Programmation C++ sous linux

10 janvier 2012

Programmation C/C++ de base : Obligatoire pour tous

Programmation Orientée Objet - Les Classes : Facultatif

Introduction à la simulation d'événements aléatoires : Obligatoire pour ceux qui ont choisi un projet nécessitant la génération de nombres aléatoires

Résolution d'équations différentielles : Obligatoire pour ceux qui ont choisi un projet nécessitant la résolution d'un système d'équation différentielles.

Table des matières

1. Programmation C/C++ de base

Le but de se chapitre est de se familiariser avec les notions de base du langage :
Types et expression de base , types de base, Structures de contrôle, structures conditionnelles, structures itératives, Construction d'un makefile, Tableaux, Fonctions , passage par valeur, par adresse, par référence, pointeurs et références, Quelques commandes du préprocesseur.

1) Faire un petit programme de conversion Euro - Dollar. Vous pouvez utiliser l'éditeur gedit pour écrire le fichier source (pour l'exécuter, il suffit de taper gedit& dans un terminal)

```
g++ -o votreProgram votreFichierCpp.cc
```

permet de compiler votre programme. L'exécutable se prénomme votreProgram. Pour l'exécuter, taper ./votreProgram

2) Écrire un programme qui permet :

a) de saisir la note d'un étudiant. **On doit obliger l'utilisateur à saisir un réel entre 0 et 20.**

b) d'afficher la mention qui correspond à la note saisie :

Echec si note <10 , Passable si 10<=note<12, Assez Bien si 12<=note<14, Bien si 14<=note<16, Très Bien si note >= 16.

3) Écrire un programme qui permet de chercher, par dichotomie, le zéro de la fonction f(x) = $\mathbf{x+e^x-2}$.

Il faudra donc trouver la valeur de x_0 où $f(x_0) = 0$. On cherchera le zéro entre 2 valeurs *a* et *b* saisies par l'utilisateur. Soit *epsi* , saisie par l'utilisateur, la précision avec laquelle on cherche la valeur.

4) Écrire un programme qui permet de calculer la valeur de π en utilisant la série :

$$\Pi = 2\sum_{n=1}^{+\infty} S_n \text{ où } S_{n+1} = \frac{n}{2n+1} S_n \text{ et } S_1 = 1$$

5) Écrire un programme qui permet de chercher tous les nombres premiers inférieurs à 1000.

6) Écrire un programme qui comporte une fonction qui permet de chercher tous les nombres premiers entre 2 entiers a et b. Dans main demander à l'utilisateur les valeurs de a et b et calculer ensuite les nombres premiers.

7) Même exercice en définissant les prototypes dans un fichier d'entête .h et l'implémentation dans un fichier .cc.

8) Ecrire un programme qui permet de calculer numériquement une intégrale de type : $I=\int_a^b f(x)\,dx$

Cette intégrale sera calculée en utilisant l'équation suivante (méthode de Simpson) :

$I=\frac{h}{3}\left(f(a)+f(b)+\left(4\sum_{i=1,i\,impair}^{i=n} f(a+ih)\right)+\left(2\sum_{i=2,i\,pair}^{i=n-1} f(a+ih)\right)\right)$ où n est un entier impair >0 et $h=\frac{(b-a)}{n+1}$

9) Écrire un autre programme calculant la valeur moyenne d'un ensemble de nombres entrés au clavier. Utiliser un tableau pour stocker les valeurs en question et calculer la valeur moyenne dans une fonction.

10) Même exercice en définissant les prototypes dans un fichier d'entête .h et l'implémentation dans un fichier .cc.

11) Écrire un programme pour :

saisir des valeurs au clavier, les stocker dans un tableau

ne retenir que les valeurs positives et les copier dans un deuxième tableau qui sera alloué dynamiquement.

12) Faire un programme pour :

générer aléatoirement des valeurs réelles,

les écrire dans un fichier,

relire ces valeurs dans le fichier,

faire quelques opérations simples (valeur min, valeur max, moyenne)

réordonner les valeurs dans l'ordre croissant à l'intérieur du tableau.

2. Programmation Orientée Objet - Les Classes (Facultatif)

Les notions générales de programmation objet sont abordés rapidement :

Notion de classe

définition de type structuré,

attributs, méthodes,

encapsulation de données.

