Projet : consignes

Version du 16 novembre 2021

Echéances

1. **Au début de la première séance de projet** (semaine du 22 au 27 novembre 2021), vous devrez envoyer un courriel par binôme ou monôme à votre encadrant de TP (avec comme champ *Sujet* « LU2ME005-F : groupe projet ») en indiquant dans le message
   — le ou les noms et prénoms du ou des étudiants,
   — le ou les numeros d'étudiant correspondants.

---

2. **AVANT la fin de la deuxième séance de projet** (du 29 novembre au 3 décembre), vous devrez placer sur Moodle **(un seul dépôt par binôme)**
   (a) le code (fichiers sources Fortran 90 uniquement, pas de fichiers exécutables) correspondant à la « première version » du sujet.
   (b) un fichier texte nommé README.txt qui contient la composition du groupe ainsi que la ou les instructions de compilation de cette première version si vous avez travaillé avec un compilateur particulier (par exemple `g95 -o proj proj.f90`).

---

3. **AVANT la fin de la dernière séance de projet** (du 3 au 7 janvier 2022), vous devrez placer sur Moodle **(un seul dépôt par binôme)**
   (a) **le rapport** sous forme électronique (**format pdf uniquement**) ;
   (b) **les fichiers source Fortran 90** (pas de fichiers exécutables). Ces fichiers sources seront éventuellement accompagnés des autres fichiers utiles à votre projet (fichiers de paramètres nécessaires à l'exécution, ...).
   (c) un fichier texte nommé README.txt qui contient la composition du binôme ainsi que la ou les instructions de compilation de votre projet.

Avant de rendre la première version

Vérifiez que vous avez bien appliqué les instructions ci-dessous.

- ☐ Le programme est bien indenté.
- ☐ Le programme est bien commenté.
- ☐ La compilation ne génère pas d'erreur.
- ☐ L'exécution ne génère pas d'erreur.
- ☐ Nous avons/j'ai placé dans la boîte de dépôt le programme FORTRAN `.f90` (et pas l'exécutable).
- ☐ Nous avons/j'ai placé dans la boîte de dépôt le fichier README.txt avec la composition du groupe.

Rapport

Le rapport comportera environ 6 pages images comprises, sans faute d'orthographe, bâti sur le plan suivant :

— **Page de garde** – Noms et numéros d'étudiant, année, formation, UE, groupe, titre du projet.

— **Introduction** (une demi-page maximum) – Sujet de l'étude (1 ou 2 phrases), configurations étudiées, état du projet : dire si vous êtes repartis de la correction de la première version, aboutissement du projet, type de résultats obtenus et résumé des principales difficultés rencontrées.

— **Programmation** (1,5 page maximum) – Travail de programmation réalisé, choix faits, éventuels algorithmes, rôle des différents fichiers

Ne pas mettre de code FORTRAN dans le rapport.

— **Résultats** (1,5 page maximum) – Résultats obtenus, commentaires, interprétations.

— **Améliorations** (1 ou 2 pages) – Brièvement : choix faits, solutions utilisées, résultats obtenus.

— **Conclusion** (une demi-page maximum) – Résumé du travail effectué et des perspectives.

Comme pour tout rapport, les figures doivent être numérotées et légendées. Le texte devra faire référence à TOUTES les figures (voir par exemple comment sont traitées les figures dans l'énoncé du projet).

Avant de rendre la version finale

Vérifiez que vous avez bien appliqué les instructions ci-dessous.

☐ Le rapport suit le plan donné dans les consignes ainsi que les limitations en nombre de pages.

☐ Le rapport ne comporte aucune faute d'orthographe.

☐ Les figures sont numérotées, légendées et sont toutes référencées dans le texte.

☐ Nous avons/j'ai placé dans la boîte de dépôt le rapport au format `pdf`.

☐ Les programmes sont bien indentés.

☐ Les programmes sont bien commentés.

