CONTS

L'apport des logiciels dans un cours d'info.

Hugues Saulnier

Utiliser le plus possible des méthodes ou des résultats physiques et mathématiques. Notre groupe d'info au SEG s'est fait une spécialité de ces applications scientifiques. On les retrouve dans le cadre de travaux pratiques, les périodes de cours étant naturellement utilisées par la théorie propre à l'info.

Certaines méthodes ou résultats mathématiques sont souvent accessoires aux cours de maths normaux et nos étudiantes les connaissent déjà : la différentielle, les dérivées numériques du premier et deuxième ordre, constructions de polynômes, méthodes de Simpson ou de Runge-Kutta, on peut y ajouter l'obtention d'évènements suivant une distribution donnée et l'estimé de probabilités assez difficiles avec simulation. Je n'en traiterai pas ici, il existe d'ailleurs une vaste littérature informatique sur ces sujets -- code compris --

Parfois les mathématiques requises par la problématique donnée sont d'un niveau plus avancé que celui des cours normaux. Deux variations apparaissent :

- Les procédés mathématiques devront être implémentés -- en C par exemple--. J'en donne un exemple complet avec l'utilisation du théorème chinois du reste agréablement détourné pour la cryptographie. (annexe #1 pour les maths et #2 pour la présentation)

- Les procédés mathématiques sont trop complexes pour être implémentés. On utilise alors les fonctions d'un logiciel en pré-calcul pour dresser des tables de représentation d'objets mathématiques qui eux seront directement utilisés par l'application demandée. J'en donne l'exemple avec l'utilisation de la fonction Primitive de Maple dans la construction des LFSR générateurs pseudo-aléatoires. (annexe #3)

Certains sceptiques diront je suis d'accord……

/*----------annexe #1----------*/

Le théorème chinois du reste.

/*----------*/

/*----------*/

Soient $\{n_1, n_2, ..., n_k\}$ des entiers naturels relativement premiers.

Donc tels que :

$PGCD(n_i, n_j) = 1$ avec $i \neq j$

Avec $\prod_i n_i = M$

Alors tout naturel E<M s'écrit de façon unique
comme un vecteur V d'entiers de dimension k

$$V = [E \bmod n_1, E \bmod n_2, ..., E \bmod n_k]$$

/*----------*/

Qui dit représentation unique dit bijection
et inversion possible.

On devra pour inverser construire le vecteur W
dépendant uniquement de $\{n_1, n_2, ..., n_k\}$:

Soient $\{M_1, M_2, ..., M_k\}$ définis comme :

$$M_i = \prod_{j \neq i} n_j = \frac{M}{n_i}$$

Et soient $\{in_1, in_2, ..., in_k\}$ définis comme les
inverses modulaires des M_i modulo n_i

$M_i * in_i \equiv 1 \bmod n_i$

Alors

$W = [(M_1 * in_1), (M_2 * in_2), ..., (M_k * in_k)]$

Avec le vecteur W connu, un produit scalaire
suffira pour inverser

$(V \bullet W) \bmod M = E$

/*----------*/

/*----------*/

/*----------annexe #2----------*/
Le théorème chinois en crypto.

Utiliser le théorème chinois en cryptographie non professionnelle est intéressant : de belles maths couplées à l'ampleur certaine de la variabilité possible du procédé en font un petit bijou à utiliser en composition avec des procédés plus standard comme un Xor ou une permutation.

1. Pour crypter : un petit groupe $n_1, n_2, ..., n_k$ d'entiers relativement premiers permettent d'obtenir une transformation radicale d'un flot – stream ou canal -- source quelconque traité séquentiellement par blocs de P1 bits. Chaque bloc représente un entier E. Chaque résultat $\left(E \bmod \left(n_i\right)\right)$ peut naturellement s'écrire sur le nombre de bits nécessaires à écrire $\left(n_i\right)$ lui-même. Les k blocs de bits obtenus seront concaténés dans un train de bits de longueur P2 $\geqslant$ P1 et forment une représentation unique de E. Ce bloc de P2 bits est aussitôt inséré dans le flot cible.

2. Pour décrypter : une fois trouvés les k inverses modulaires associés au groupe $n_1, n_2, ..., n_k$ -- annexe #1--, le flot codé est traité par bloc de P2 bits. De chaque bloc, on extrait les k restes pour effectuer ensuite les opérations mathématiques qui redonneront E. L'entier original sur P1 bits sera alors inséré dans le nouveau flot cible.