Construction d'objets

notion de construction,

constructeur de copie,

copie superficielle vs copie en profondeur

Écrire la classe Point représentant un point dans l'espace a 2 dimensions. les attributs de la classe sont les coordonnées (x, y) d'un point. la classe doit proposer les méthodes suivantes :

abscisse,

ordonnée,

distance à l'origine,

déplacement d'une quantité (dx, dy). Écrire la classe Vecteur permettant de manipuler un vecteur. Les attributs de la classe sont un tableau de valeurs de type double et sa taille. La classe devra contenir :

plusieurs constructeurs permettant de définir un vecteur,

l'opérateur d'affectation,

le destructeur de la classe,

une méthode dotprod pour calculer le produit scalaire,

la surcharge de l'opérateur << pour afficher le contenu du vecteur e Il faudra tester le fonctionnement de ces deux classes à l'aide d'un programme principal approprié.

3. Introduction à la simulation d'événements aléatoires

Si vous avez choisi un projet dans lequel vous aurez besoin d'un générateur de nombre aléatoire, vous devez faire cette partie du TP avant de commencer votre projet.

Dans ce chapitre nous étudions quelques points pratiques associés à la génération et à l'utilisation de ces nombres aléatoires.

3.1 Génération de nombres aléatoires

La génération de nombres pseudo-aléatoires est la clé de toutes les techniques de simulation. Un bon générateur doit pouvoir justifier d'un ensemble de qualités avant de pouvoir être utilisé intensivement. Les tests statistiques standards de moyenne, écart-type, etc. sont évidemment nécessaires, mais ils sont notoirement insuffisants pour garantir des résultats fiables.

De très nombreux tests destinés à valider un générateur ont été proposés, tests dont la description est hors de propos ici. Il faut en particulier connaître le cycle du générateur ou du moins savoir si ce générateur possède un cycle qui est supérieur au nombre de tirages envisagés. Il existe de bonnes sources, citons en particulier : Numerical Recipes, les bibliothèques CERN, NAG, GNU Scientific Library, etc.

Nous allons utiliser ici la fonction rand() déjà implémentée dans cstdlib ainsi que 2 autres fournies en TP basées sur les algorithmes 'Mersenne Twister' et 'multiply-with-carry'.

3.2 Distribution uniforme

En règle générale, les nombres aléatoires distribuées uniformément sont générés à partir de congruences :

$$u_{j+1} = a\, u_j + c \text{ (modulo m)}$$

m est le module, et a et c sont le multiplicateur et l'incrément respectivement. La difficulté réside évidemment dans le choix de ces paramètres. u_0 est la graine (*seed* en anglais : un nombre employé pour produire une suite pseudo-aléatoire habituellement plus longue). La fonction rand() est la fonction 'Standard minimal' qui correspond à «a= 16807, c=0 et m = 2^{31}-1».

La méthode « multiply-with-carry» inventée by George Marsaglia est la suivante :

S = 2111111111*X[n-4] + 1492*X[n-3] + 1776*X[n-2] + 5115*X[n-1] + C

X[n] = S modulo 2^32

C = floor(S / 2^32)

C'est une méthode avec très grande période (variant de 2^60 à $2^{2000000}$) en plus la fait qu'elle est rapide par rapport à d'autres méthodes de même qualité comme la méthode '**Mersenne twister'** developpée in 1997 by Makoto Matsumoto et Takuji Nishimura. *Le code source de cette méthode sera fourni en TD (Voir spiral). On vous demandera de l'utiliser pour la comparer à la méthode Standard minimal utilsiée*

par la fonction rand().

Pour analyser statistiquement une distribution de nombres aléatoires, on peut en faire un histogramme en calculant aussi la valeur moyenne et l'écart quadratique moyen.

1) Générer une suite de N (N de l'ordre de 1000 à 100000) nombres réels pseudo-aléatoires (on testera les 3 méthodes citées ci-dessus) entre 0 et 1 .
2) Calculer la valeur moyenne et l'écart type en fonction N.
3) Comment évoluent ces résultats avec N? Visualiser le résultat avec gnuplot.
4) Calculer et tracer la distribution sous forme d'un histogramme. Pour cela :
On décompose notre intervalle [0,1[en M petits intervalles. On déclare un tableau de M éléments que l'on initialise à 0. On tire un nombre x entre 0 et 1. On ajoute 1 à l'élément x*N du tableau. On fera 10000 tirages (au moins). On stocke le résultat dans un fichier et on visualise le résultat avec gnuplot.