☐ La compilation ne génère pas d'erreur.

☐ L'exécution ne génère pas d'erreur.

☐ Nous avons/j'ai placé dans la boîte de dépôt les sources FORTRAN `.f90`.

☐ Nous avons/j'ai placé dans la boîte de dépôt au moins un fichier de données (`param.dat`).

☐ Nous avons/j'ai placé dans la boîte de dépôt le fichier README.txt avec la composition du groupe.

☐ Nous n'avons pas/je n'ai pas placé dans la boîte de dépôt une archive (format `zip`, `rar`, `7z`, ...) qui compliquerait la tâche du correcteur et le mettrait forcément de mauvaise humeur.

Notation

— Le projet sera noté sur 25. Pour dépasser 20/25 (c'est-à-dire 16/20), la partie « Améliorations » devra nécessairement avoir été abordée.

— Si le projet est fait en binôme, votre note sera individuelle. Le travail doit donc être réparti équitablement entre les deux membres du binôme. De plus, **une brève interrogation orale sur votre code aura lieu lors de la dernière séance** : chacun des membres du binôme doit pouvoir répondre à toutes les questions. Une absence lors de ces séances sera pénalisante.

— En ce qui concerne le plagiat : il est évident que vous devez avoir écrit vous-même votre code et vous devez donc être capable de l'expliquer **entièrement**. Si des similarités entre deux projets sont détectées, une pénalité sera appliquée aux deux projets.

— L'aspect **opérationnel** du code est un point très important : il est nettement préférable de présenter un code aux possibilités restreintes mais qui compile et s'exécute correctement, que de présenter un code aux possiblités étendues dans le rapport alors que le code présente des erreurs à l'exécution ou pire, à la compilation.

— La qualité de rédaction du rapport (**sans faute d'orthographe**) ainsi que la lisibilité du code Fortran (**indentation et commentaires**) seront prises en compte dans la note de projet.

— La partie « Améliorations » vous donne des pistes pour améliorer votre projet. Vous pouvez bien sûr proposer vos propres améliorations ou adapter celles qui sont proposées.

Projet : simulation d'un écoulement granulaire

Version du 16 novembre 2021

Présentation

Les milieux granulaires sont constitués de particules solides de taille typique supérieure au dixième de millimètre (des particules plus petites forment des poudres, ou au-dessous du micron, des colloïdes). Ils sont très courants dans la nature (sable, boues, pierriers, anneaux des planètes), dans la vie de tous les jours (sucre, riz), dans les processus industriels (second type de matériau après l'eau, on les trouve en chimie, pharmacie, construction, ...) et dans l'agriculture (grains). Ils ont un comportement multiforme : solides et très résistants au repos, ils peuvent s'écouler comme des liquides ou des gaz quand ils sont en mouvement. Il existe une recherche très active pour modéliser ces matériaux, recherche complexe car les comportements des milieux granulaires sont très divers.

FIGURE 1 – Tas de sable en formation.

Le projet propose de simuler numériquement la dynamique d'un tas de sable (figure 1). Ce tas est alimenté régulièrement en grains, il grossit et se réorganise suite aux avalanches qui s'y produisent. Nous allons utiliser un modèle extrêmement simplifié, mais qui comporte certaines propriétés physiques assez réalistes. Le modèle original, appelé BTW du nom de ses inventeurs Per Bak, Chao Tang & Kurt Wiesenfeld, est exposé dans un article intitulé *Self-organized criticality* paru en 1988 dans la revue *Physical Review A*. Il peut être appliqué à la croissance d'un tas bi-dimensionnel (entièrement compris dans un plan vertical) ou tri-dimensionnel (le cas réel). Nous utiliserons ici un modèle bi-dimensionnel un peu modifié par rapport à celui de l'article.

