Recherche de Similarités Temporelles

Épreuve pratique d'algorithmique et de programmation
Concours commun des Écoles normales supérieures

Durée de l'épreuve : 3 heures 30 minutes

Juin 2025

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.

Il vous a été donné un numéro u_0 qui servira d'entrée à vos programmes. Les réponses attendues sont généralement courtes et doivent être données sur la fiche réponse fournie à la fin du sujet. À la fin du sujet, vous trouverez en fait deux fiches réponses. La première est un exemple des réponses attendues pour le numéro particulier $\widetilde{u_0}$. Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes. Vous indiquerez vos réponses (correspondant à votre u_0) sur la seconde et vous la remettrez à l'examinateur à la fin de l'épreuve.

En ce qui concerne la partie orale de l'examen, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures !

Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple : $O(n^2)$, $O(n \log n)$,...

Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main ou bien à l'aide de la fiche réponse type fournie en annexe.*** Enfin, il est recommandé de lire l'intégralité du sujet avant de commencer afin d'effectuer les bons choix de structures de données dès le début.

1 Préliminaires

Nous allons étudier le problème de recherche de similarités dans des bases de données de séries temporelles. Une série temporelle est une suite de réels représentant les valeurs successives d'une grandeur au cours du temps. Étant donné une base de données D, contenant n séries (numérotées avec $i \in [0, n-1]$) et une requête (une série temporelle Q), l'objectif est de trouver le plus proche voisin de Q dans D. Pour cela, nous devons être capables de mesurer la similarité entre deux séries de manière efficace. Nous pouvons aussi utiliser des méthodes d'approximation pour affiner la recherche. Nous allons aborder ces deux points dans le sujet.

La partie 1 est nécessaire pour pouvoir traiter le reste du sujet. Les parties 2 et 3 sont indépendantes, se concentrant sur des mesures de distances élastiques et des méthodes d'approximation plus fines.

Dans la plupart des questions d'implémentation, calculer la réponse nécessite une implémentation efficace. Nous vous suggérons de réfléchir à la complexité en temps avant de commencer l'implémentation. Si votre algorithme n'est pas assez efficace pour calculer certaines réponses, nous vous conseillons d'aborder les questions suivantes, avant de revenir optimiser votre implémentation.

Ce sujet contient des questions portant sur le calcul en virgule flottante. En OCaml, on utilisera `float_of_int` pour convertir un entier en flottant, `Float.floor` pour calculer la partie entière d'un flottant, et `sqrt` pour calculer la racine carrée d'un flottant. En Python, les équivalents sont `float()` et `math.floor()` pour la conversion, et `math.sqrt()` pour la racine carrée (nécessitant d'importer le module math de la manière suivante : `import math`). Il n'est pas autorisé d'utiliser un autre type de flottants que celui proposé par défaut dans Python, à savoir le type retourné par la fonction `float()`, équivalent à `float64`. Enfin, nous rappelons que les opérateurs OCaml d'addition, soustraction, multiplication et division sur des flottants sont `+.`, `-.`, `*.`, et `/.` respectivement.

1.1 Notations

Dans le sujet, n, k et l sont des entiers positifs, et strictement positifs dans le cas de n et l. Chaque question d'implémentation vous fournira plusieurs valeurs de n, k ou l sur lesquelles tester vos algorithmes.

Une série temporelle T de taille l est définie comme un vecteur $[t_0, t_1, ..., t_{l-1}] \in \mathbb{R}^l$. On note $|T| = l$ sa taille. Nous définissons la moyenne $\mu(T)$ de la série temporelle T par :

$$\mu(T) = \frac{1}{l} \sum_{k=0}^{l-1} t_k$$

Enfin, nous notons $D_{n,l} = [T_0, \ldots, T_{n-1}]$ une base de données contenant n séries temporelles de taille l.

1.2 Génération de nombres pseudo-aléatoires

Étant donné u_0, on considère la suite u générée par la relation de récurrence suivante :

$$u_{k+1} = (1\,103\,515\,245 \times u_k + 12\,345) \mod 2^{15}$$

L'entier u_0 vous est donné, et doit être recopié sur votre fiche réponse avec vos résultats. Une fiche réponse type vous est donnée en exemple, et contient tous les résultats attendus pour une valeur de u_0 différente de la vôtre (notée $\widetilde{u_0}$). Il vous est conseillé de tester vos algorithmes avec

cet $\widetilde{u_0}$ et de comparer avec la fiche de résultats fournie. Pour chaque calcul demandé, avec le bon choix d'algorithme le calcul ne devrait demander qu'au plus de l'ordre de la minute.