3.3 Distribution non uniforme (Facultatif)

Plutôt qu'une distribution uniforme, on peut être amené à générer des nombres suivant une distribution normale, poissonnienne, etc.

3.3.2 Génération d'une distribution normale

La loi normale, notée $N(g,m,\sigma)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(g-m)^2}{2\sigma^2}}$, est caractérisée par sa moyenne m et son écart-type σ (ou sa variance σ^2). La loi N(g,0, 1) est appelée *loi normale réduite.*
Pour générer des nombres aléatoires qui suivant une distribution normale, on utilisera la méthode de Box-Muller :
Si x_1 et x_2 sont deux nombres aléatoires uniformes sur]0,1[, les nombres y_1 et y_2 définis par :

$$y_1=\sqrt{-2\ln x_1}\cos 2\pi x_2$$
$$y_2=\sqrt{-2\ln x_1}\sin 2\pi x_2$$

suivent la loi normale réduite.
On en déduit que les nombres $g_1=m+\sigma y_1$ et $g_2=m+\sigma y_2$ suivent la loi normale N(m, s)

$$N(g,m,\sigma)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(g-m)^2}{2\sigma^2}} .$$

1) Générer une suite de N (N de l'ordre de 1000 à 100000) nombre avec une moyenne m et un écart-type σ ; valeurs maximum et minimum du tirage.
2) Vérifier si votre échantillon est tel que environ 68% de la population se situe dans m± σ . Quel est le pourcentage dans les intervalles m ± 2σ et m ± 3σ ?
3) Comment évoluent ces résultats avec N?

3.3.3 Distribution exponentielle

Pour une distribution $\frac{1}{\lambda}e^{-\lambda x}$ exponentielle de moyenne λ on procède comme suit :

1) On tire u uniforme sur [0, 1[
2) le nombre cherché est : $-\log(u)\lambda$

Faire le même exercice que précédemment avec la distribution exponentielle.

3.4 Méthode de Monte Carlo

On appelle **méthode de Monte-Carlo** toute méthode visant à calculer une valeur numérique, et utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués à Monte-Carlo, a été inventé en 1947 par Nicholas Metropolis.

Cette méthode permet l'estimation des moyennes de grandeurs physiques données par la formulation de Gibbs de la mécanique statistique sous la forme d'intégrales multidimensionnelles.
La technique de MC est en effet particulièrement adaptée au calcul des intégrales de dimension supérieure à dix. Les premières simulations furent réalisées dans l'ensemble canonique (N, V et T constants), puis la technique fut étendue aux autres ensembles statistiques. On génère une séquence aléatoire d'états accessibles dans l'espace des configurations du système. On échantillonne en privilégiant les régions où le facteur de Boltzmann (exp(-U/k_BT)), c'est-à-dire la densité de probabilité de l'ensemble canonique dans cet espace, est le plus élevé (algorithme de Metropolis). La probabilité d'une configuration particulière d'énergie potentielle Ui est alors proportionnelle à exp(-Ui/k_BT) , autrement dit l'acceptation d'une configuration de la chaîne de Markov est pondérée par une fréquence *proportionnelle au facteur de Boltzmann*. Une propriété d'équilibre est alors obtenue comme une moyenne simple sur les configurations acceptées. Cette exploration de l'espace des configurations, en suivant l'algorithme de Metropolis, constitue le premier cas d'*échantillonnage suivant l'importance* en mécanique statistique. Elle est encore

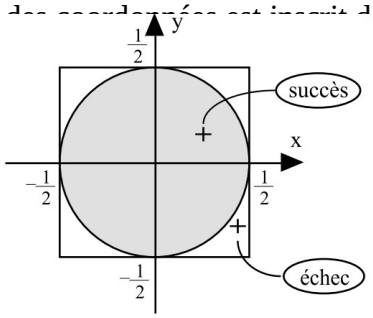
largement utilisée de nos jours parce qu'elle représente un moyen simple et relativement efficace d'obtenir des moyennes de grandeurs physiques dans un ensemble statistique. La méthode de MC est en général limitée au calcul des *propriétés statiques* puisque seule la partie configurationnelle de l'espace des phases est explorée et que le temps n'est pas une variable explicite. Les propriétés dynamiques sont inaccessibles et devront être obtenues par une autre technique.

3.4.1 Calcul de π

On va introduire le principe de la méthode à l'aide d'une estimation de la valeur du nombre π par la technique de MC de type *tirage par noir ou blanc*.

