PROG
TP#2: Convolution discrète, Automates cellulaires

2020-W38

Exercice 1 : Convolution discrète

La *convolution discrète* de deux fonctions a et b à indices dans $\mathbb{Z}$ est donnée par la formule $(a * b)[i] = \sum_{j=-\infty}^{\infty} a[i-j]\, b[j]$.

Q.1 : Écrivez une fonction

```
void conv(double a[], double b[], double c[], int n);
```

qui calcule la convolution discrète des fonctions (ou signaux) a et b représentées par `a` et `b`, et stocke le résultat dans `c`, toutes ces variables étant des tableaux de `n` `double`. On supposera ainsi que le support de a et b est inclus dans $[\![0, n-1]\!]$, et on ne demande de calculer le résultat que sur cet intervalle.

Q.2 : Calculez sur 32 points la convolution de deux signaux rectangulaires valant 1 sur $[\![0, 15]\!]$ et 0 partout ailleurs. Expliquez le résultat.

Q.3 : Écrivez une nouvelle fonction

```
void conv2(double a[], double b[], double c[], int an, int bn);
```

qui calcule sur an points la convolution de deux fonctions dont les supports, potentiellement distincts, sont $[\![0, an-1]\!]$ et $[\![0, bn-1]\!]$.

Q.4 : Utilisez `conv2` pour implémenter un filtrage par moyenne glissante sur 5 points pour les signaux suivants, avec un résultat stocké sur 32 points :

- un signal triangulaire isocèle de support $[\![8, 23]\!]$ (valant par ex. 8 en son maximum) ;
- un signal triangulaire rectangle de support $[\![8, 15]\!]$ (valant par ex. 8 en son maxium) ;
- un signal rectangulaire de support $[\![8, 15]\!]$ (par ex. d'amplitude 1) ;
- une impulsion en 16 (par ex. d'amplitude 1) ;
- un peigne d'impulsions de période 8 (par ex. d'amplitude 1) ;
- un peigne d'impulsions de période 4 (par ex. d'amplitude 1) ;

Expliquez les résultats.

Q.5 : Écrivez une nouvelle fonction

```
void conv3(double a[], double b[], double c[], int an, int bn, int
↪  bnegoffset);
```

qui calcule sur an points la convolution de deux fonctions dont les supports $[\![0, an-1]\!]$ et $[\![-bnegoffset, bn-1-bnegoffset]\!]$ ne sont plus nécessairement alignés en zéro. Refaites les calculs de la question précédente en appliquant un décalage de -2 pour le signal de filtrage. Comparez les résultats.

Q.6 : Refaites les calculs de la question précédente avec le signal de filtrage suivant $[0.2, 0.4, 0.8, 0.4, 0.2]$.

Exercice 2 : Automates cellulaires à une dimension

On considère dans cet exercice des automates cellulaires à une dimension dont les cellules ne peuvent prendre que deux valeurs. L'état d'un automate comportant n cellules est stocké dans un tableau de n entiers valant 0 ou 1. Le but de cet exercice est d'implémenter quelques exemples d'automates dont les règles de transition ne prennent en compte que les voisines immédiates d'une cellule : la valeur au temps $i+1$ de chaque cellule est donnée par un prédicat appliqué à sa valeur et celle de ses voisines au temps i, qu'on représente par le triplet *(voisine de gauche, cellule centrale, voisine de droite)*. Par exemple, l'automate défini par la *règle 30* a le comportement suivant :

- si le triplet au temps $i \in \{(1,1,1), (1,1,0), (1,0,1), (0,0,0)\}$, la cellule centrale vaut 0 au temps $i+1$;
- si le triplet au temps $i \in \{(1,0,0), (0,1,1), (0,1,0), (0,0,1)\}$, la cellule centrale vaut 1 au temps $i+1$;

Q.1 : Implémentez une fonction

```
void print_state(int a[], int an);
```

qui affiche l'état courant d'un automate représenté par le tableau `a`. On affichera le caractère `' '` pour une cellule valant 0, et le caractère `'*'` pour une cellule valant 1 (on rappelle que le modificateur de format de `printf` permettant d'afficher un unique caractère est `%c`). Pour plus de lisibilité, vous ne devrez aussi faire qu'un unique retour à la ligne par état.

Q.2 : Implémentez une fonction

```
void rule30(int a[], int an, int nsteps);
```

qui calcule et affiche `nsteps` états successifs de l'automate fourni en argument en utilisant la règle définie plus haut. On considérera par défaut que les voisines non définies des cellules en bord de tableau valent 0.

Écrivez une fonction de test pour afficher 100 états d'un automate de taille égale à la largeur d'affichage de votre terminal, dont l'état initial vaut 0 partout sauf 1 en son milieu.

Q.3 : On peut remarquer que pour le type d'automate considéré, la règle d'évolution étant donnée par une fonction Booléenne à trois variables, il n'y a que $2^{2^3} = 256$ automates différents possibles. On adopte la terminologie suivante : l'automate défini par le prédicat :

- $(0,0,0) \mapsto x_0$
- $(0,0,1) \mapsto x_1$
- $(0,1,0) \mapsto x_2$
- $(0,1,1) \mapsto x_3$
- $(1,0,0) \mapsto x_4$
- $(1,0,1) \mapsto x_5$
- $(1,1,0) \mapsto x_6$
- $(1,1,1) \mapsto x_7$

est dit défini par la *règle* x, où $x = \sum_{i=0}^{7} x_i 2^i$.

Implémentez une fonction

```
void one_d_ca(uint8_t rule, int a[], int an, int nsteps)
```

similaire à `rule30`, qui prend la règle d'évolution de l'automate comme un argument supplémentaire `rule`. Comparez le résultat pour la règle 30 avec votre fonction précédente.

Q.4 : Écrivez une fonction faisant tourner une boucle infinie demandant à un utilisateur d'entrer une règle, et affichant 100 étapes d'évolution depuis le même état que dans les fonctions précédentes.

Même question pour un état initial valant 1 sur une dizaine de positions que vous choisirez.

Même question pour un état initial valant 1 sur environ la moitié de ses positions.

Même question pour un état initial nul.

Testez chacun des cas ci-dessus avec plusieurs règles, par exemple 30, 77, 90, 110, 177.