La clé d'une implémentation réussie par mes étudiants, c'est de leur montrer lors de la présentation du travail demandé tous les outils de la TI ou de Maple qui serviraient à obtenir les résultats nécessaires à la main et dont ils devront coder des versions analogues en C-C++ . Les étudiants obtiennent donc cryptage et décryptage réels lors de cette présentation. L'ensemble des fonctions utilisées leur dicte même le squelette de la librairie demandée. Ils s'y réfèreront constamment pour valider les résultats de leurs propres fonctions.
Les trois grandes phases d'une implémentation procédurale correcte : modularité, factorisation et validation y sont présentes.
Cette forme de présentation me permet d'aborder avec sérénité un problème informatique et mathématique intéressant – Dieu! Que sans cela, c'est l'enfer--

Considérerons maintenant l'ensemble 112, 45, 121. comme exemple petit mais privilégié dans le reste de ce texte et voyons les relations entre fonctions offertes par les logiciels et fonctions à coder :

1. Nos entiers sont-ils relativement premiers? On teste avec Gcd(…..) . Son équivalent C de l'algorithme d'Euclide est simple et courant.

2. Un inverse modulaire – s'il existe bien sur, ce qui est toujours notre cas -- s'obtient directement avec le Msolve(…) de Maple

   - msolve((45*121)* in_1 = 1, 112); $\{in_1 = 13\}$
   - msolve((112*121)* in_2 = 1, 45); $\{in_2 = 13\}$
   - msolve((45*112)* in_3 = 1, 121); $\{in_3 = 72\}$

   Les étudiantes obtiennent aussitôt le vecteur de décodage qui sera ici
   ========== [70785 , 176176 , 362880]==========

   La forme informatique en C de l'obtention de l'inverse module X de A modulo B se base sur un retournement de l'algorithme traditionnel d'Euclide pour obtenir explicitement la combinaison linéaire AX+Bα =1 -- avec une difficulté informatique certaine --.

3. Les k résultats $\left(E \bmod \left(n_i\right)\right)$ peuvent chacun s'écrire sur le nombre de bits nécessaires à écrire $\left(n_i\right)$, une fois concaténés, ils formeront normalement un train de bits de longueur légèrement supérieure à P1.
   - Avec {112, 45, 121} dont le produit vaut 609840 ou
     1001010011100011000 sur 20 bits obtenu directement avec l'opérateur
     ▸ bin de la TI, nous pourrons donc obtenir une représentation unique pour tous les entiers blocs de 19 bits --Max 524287--.
   - Le modulo 112 s'inscrira sur 7 bits, le modulo 45 sur 6 bits et le modulo 121 sur 7 bits. Pour un total minimum de 20 bits, légère augmentation parfaitement assumée.

   Ainsi prenons au hasard un bloc de 19 bits du flot source : 0111000110011110000 (232688 en décimal) donnera
   - 232688 mod 112 = 64 pour 1000000
   - 232688 mod 45 = 38 pour 100110
   - 232688 mod 121 = 5 pour 0000101
   - pour finir avec le bloc de 20 bits 10000001001100000101

Avec le vecteur de décodage [70785 ,176176 , 362880], on termine notre exemple
(70785 * 64+ 176176 * 38 + 5 * 362880) mod 609840 = 232688

/*----------annexe #3----------*/

Les LFSR forment une classe de générateurs pseudo aléatoires extrêmement utilisée.

- Ils prennent un espace Ram réduit, un simple tableau d'octets d'éléments dans Z_2
- Le changement d'état du tableau émet un bit pseudo aléatoire et se résume à un XOR de quelques positions du tableau et à l'incrémentation d'un indice -- séquence relativement simple algorithmiquement parlant --.
- Ils ne sont pas les plus rapides mais très souvent leur temps d'exécution reste marginal face à l'ensemble du traitement requis.
- **Bien choisis**, ils ont des périodes incroyables…..très facilement supérieures à 10^{60} ce qui se rapproche, je crois, du nombre d'atomes dans l'Univers…..Et en plus, ceux-là possèdent de bonnes caractéristiques statistiques.
- Bien qu'ils ne doivent jamais être utilisés comme outil unique de cryptographie -- un simple XOR de l'output avec le message serait balayé en cryptanalyse -- , l'utilisation d'un « bon » LFSR dans un protocole plus complexe impliquant la composition d'outils distincts est tout à fait envisageable.

Donc le problème se résume à séparer les bons des mauvais……
Ici c'est l'Algèbre qui s'en chargera….

Un LFSR est complètement déterminé par son état initial et les positions dans le tableau prises en compte lors du XOR.
Mais sa qualité ne dépend que des positions choisies. On construit le polynôme dit caractéristique d'un LFSR :

- La taille du tableau donne le degré du polynôme et son terme constant est 1.
- Les autres positions du tableau qui servent au XOR nous donnent les coefficients non nuls du polynôme.

Voici un théorème bien connu en cryptographie : La période d'une suite produite par un LFSR de taille n est $2^n - 1$ si et seulement si son polynôme caractéristique est primitif modulo 2.

Curieusement les listes qu'on retrouve dans la littérature spécialisée nous donnent toutes des polynômes à degré ridiculement bas!

La fonction Primitive de Maple teste si un polynôme donné est primitif mod p.