Ce type de modèle est apparenté aux *automates cellulaires*. Dans un premier temps, on ne simule qu'une des moitiés du tas de sable (suivant un rayon). Les hypothèses sont les suivantes (voir aussi l'exemple de la figure 2) :

1. le tas de sable est représenté par une juxtaposition de $r_{\max}$ piles, numérotées de 0 à $r_{\max} - 1$, de hauteur proportionnelle au nombre de grains qu'elles comportent. Le tas occupe donc la partie basse d'une grille régulière dans laquelle on aurait placé un grain par case ;

2. on note p_i le nombre de grains dans la i-ème pile. Le tas évolue dans le temps (à chaque *itération temporelle*, on dit aussi plus familièrement à chaque *pas de temps*) suivant la règle :

   Pour i de 0 à $r_{\max} - 2$: si $p_i - p_{i+1} \geq 2$, alors n_g grains sont transférés de la pile i à la pile $i+1$;

3. le nombre de grains transféré de la pile i à la pile $i+1$ est aléatoire (on dit que le modèle est *stochastique*) et donné par
   $$n_g = 1 + \lfloor \tfrac{1}{2}(2 + p_i - p_{i+1})\xi \rfloor$$
   où ξ désigne un réel aléatoirement choisi dans l'intervalle $[0, 1[$, et où $\lfloor x \rfloor$ désigne la partie entière de x ;
4. on introduit un grain de sable supplémentaire dans la pile 0 tous les n_t pas de temps ;
5. on s'arrête lorsqu'une pile atteint une hauteur $h_{\max}$ fixée à l'avance.

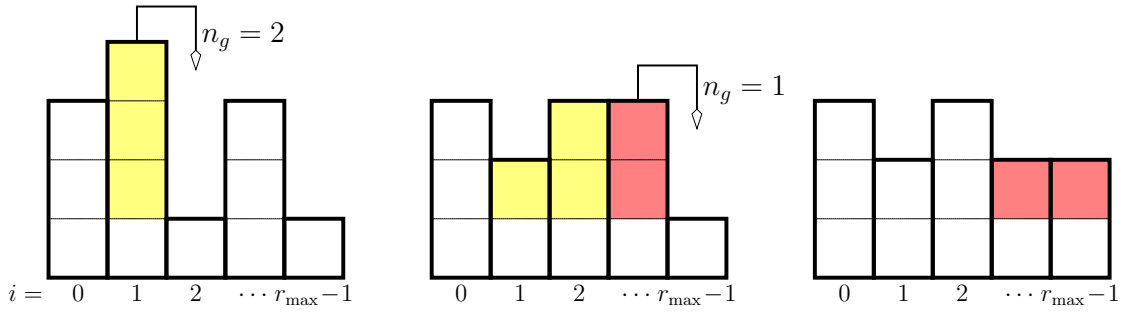

FIGURE 2 – Exemple d'évolution du modèle sur un pas de temps : dans le premier tas, on a $p_0 - p_1 = 3 - 4 = -1 < 2$, on n'a donc pas de transfert de grains de la pile 0 vers la pile 1 ; en revanche, on a $p_1 - p_2 = 4 - 1 = 3 \geq 2$, ce qui va provoquer le transfert de n_g grains de la pile 1 vers la pile 2, avec par exemple ici $n_g = 2$. On traite ensuite de la même façon le transfert de grains entre les piles 2 et 3, puis 3 et 4, ..., $r_{\max} - 2$ et $r_{\max} - 1$.

Avec ces règles, les grains ne peuvent que s'accumuler dans la dernière pile $i = r_{\max} - 1$: le tas de sable est confiné, comme si une paroi était présente dans la colonne $i = r_{\max}$.

La programmation en FORTRAN 90 se fera en plusieurs temps :
- dans une **première version** (deux premières séances), on codera la définition du type dérivé nécessaire, l'initialisation et l'évolution du tas, l'affichage du nombre de grains dans chaque pile en fin de simulation, et l'affichage du tas de sable à l'écran durant son évolution.
- dans la **version finale** (deux dernières séances + libre service), on mettra en œuvre l'analyse des avalanches (taille, nombre...), et éventuellement des améliorations, proposées dans l'énoncé ou personnelles.