Lors du calcul de la suite u_k, nous vous suggérons de faire attention au problème de dépassement d'entier.

Question 1 *Calculer les valeurs* $u_k \mod 1000$ *, avec :*

a) $k = 1$ ***b)*** $k = 54$ ***c)*** $k = 684$ ***d)*** $k = 3945$

1.3 Génération de séries temporelles pseudo-aléatoires

A partir d'une certaine valeur de u_k, nous pouvons générer une série temporelle T de taille l de la manière suivante :

$$\overline{t_0} = u_k, \quad v_0 = u_k$$

$$\overline{t_{i+1}} = \begin{cases} \overline{t_i} + 1 & \text{si } 2 \times v_{i+1} - 2^{31} > 0 \\ \overline{t_i} - 1 & \text{sinon} \end{cases} \quad \text{avec } v_{i+1} = (1\,164\,317\,245 \times v_i + 12\,442) \mod 2^{31}$$

Une fois la série temporelle générée, nous normalisons chaque valeur t_i comme suit :

$$t_i = \overline{t_i} - \mu(T)$$

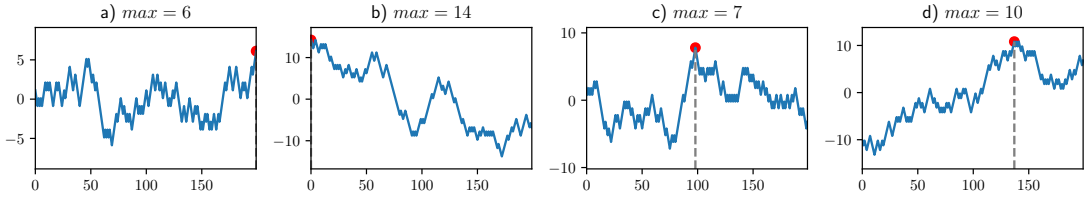


FIGURE 1 – Séries temporelles pseudo-aléatoires (en utilisant $\widetilde{u_0}$) de la Question 2

La figure 1 illustre quatre exemples de séries temporelles pseudo-aléatoires. **Dans la suite du sujet, nous notons T_k la série temporelle générée avec la valeur u_k.**

Question 2 *Calculer la partie entière de la valeur maximale t_i des séries temporelles T_k (de taille $l = 200$) pour les valeurs de k suivantes :*

a) $k = 1$ ***b)*** $k = 54$ ***c)*** $k = 684$ ***d)*** $k = 3945$

1.4 Une première distance et son approximation

La distance euclidienne ED entre deux séries temporelles $T_a = [t_{a,0}, \ldots, t_{a,l-1}]$, $T_b = [t_{b,0}, \ldots, t_{b,l-1}]$ de même taille l est définie comme suit :

$$\mathsf{ED}(T_a, T_b) = \sqrt{\sum_{k=0}^{l-1} (t_{a,k} - t_{b,k})^2}$$

Pour de très longues séries temporelles, le calcul de la distance euclidienne peut être coûteux, limitant significativement la rapidité de la recherche dans une base de données. Pour accélérer la recherche, nous pouvons au préalable calculer une représentation simplifiée de la série temporelle.

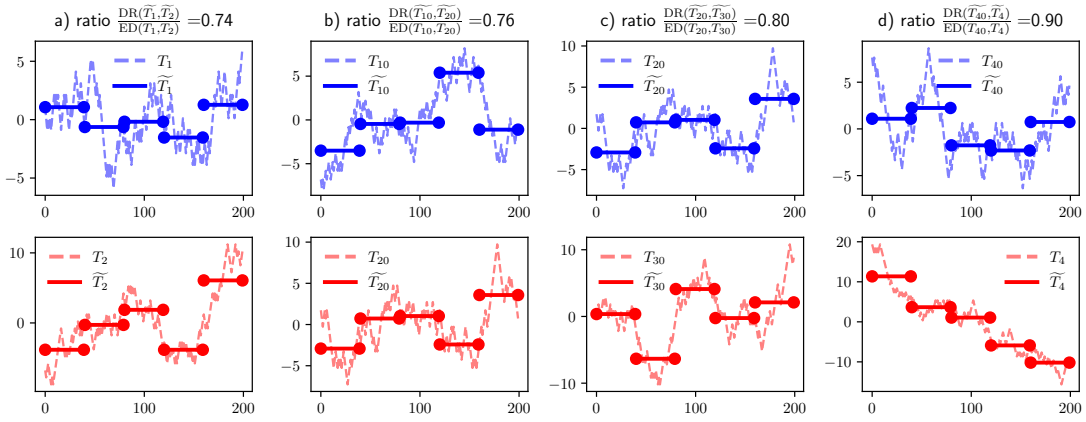


FIGURE 2 – Représentation PAA pour les séries temporelles (en utilisant $\widetilde{u_0}$) de la Question 3