Cette estimation peut être réalisée en déterminant l'aire, , d'un cercle. Le cercle de rayon 1/2 centré à l'origine du repère des coordonnées est inscrit dans un carré de surface unité.

$$\pi = 4\,\frac{Aire\,du\,cercle}{Aire\,du\,carrée}$$



Comment mesurer le rapport des aires ?

On génère à l'ordinateur deux nombres aléatoires compris entre – 1/2 et 1/2. On les utilise pour obtenir les coordonnées d'un point (x,y) situé dans le carré. Si la distance entre ce point et l'origine du repère est inférieure ou égale au rayon du cercle (1/2), le point se trouve à l'intérieur du cercle et l'on comptabilise un succès, autrement c'est un échec. Dans cette expérience, seul le hasard intervient. Le nombre de succès, n_s, est proportionnel à l'aire du cercle, tandis que le nombre d'échecs, n_E , est proportionnel à l'aire de la région du carré non recouverte par le cercle. π est alors donné par la relation :

$$\pi = \lim_{n\to\infty} 4\,\frac{n_s}{n_s + n_E}$$

Écrire le programme qui fait le calcul pour n =100, 10^4, 10^6, 10^8

3.4.2 Calcul d'intégrale (facultatif)

L'évaluation des intégrales multiples est un domaine où la méthode de Monte Carlo rend des services inestimables. Cependant, nous nous limiterons ici au calcul d'une intégrale simple.

Supposons que nous disposions de n points, distribués de manière aléatoire dans un volume V (supposé de dimension d quelconque). Le théorème de base de MC consiste à estimer que l'intégrale d'une fonction f sur le volume V est approchée par la relation,

$\int f\,dV \approx V(< f > \pm \sigma_f)$ où $< f >= \frac{1}{n}\sum_{i=0}^{N} f(x_i)$ est la moyenne f sur les n points x_i choisis aléatoirement et $\sigma_f = \sqrt{< f^2 > - < f >^2}$.

Le terme d'erreur, proportionnel à la variance de f, est une estimation, à une erreur standard, et n'est absolument pas une borne rigoureuse. A partir de cette formulation, on réalise que la précision augmente seulement comme la racine carrée du nombre n de tirages. La convergence est donc, à priori, très mauvaise dans le cas à une dimension.

Comme exemple de l'évaluation d'un intégrale, par MC, nous prendrons l'intégrale très simple :

$$\int_0^1 \frac{1}{1+x^2}\,dx = \frac{\pi}{4}$$

1) Écrire le programme qui permet de calculer cette intégrale.
2) Étudiez la convergence du calcul en fonction du nombre de tirages N (prendre N = 2, 4, 8, 16, . . .)
3) Vérifier que cette précision augmente comme $\sqrt{N}$
4) Comment peut-on imaginer d'améliorer la convergence de la méthode ?

4. Résolution d'équations différentielles

Si vous avez choisi un projet dans lequel vous allez résoudre une équation différentielle, vous devez faire cette partie du TP avant de commencer votre projet.

La connaissance des lois physiques décrivant les propriétés et le comportement d'un système permet naturellement d'aborder la problématique de la prédiction de l'évolution de ce système dans le temps, Cette évolution du système est naturellement représentée par une équation maîtresse qui prend généralement la forme d'une équation (ou plusieurs) différentielle. Il est parfois possible de résoudre analytiquement cette équation mais certaines équations ne peuvent être résolues que numériquement.

Les **méthodes de Runge-Kutta** sont des méthodes d'analyse numérique d'approximation de solutions d'équations différentielles. Elles ont été nommées ainsi en l'honneur des mathématiciens Carl Runge et

Martin Wilhelm Kutta lesquels élaborèrent la méthode en 1901.

Ces méthodes reposent sur le principe de l'itération, c'est-à-dire qu'une première estimation de la solution est utilisée pour calculer une seconde estimation, plus précise, et ainsi de suite.

La méthode de Runge-Kutta d'ordre 1 (RK1)

Cette méthode est équivalente à la méthode d'Euler, une méthode simple de résolution d'équations différentielles du 1er degré. Considérons le problème suivant:

$$y' = f(t, y), y(t_0) = y_0$$

La méthode RK1 est donnée par l'équation :

$y_{n+1} = y_n + h\, f(t_n, y_n)$ où h est le pas de l'itération.

La méthode de Runge-Kutta classique d'ordre quatre (RK4)

C'est un cas particulier d'usage très fréquent, dénoté RK4.