1 Première version

On demande de respecter dès le début de la programmation la structure des données ci-dessous.

1.1 Structure des données

Dans un module, créer le type dérivé `tas` avec les champs suivants :
- un entier `rayon` qui contiendra le nombre de piles
- un entier `hmax` qui contiendra la hauteur maximale du tas de sable
- un tableau dynamique `pile` d'entiers destiné à recevoir le nombre de grains par pile
- un tableau dynamique `grille` de caractères destiné à représenter le tas de sable en deux dimensions.

1.2 Contenu du programme

Dans le programme principal, créer un tas de sable en déclarant une variable de type dérivé `tas` (nommée par exemple `mon_tas`), sur laquelle on travaillera ensuite.
Programmer :

- la saisie contrôlée de son rayon maximal $r_{\max}$ et de sa hauteur maximale $h_{\max}$. On pourra utiliser une fonction `lecture_controlee (nmin, nmax)` qui renvoie une valeur saisie par l'utilisateur en s'assurant qu'elle est comprise entre les 2 valeurs passées en argument (pour $r_{\max}$ comme pour $h_{\max}$, on pourra adopter les valeurs 3 et 40 pour ces valeurs minimale et maximale) ;
- l'allocation et l'initialisation des champs dynamiques du tas `mon_tas` ;
- la boucle (qu'on appelle communément *boucle en temps*) permettant de coder le modèle présenté plus haut. On rappelle qu'un grain est introduit dans la pile 0 tous les n_t pas de temps (par exemple $n_t = 10$), et que la boucle s'arrête quand une pile atteint la hauteur $h_{\max}$;
- l'affichage du nombre de grains dans chaque pile du tas final.

Une fois que cet ensemble fonctionne, on pourra ajouter un sous-programme `affiche(un_tas)` qui affiche à l'écran le tas de sable `un_tas` passé en argument. Pour ce faire, on remplira le champ `grille` du tas `un_tas` en plaçant par exemple des caractères 'o' dans les cases occupées par des grains, puis on affichera la grille ainsi constituée. Ce sous-programme sera appelé après tout changement de hauteur d'une pile.

Lors de l'exécution, on peut, sous Linux, effacer l'écran grâce à l'instruction

```
CALL system('clear')
```

et ralentir l'exécution grâce à

```
CALL system('sleep 0.1')
```

qui ici force une pause de 0,1 s.

1.3 Version à rendre

Suivez bien la section *Consignes* pour rendre la première version du programme.

2 Version finale

En aucun cas les évolutions demandées ici ne seront rendues dans la première version.

2.1 Évolutions demandées

1. **Affichage optionnel**
   Quand le tas est très grand, ou quand on veut accélerer l'exécution du programme, on souhaite pouvoir désactiver l'affichage à l'écran. Insérer les lignes de code :
   - qui demandent à l'utilisateur s'il souhaite ou non l'affichage du tas de sable pendant l'exécution
   - qui désactive cet affichage si tel est le souhait de l'utilisateur.

2. **Écriture d'un fichier de résultats**
   On souhaite tracer à l'aide d'un logiciel graphique (gnuplot ou autre) le nombre de grains **final** en fonction du numéro de pile. Écrire les lignes de code qui permettent de générer un fichier de résultats `tas_final.res` comportant deux colonnes, la première étant le numéro de pile, la seconde le nombre de grains correspondant. Tracer ensuite la courbe et en tirer un fichier image.