Pour une série $T = [t_0, \ldots, t_{l-1}]$ et un entier N tel que $|T|$ est divisible par N et $\frac{N}{|T|} < 1$, la représentation PAA (*Piecewise Aggregate Approximation*) d'ordre N de T, notée $\widetilde{T} = [x_0, \ldots, x_{N-1}]$, est définie par :

$$x_i = \frac{N}{|T|} \sum_{j=\frac{|T|}{N}\times i}^{\frac{|T|}{N}\times(i+1)-1} t_j$$

La Figure 2 montre quatre exemples de représentation PAA pour quatre paires de séries temporelles. Nous pouvons maintenant définir une distance entre représentations PAA d'ordre N. La distance DR entre deux représentations PAA d'ordre N, $\widetilde{T_a} = [x_{a,0}, \ldots, x_{a,N-1}]$ et $\widetilde{T_b} = [x_{b,0}, \ldots, x_{b,N-1}]$ (avec $|T_a| = |T_b| = l$), est définie comme suit :

$$\mathsf{DR}(\widetilde{T_a}, \widetilde{T_b}) = \sqrt{\frac{l}{N} \sum_{i=0}^{N-1} (x_{a,i} - x_{b,i})^2}$$

De plus, DR est un minorant de ED (on l'admettra). Formellement :

$$\forall T_a, T_b \in \mathbb{R}^l, \mathsf{DR}(\widetilde{T_a}, \widetilde{T_b}) \leq \mathsf{ED}(T_a, T_b)$$

Question 3 *Pour deux séries temporelles T_{k_1}, T_{k_2} de taille $l = 200$, calculer le ratio $\frac{\mathsf{DR}(\widetilde{T_{k_1}},\widetilde{T_{k_2}})}{\mathsf{ED}(T_{k_1},T_{k_2})}$ (tronqué à deux chiffres après la virgule seulement) pour une représentation PAA d'ordre $N = 5$ et pour les paires (k_1, k_2) suivantes :*

a) $k_1 = 1, k_2 = 2$ ***b)*** $k_1 = 10, k_2 = 20$ ***c)*** $k_1 = 20, k_2 = 30$ ***d)*** $k_1 = 40, k_2 = 4$

1.5 À la recherche du plus proche voisin

Si nous considérons la distance euclidienne, le plus proche voisin $NN \in D_{n,l}$ d'une série temporelle Q (de taille l) est défini comme l'unique élement de $D_{n,l}$ tel que :

$$\forall T \in D_{n,l}, \quad \mathsf{ED}(Q, NN) \leq \mathsf{ED}(Q, T)$$

Pour la question suivante, on précalculera la base de données $D_{n,l} = [T_0, T_1, \ldots, T_{n-1}]$ ainsi que $\widetilde{D_{n,l}} = [\widetilde{T}_0, \widetilde{T}_1, \ldots, \widetilde{T}_{n-1}]$ pour $n = 200$, $l = 200$ et $N = 5$.

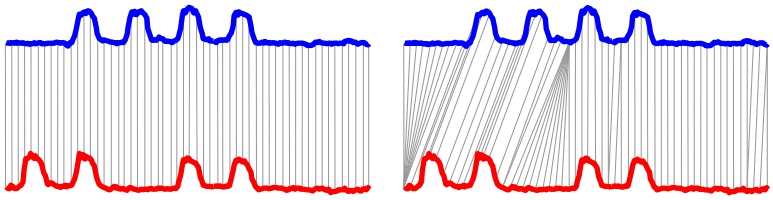
Question 4 *Proposer une implémentation d'un algorithme* findNN *qui renvoie le plus proche voisin $NN \in D_{n,l}$ de Q avec une complexité en temps dans le meilleur cas en $O(n \times N + l)$. Renvoyer k de telle sorte que $T_k \in D_{n,l}$ est le plus proche voisin de Q pour les Q suivants :*

a) $Q = T_{201}$ ***b)*** $Q = T_{202}$ ***c)*** $Q = T_{203}$ ***d)*** $Q = T_{204}$