La méthode RK4 est donnée par l'équation :

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5)$$

où $k_1 = f(t_n, y_n)$; $k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$; $k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$; $k_4 = f(t_n + h, y_n + hk_3)$

L'idée est que la valeur suivante (y_{n+1}) est approchée par la somme de la valeur actuelle (y_n) et du produit de la taille de l'intervalle (h) par la pente estimée. La pente est obtenue par une moyenne pondérée de pentes :

- k_1 est la pente au début de l'intervalle ;
- k_2 est la pente au milieu de l'intervalle, en utilisant la pente k_1 pour calculer la valeur de y au point $t_n + h/2$ par le biais de la méthode d'Euler ;
- k_3 est de nouveau la pente au milieu de l'intervalle, mais obtenue cette fois en utilisant la pente k_2 pour calculer y;
- k_4 est la pente à la fin de l'intervalle, avec la valeur de y calculée en utilisant k_3.

Dans la moyenne des quatre pentes, un poids plus grand est donné aux pentes au point milieu.

$$pente = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

La méthode RK4 est une méthode d'ordre 4, ce qui signifie que l'erreur commise à chaque étape est de l'ordre de h^5, alors que l'erreur totale accumulée est de l'ordre de h^4.

Pour un système couplé du premier :

$$y' = f(t, y, z)$$

$$z' = g(t, y, z)$$

Le schéma itératif (RK4) se présente ainsi :

$$y_{n+1}=y_n+\frac{h}{6}(k_1+2k_2+2k_3+k_4)+O(h^5)$$

$$z_{n+1}=z_n+\frac{h}{6}(l_1+2l_2+2l_3+l_4)+O(h^5)$$

les coefficients k_i et l_i sont obtenus par :

$$k_1=f(t_n,y_n,z_n)\ ;\ l_1=g(t_n,y_n,z_n)$$

$$k_2=f(t_n+\frac{h}{2},y_n+\frac{h}{2}k_{1,}z_n+\frac{h}{2}l_1)\ ;\ l_2=g(t_n+\frac{h}{2},y_n+\frac{h}{2}k_{1,}z_n+\frac{h}{2}l_1)$$

$$k_3=f(t_n+\frac{h}{2},y_n+\frac{h}{2}k_{2,}z_n+\frac{h}{2}l_2)\ ;\ l_3=g(t_n+\frac{h}{2},y_n+\frac{h}{2}k_{2,}z_n+\frac{h}{2}l_2)$$

$$k_4=f(t_n+h,y_n+hk_{3,}z_n+hl_3)\ ;\ l_4=g(t_n+h,y_n+hk_{3,}z_n+hl_3)$$

Une équation différentielle du second ordre peut être décomposée en 2 équations différentielles couplées su premier ordre.

Prenons par exemple l'équation d'un oscillateur amorti entretenu :

$$\ddot{x}(t)+2\lambda\dot{x}(t)+\omega_0^2x(t)=e(t)$$

où:

x(t) est la variable de position du système : paramètre dynamique (dépendant du temps t),

$\lambda et\,\omega_0$ sont des paramètres statiques du système : ils ne dépendent pas explicitement du temps ou des paramètres dynamiques,

On définit la nouvelle variable dynamique $z(t)=\dot{x}(t)$, on obtient alors deux équations du premier ordre couplées l'une à l'autre :

$$\dot{x}(t)=z(t)$$

$$\dot{z}(t)=e(t)-2\lambda\dot{x}(t)-\omega_0^2x(t)$$

On se ramène ici à la résolution parallèle d'une équation différentielle du premier ordre en x et d'une équation différentielle du premier ordre en z (z coïncide ici avec la vitesse de l'oscillateur).

Avec les conditions initiales : position x(t = 0) = x0 et vitesse z(t = 0) = 0

et $\lambda<\omega_0$ (frottements faibles), et e(t) =0, la solution analytique est connue :

$$x(t)=x_0e^{-\lambda t}(\cos\omega' t+B\sin\omega' t)$$

$$z(t)=-x_0e^{-\lambda t}(\frac{\lambda^2+\omega'^2}{\omega'})\sin\omega' t \qquad \text{Avec} \quad \omega'=\sqrt{\omega_0^2-\lambda^2} \quad \text{et} \quad B=\lambda/\omega'$$

Résoudre numériquement cette équation par la méthode d'Euler et la méthode RK4.

Comparer vos résultats à la solution analytique.