Sorbonne Université
Master 2 Ingénierie mathématique
Programmation en C++

Examen du 24 octobre 2018
Durée : 1 heure 30

— **Tous** *les fichiers rendus devront contenir vos* **nom et prénoms**.
— *Le fait d'envoyer des fichiers qui ne se compilent pas correctement sera sanctionné.*
— *Les deux parties sont indépendantes entre elles.*

1 Schéma numérique dans $\mathbb{R}^2$

Dans cette partie, on va écrire une classe pour implémenter un schéma numérique de résolution de l'équation différentielle

$$x''(t) = -\frac{x(t)}{|x(t)|^3},$$

avec $x(t) \in \mathbb{R}^2$, qui correspond par exemple au mouvement d'une planète autour d'une étoile.

Les relations de récurrence suivantes, appelées *schéma de Størmer-Verlet* permettent de résoudre numériquement cette équation :

$$\begin{cases} v_{n+1} &= v_n - \delta_t \frac{x_n}{|x_n|^3}, \\ x_{n+1} &= x_n + \delta_t v_{n+1}, \end{cases} \tag{1}$$

où l'on part des conditions initiales $x_0 = x(0)$ et $v_0 = x'(0)$. Ici, δ_t représente le pas de discrétisation du schéma. On a alors l'approximation $x(n\delta_t) \simeq x_n$. Pour écrier facilement un programme calculant les suites $(x_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$, on va définir une classe permettant de manipuler des vecteurs de $\mathbb{R}^2$.

1. Écrire une classe `vecteur`, contenant deux membres privés `x` et `y` qui correspondront à l'abscisse et à l'ordonnée du vecteur considéré.
2. Écrire un constructeur pour cette classe, prenant en argument les deux valeurs à assigner à `x` et `y`.
3. Écrire des surcharges des opérateurs `+`, `-`, `/` et `*`, correspondant aux opérations attendues sur les vecteurs. L'opérateur `/` correspondra à la division par un scalaire, et l'opérateur `*` correspondra à la fois à la multiplication par un scalaire et au produit scalaire de deux vecteurs.
4. Surcharger l'opérateur `<<` pour permettre d'afficher les deux coordonnées du vecteur fourni.
5. Écrire un programme `planete.cpp` qui calcule les termes de la suite (1), et les écrit dans un fichier `planete.dat`. On pourra par exemple prendre $\delta_t = 0.1$, $x_0 = (1, 0)$ et $v_0 = (-0.5, 1)$, et faire 100 itérations (de sorte à arriver au tempts $100\delta_t = 10$). Une visualisation du résultat (par exemple sous gnuplot) ferait apparaître un ellipse.

2 Processus du restaurant chinois

On va considérer dans cette partie une suite de permutations aléatoires, connue sous le nom de *processus du restaurant chinois*, qui a par exemple des applications en biologie.

Pour définir ce processus on imagine un restaurant initialement vide, dans lequel des clients entrent un par un. On suppose que le restaurant dispose d'une infinité de tables rondes pouvant chacune accueillir autant de clients que nécessaire. Chaque nouveau client va se placer aléatoirement dans le restaurant, selon les règles suivantes : lorsque le $n+1$-ème client entre dans le restaurant,

— avec probabilité $\frac{t}{t+n}$ il s'assoit à une table vide ;

— avec probabilité $\frac{n}{t+n}$, il choisit une personne uniformément parmi les n personnes déjà présentes dans le restaurant et s'assoit à sa droite.

On peut représenter l'état du restaurant après l'entrée des n premiers clients par une permutation, écrite comme produit de cycles. Chaque cycle correspond à une table, et décrit l'ordre dans lequel les clients sont assis à cette table. Par exemple, l'écriture

$$(153)(26)(4) \tag{2}$$

correspond au cas où les premier, deuxième et quatrième clients se sont assis à une nouvelle table, alors que les clients numéro 3, 5 et 6 se sont respectivement assis à côté des clients 1, 1 et 2.

Le but de cette partie est d'écrire une classe permettant d'implémenter ce processus.

1. Écrire une classe `resto` contenant en membres privés trois objets `table`, `n` et `t` de types respectifs `std::vector<std::list<int> >`, `int` et `double`.

   Les variables `n` et `t` correspondent respectivement au nombres de clients entrés dans le restaurant jusque là et au paramètre t de la définition du processus. La variable `table`, quant à elle sera un vecteur de listes d'entier, telle que `table[i]` soit la liste des clients assis à la table numéro `i`. [1]
2. Écrire un constructeur prenant en argument un réel correspondant à la valeur de t (auquel on donnera la valeur par défaut 1), et créant un restaurant vide.
3. Écrire deux méthodes `n_tables` et `n_seuls` qui renvoient respectivement le nombres de tables occupées dans le restaurant, et le nombre de tables occupées par une seule personne.
4. Surcharger l'opérateur `<<` pour permettre d'afficher une variable de type `resto` sous la forme de (2).
5. Écrire une méthode `ajout` prenant en argument une référence sur un générateur pseudo-aléatoire, et ajoutant un client dans le restaurant, en respectant la règle énoncée en début de partie.
6. Écrire un premier programme `restaurant.cpp` qui affiche les états successifs du restaurant après les entrées successives des 10 premiers clients.
7. Écrire un deuxième programme nommé `convergence.cpp` qui écrit dans un fichier `convergence.dat` la valeur de la quantité $\tau_n/\log(n)$ (pour n variant jusqu'à 10000), où τ_n est le nombre de tables occupées après l'arrivée du $n$ème client. On peut en fait montrer que la quantité $\tau_n/\log(n)$ converge presque sûrement vers t, mais la convergence est très lente.

1. On rappelle que les classes `std::list` et `std::vector` disposent d'une méthode `size` renvoyant la taille de l'objet et d'une méthode `push_back` qui permet d'insérer un élément à la fin de l'objet. Par ailleurs, la classe `std::vector` dispose d'une méthode `resize` permettant de changer la taille d'un vecteur, et la classe `std::list` dispose d'une méthode `insert` permettant d'ajouter un élément à une position indiquée par un itérateur fourni en argument.