Question à développer pendant l'oral 1 *Quelle est la complexité en temps de votre implémentation de* findNN *dans le pire cas en fonction de n, N et l ? Quelles conditions faut-il pour avoir une complexité en temps en $O(n \times N + l)$ dans le meilleur cas ?*

2 Une distance élastique

La distance euclidienne est une distance point-par-point, ne prenant pas en compte de possibles décalages temporels entre les sous-suites de deux séries temporelles (comme l'illustre la Figure 3(a)).



(a) Distance Euclidienne *(b) Dynamic Time Warping (DTW)*

FIGURE 3 – Comparaison de la distance Euclidienne et *Dynamic Time Warping* (DTW)

Pour pallier cette limitation, nous pouvons utiliser une autre mesure de distance appelée *Dynamic Time Warping* (DTW). L'objectif de cette dernière est de trouver le meilleur alignement possible entre les points des séries temporelles. Pour deux séries temporelles $T_a = [t_{a,0}, \ldots, t_{a,l_a-1}]$ et $T_b = [t_{b,0}, \ldots, t_{b,l_b-1}]$ (de taille l_a et l_b respectivement), DTW se calcule de la manière suivante :

$$\mathsf{DTW}(T_a, T_b) = \min_{\pi \in A(T_a, T_b)} \sqrt{\sum_{(i,j)\in\pi} (t_{a,i} - t_{b,j})^2}$$

Dans l'équation ci-dessus, un chemin d'alignement π de taille K (illustré en Figure 4(a)) est une suite de K paires d'entiers $((i_0, j_0), \ldots, (i_{K-1}, j_{K-1}))$ et $A(T_a, T_b)$ est l'ensemble de tous les chemins admissibles. Pour être considéré admissible, un chemin doit satisfaire les propriétés suivantes :

(P1) Le début et la fin de T_a et T_b coïncident :

$$\pi_0 = (0,0) \text{ et } \pi_{K-1} = (l_a - 1, l_b - 1)$$

(P2) La suite $(i_0, \ldots, i_{K-1})$ est croissante et contient tous les indices de T_a (de 0 à $l_a - 1$) et de même $(j_0, \ldots, j_{K-1})$ est croissante et contient tous les indices de T_b :

$$i_{k-1} \leq i_k \leq i_{k-1} + 1 \text{ et } j_{k-1} \leq j_k \leq j_{k-1} + 1 \text{ pour tout } k$$

(P3) Chaque paire (i, j) dans π est unique.

Question à développer pendant l'oral 2 *Pour deux séries temporelles T_a et T_b de taille l_a et l_b, quelle est l'ordre de grandeur de la complexité en temps dans le pire cas de la solution triviale consistant à évaluer tous les chemins admissibles dans $A(T_a, T_b)$?*

Pour permettre un calcul efficace de DTW pour deux séries temporelles T_a et T_b de taille l_a et l_b, on pourra identifier une relation de récurrence entre DTW et un sous-problème. Nous notons qu'il peut, en général, exister plusieurs chemins optimaux. Cependant, dans les questions suivantes, les valeurs ont été choisies de sorte que tous les chemins optimaux aient la même longueur.

Question 5 *Implémenter une fonction renvoyant la distance* DTW *et la longueur du chemin optimal π. Calculer la longueur du chemin optimal π modulo* 1000 *pour les paires de séries temporelles T_a, T_b (de taille l_a, l_b) suivantes :*

- ***a)*** *T_1, T_2 et $l_1 = 50, l_2 = 60$*
- ***b)*** *T_{10}, T_{20} et $l_{10} = 100, l_{20} = 80$*
- ***c)*** *T_{30}, T_{40} et $l_{30} = 2000, l_{40} = 1500$*
- ***d)*** *T_{40}, T_4 et $l_{40} = 5000, l_4 = 5000$*

Question à développer pendant l'oral 3 *Quelle est la complexité en temps de votre implémentation de* DTW *dans le pire cas en fonction de l_a et l_b ?*

Question à développer pendant l'oral 4 *Quelle est la complexité en mémoire de votre implémentation de* DTW *dans le pire cas en fonction de l_a et l_b ?*

