envergure moyenne à grande pose plusieurs défis sur le plan de la programmation :
  - structure cohérente entre les éléments;
  - maintenance de cette structure;
  - organisation du code.
- La programmation orientée objet (POO) offre plusieurs avantages pour faciliter ces développements :
  - modularité;
  - abstraction;
  - encapsulation;
  - sûreté;
  - productivité et réutilisabilité.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Division d'un logiciel en modules indépendants ;
- Ces modules regroupent les **données** et les **opérations associées** ;
  - Exemple : un module Maillage contenant
    - les coordonnées des noeuds,
    - les indices des noeuds formant les faces,
    - une fonction pour importer un maillage à partir d'un fichier,
    - une fonction pour retourner les limites du maillage,
    - une fonction pour retourner le nombre de faces ou de voxels,
    - etc
- **Classe** : formalisme permettant de représenter le module.
- **Objet** : représentation du module, *instance* de la classe qui existe dans la mémoire de l'ordinateur.
- La modularité permet une organisation hiérarchique du code.



Motivation

Règles d'or en  
modélisation

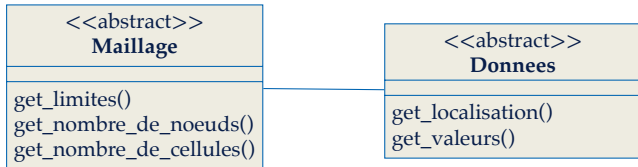
Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Abstraction :
  - La **spécification des fonctionnalités** d'une classe est définie ;
    - Classe dite *abstraite*
    - Les fonctionnalités constitue *l'interface*
  - La *représentation* (l'implémentation) n'est pas définie ;
    - l'implémentation se fait dans une classe *dérivée*.
- Il ne peut exister une instance d'une classe abstraite.
- L'abstraction permet
  - de définir de façon générale les relations entre les classes,
  - permet de définir une structure cohérente pour un programme/logiciel.



Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Encapsulation : fonctionnalité du langage qui permet la modularité;
  - Une classe permet de lier des variables, ou **attributs**, à des fonctions, ou **méthodes**
  - Un objet contient les valeurs des variables, les fonctions appelées par l'objet produisent un résultats propre à ses valeurs.
- Permet de changer librement l'implémentation.
- Possibilité de **restreindre l'accès direct** aux données ;
  - Des variables *privées* ne sont accessibles que par les méthodes de la classe.
  - Tenter d'accéder aux variables privées produit une erreur.
  - Permet de produire un code plus "sûr".

Motivation

Règles d'or en  
modélisation

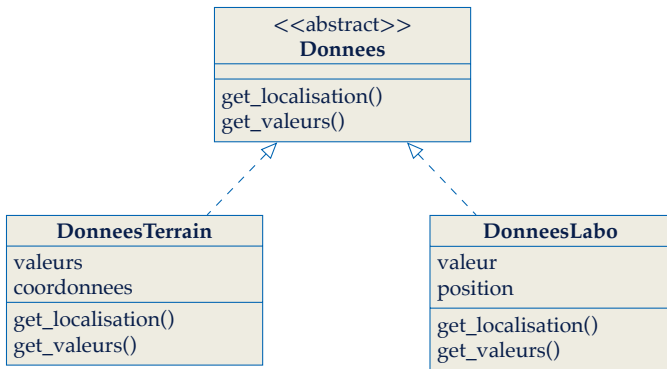
Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- La POO favorise *en principe* la productivité et la réutilisabilité
  - Travail en équipe : la modularité simplifie l'organisation du code et sa gestion et peut améliorer la productivité.
  - L'abstraction et l'héritage permettent, dans une certaine mesure, la réutilisabilité du code.



Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Chaque classe possède des caractéristiques (attributs et méthodes) qui lui sont propres.
- On dit d'une classe dérivée qu'elle hérite de la classe *mère*
  - la classe *filles* peut alors utiliser les caractéristiques de la classe mère;
  - le code écrit pour la classe mère *n'a pas* à être écrit à nouveau pour la classe fille;
  - une classe fille *peut* réimplémenter une méthode de la classe mère.
- Propriétés de l'héritage :
  - Transitivité : si B hérite de A et si C hérite de B alors C hérite de A
  - Non réflexif : une classe ne peut hériter d'elle-même
  - Non symétrique : si A hérite de B, B n'hérite pas de A
  - Sans cycle : il n'est pas possible que B hérite de A, C hérite de B et que A hérite de C.

Motivation

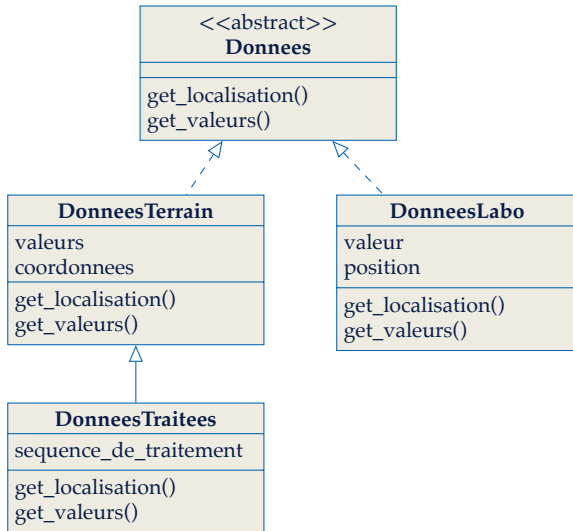
Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références



Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Avec le polymorphisme, une interface est partagée par des classes différentes;
  - e.g. : multiplication définie pour des scalaires, des matrices ou des vecteurs.
- Sortes de polymorphisme :
  - *Ad hoc* : interface implémentée par plusieurs fonctions de même nom ayant des arguments différents;
  - paramétré : interface implémentée par une seule fonction prenant en arguments un type *générique* pour chaque entité (templates en C++);
  - par sous-typage : classes filles possédant chacune son implémentation d'une méthode de la classe mère.

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

# Références

Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discrétisation

Programmation  
orientée objet

Références

- Fornberg, B. (1998). *A Practical Guide to Pseudospectral Methods*, volume 1 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press
- Franek, F. (2004). *Memory as a Programming Concept in C and C++*. Cambridge University Press
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley



Motivation

Règles d'or en  
modélisation

Notions utiles en  
calcul numérique

Discretisation

Programmation  
orientée objet

Références

- Karniadakis, G. E. and Kirby, II, R. M. (2003). *Parallel Scientific Computing in C++ and MPI*. Cambridge University Press
- Mattson, T. G., Sanders, B. A., and Massingill, B. L. (2004). *Patterns for Parallel Programming*. Software Patterns Series. Addison-Wesley
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 2<sup>nd</sup> edition