3. **Utilisation d'un fichier de données**
   Dans le domaine de la recherche, un code de calcul est rarement interactif, car l'utilisateur n'est pas forcément présent devant la machine au moment où le code est lancé. De plus, la saisie au clavier des nombreux paramètres physiques ou numériques peut s'avérer fastidieuse et des erreurs sont facilement commises. On va donc ici remplacer l'entrée standard (au clavier) des différents paramètres par la lecture d'un fichier de données `param.dat` qui contiendra ces paramètres. Le fichier pourra par exemple se présenter ainsi :

   ```
   10              rmax
   12              hmax
   o               affichage du tas (o/n) ?
   tas_final.res   nom du fichier de resultats
   ```

   Le programme devra lire successivement les valeurs des paramètres (ici le premier élément de chaque ligne, c'est-à-dire 10, 12, 'o', ...) et les affecter aux variables correspondantes, sans contrôle des valeurs.

4. **Analyse physique**
   Au cours de l'évolution du tas de sable, le transfert de grains d'une pile à l'autre peut provoquer à son tour le transfert de grains entre les deux piles suivantes et ainsi de suite... Cette séquence de transferts de grains entre piles consécutives est appelée *avalanche*, et la *taille de l'avalanche* est le nombre de ces transferts consécutifs (ce n'est pas le nombre de grains impliqués, qui représenterait plutôt la masse de l'avalanche).
   On s'intéresse à la *distribution des tailles des avalanches*, c'est-à-dire au nombre d'avalanches $n_a(T)$ de taille T. Écrire les lignes de code permettant de déterminer la taille de chaque avalanche et de tenir le compte des avalanches en fonction de leur taille. Le programme devra écrire un fichier de résultats `distrib_taille.res`, permettant ensuite de tracer $n_a(T)$ en représentation linéaire puis log-log (plus adaptée). Pour le modèle BTW, cette loi est $n_a(T) = T^{\alpha}$ avec $\alpha \approx -0.90$ (quand $\alpha = -1$, on parle de *bruit en 1/f*). Obtient-on la même distribution ici ? On pourra étudier l'influence de $r_{\max}$ et/ou de $h_{\max}$ sur cette courbe.

2.2 Version à rendre

Respectez bien les consignes pour la rédaction du compte-rendu et le rendu du projet sur Moodle. Des points seront enlevés en cas de non-respect de ces consignes, en particulier pour les retards.

2.3 Améliorations possibles

Il s'agit de choisir une ou plusieurs améliorations parmi celles présentées ci-dessous. Elles ne sont que des suggestions. Toute autre amélioration sera la bienvenue. Ici importe surtout la qualité de la réalisation, plutôt que la quantité.

1. Réorganiser le code pour faire en sorte que le programme principal ne contienne que des appels à des procédures. Placer module(s) et programme principal dans des fichiers séparés.
2. Simuler un tas de sable entier (partie gauche et droite). On pourra, par exemple, appliquer le balayage de l'algorithme du centre vers l'extrémité droite ou gauche aléatoirement.
3. Tas ou demi-tas de sable non confiné : dans cette configuration, les grains ne s'accumulent pas en colonne $r_{\max}-1$ mais tombent du plateau une fois arrivés en colonne $r_{\max}$. Leur chute est-elle aussi régulière que leur introduction au sommet du tas ? Étudier l'évolution temporelle du nombre de grains du tas (expérimentalement, on pèse le plateau pour connaître le nombre de grains). Mettre en évidence les deux phases successives de l'évolution. Caractériser la seconde phase (valeur minimale, maximale, moyenne, écart-type). Ces caractéristiques varient-elles d'une simulation à l'autre ? Comment dépendent-elle de $r_{\max}$?
4. Grâce au type dérivé, on peut facilement créer un second tas de sable. Simuler deux tas de sable en cascade, en supposant que le second est alimenté en un point par la chute de grains d'un premier (demi-)tas non confiné (voir amélioration 3).
5. Introduire un obstacle ou plusieurs sur le plateau (de manière simple).
6. La distribution des masses des avalanches ressemble-t-elle à celle des tailles ?