Donc lorsque l'ensemble de mes étudiants partent à la pêche aux primitifs avec une simple commande qui permettra d'en obtenir à 4 positions non triviales

```
for i from 200 to 255 do if
Primitive(x^i+x^181+x^101+x^71+1) mod 2 then print(i); fi ;
od; print(i) ;
                225       232
```

ou en construisant une fonction à l'interface plus agréable qui permettra d'en obtenir à 2 positions non triviales…..

```
primitif := proc( a,b )
local i,k: for i from a to b do k:=1+(rand()mod(i-2)):if
Primitive(x^i+x^k+1) mod 2 then print(k,i); fi ; od;
end proc;
```

avec comme résultat

```
primitif(200,300);
                        34, 231
                        53, 270
```

Rapidement la liste s'allonge…

En conclusion….

Le niveau mathématique est élevé, c'est celui des extensions de corps finis. J'en glisse un mot ici et là en exposant le procédé informatique. Je peux même me permettre de montrer l'énumération des éléments de $\mathbb{Z}_2[x]/x^3+x+1$ avec les puissances successives de x, et sans aucune formalité. Mais le mal est fait, bon nombre se rappelleront que des maths avancées leur sont utiles et accessibles dans nos logiciels.
Du plaisir pour tous, quasi!

Les leçons de l'info.

Les étudiantes de L'ETS suivent un cours d'intro à la programmation scientifique. Peu importe la langue qu'on y pratique – souvent le C --, on leur enseigne les mêmes principes de base qui peuvent se résumer par :

1. Factoriser au maximum : toute partie de la tâche qui se décrit en quelque mots représente une entité informatique indépendante…une fonction.

2. Utiliser des identificateurs représentatifs. Bien commenter, et avant tout les en-tête de fonctions : qui en sera le client, ce qu'elle offre, ce qui entre -- les paramètres formels -- et ce qui en sort.

3. **Ne jamais mêler traitement et interface**.

4. **Ne jamais utiliser de variables globales**.

J'ai le plaisir d'enseigner les maths, mais de façon très épisodique, le gros de mon enseignement se fait en info dure. Reste que j'ai l'opportunité chaque session de faire de l'encadrement en maths.
Surpris de ne pas voir les étudiants se servir plus des possibilités de programmation de la TI, je me suis rendu compte qu'ils connaissent – entre autres -- très mal les fonctions de base matricielles de leur machine, or elles sont analogues à celles qu'ils possèdent en C ou qu'on leur demande de créer en classe d'info. Comme les besoins réels en programmation mathématique font un usage immodéré de matrices! Oups..

En donnant un cours d'équations différentielles, je ne fais aucune programmation de la TI -- pas de temps à perdre --. Mais en présentant certaines techniques, je me permets d'indiquer mon goût pour leur programmation. C'est alors à eux de synthétiser le procédé dans ses constructions élémentaires -- la factorisation apparaîtra --.

Je ne leur dicte qu'une chose : Pas de procédure…que des fonctions, oubliez l'interface, concentrez-vous sur les résultats.

J'ai ajouté un simple exemple de programmation de la méthode de variations des paramètres en EDO qui suit les principes de bonne programmation.

Exemple de module implémentant la méthode de variation
des paramètres en EDO.

```
@============== VariPara ==================
@ La fonction en interface qui résout le système linéaire
@ Reçoit :
@ v : vecteur solution de l'EDO sans force extérieure
@ t : la variable indépendante de l'EDO
@ f : la force extérieure
@ Retourne : le vecteur ligne de la dérivée
@     des coefficients nécessaires.
@================================
VariPara ( v, t, f )
Func
     Local mat, vec

     @ on obtient la matrice des dérivées de v
     Vari_mat(vec,t)-> mat

     @ construction du vecteur solution du système
     Vari_vec(f,colDim(v))->vec

     @ résolution et la solution est transposée

     return simult(mat,vec)ᵀ
EndFunc

@============== Vari_mat ==================
@ utilitaire de VariPara
@ On reçoit :
@ v : vecteur solution de l'EDO sans oscillations forcées
@ t : la variable indépendante de l'EDO
@ On retourne a matrice carrée des dérivées successives de v
@================================
Vari_mat( v , t )
Func
     Local mat, taille, i
     @ on obtient l'ordre de l'EDO
     colDim(v)-> taille

     @ la matrice nulle sera de bonnes dimensions
     newMat(taille,taille) -> mat

     @ on remplit ses lignes des dérivées successives
     For i,1,taille
          v -> mat[i]
          d(v,t) -> v
     EndFor

     return mat
EndFunc
```

```
@============== Vari_vec ==================
@ utilitaire de VariPara
@ Reçoit :
@ f : la force extérieure du système
@ taille : la taille du vecteur à construire
@ Retourne le vecteur colonne avec comme
@ On retourne le vecteur de v
@================================
Vari_vec( f , taille )
Func
     Local vec

     @ le vecteur colonne nul sera de bonne dimension
     newMat(taille,1) -> vec

     @ et son dernier élément sera la force extérieure
     f -> vec[taille,1]

     return vec

EndFunc
```