2.1 Une élasticité restreinte

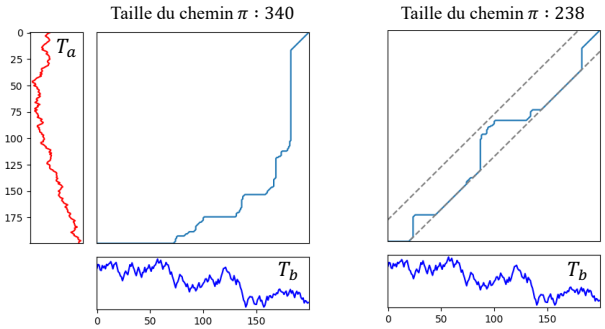
L'élasticité de la mesure DTW est robuste à de possibles décalages temporels. Cependant, le chemin optimal peut correspondre à des décalages extrêmes, inadéquats pour mesurer la similarité entre deux séries temporelles. Pour pallier ce problème, nous pouvons contraindre les chemins possibles dans $A(T_a, T_b)$. Une contrainte couramment utilisée s'appelle la *bande de Sakoe–Chiba* (illustrée par la Figure 4(b)). **Dans le cas spécifique de séries temporelles de même taille**, cette dernière contraint le calcul de DTW (d'une taille de bande w) de la manière suivante :

$$\forall (i, j) \in \pi, |i - j| \leq w$$

Pour la question suivante, nous ne considérons que le cas de séries temporelles de même taille. Nous nommons DTW_w la fonction calculant la distance DTW contrainte en utilisant la bande de Sakoe–Chiba de taille w.

Question 6 *Implémenter une fonction renvoyant la distance* DTW_w *et la longueur du chemin optimal π sous contrainte. Calculer la longueur du chemin π modulo* 1000 *pour les valeurs w et les paires de séries temporelles T_a, T_b (de taille l) suivantes :*

- ***a)*** *T_1, T_2 avec $w = 10, l = 10^2$*
- ***b)*** *T_{10}, T_{20} avec $w = 20, l = 10^3$*
- ***c)*** *T_{30}, T_{40} avec $w = 50, l = 10^4$*
- ***d)*** *T_{40}, T_4 avec $w = 5, l = 10^5$*



(a) Aucune contrainte *(b) Bande de Sakoe-Chiba*

FIGURE 4 – Comparaison de la distance DTW avec ou sans contrainte pour des séries temporelles de taille $l = 200$.

Question à développer pendant l'oral 5 *Quelle est la complexité en temps et en mémoire de votre implémentation de* DTW_w *dans le pire cas en fonction de* l *et* w *?*

3 Une approximation plus fine

Bien que la PAA offre une méthode simple et efficace pour réduire la dimension des séries temporelles, elle présente une limite importante : la division uniforme des segments peut cacher certaines évolutions temporelles. Pour résoudre ce problème, l'approche APCA (***Adaptive Piecewise Constant Approximation***) est plus flexible et précise.
L'APCA améliore la PAA en permettant une segmentation adaptative des séries temporelles. Au lieu d'imposer des segments de taille fixe, elle ajuste dynamiquement la longueur des intervalles dans le but de minimiser l'approximation. Formellement, l'approximation APCA consiste à diviser une série en N segments adaptatifs, chacun représenté par :

- Un intervalle $[b_i, e_i]$ où b_i et e_i sont respectivement les indices de début et de fin du segment i.
- Une valeur constante x_i qui représente tous les points de l'intervalle.

Ainsi, une approximation APCA de T est donnée par :

$$\mathsf{APCA}(T) = \{(x_i, [b_i, e_i])\}_{i=0}^{N-1}, \quad \text{avec } b_0 = 0, \quad e_{N-1} = l - 1$$

où chaque segment i satisfait :

$$e_i = b_{i+1} - 1, \quad x_i = \frac{1}{e_i - b_i + 1} \sum_{j=b_i}^{e_i} t_j$$

Pour un certain segment $s = (x_i, [b_i, e_i])$, nous définissons $l_s = e_i - b_i + 1$ comme étant la taille du segment s, ainsi que $\mathsf{Err}(s, T)$ l'erreur d'approximation de s sur T de la manière suivante :

$$\mathsf{Err}(s, T) = \sum_{j=b_i}^{e_i} (t_j - x_i)^2$$

Comme nous aurons à évaluer cette erreur d'approximation un grand nombre de fois, il est important de la calculer de manière efficace. Pour illustrer ce point, nous nous intéressons dans la question suivante à la recherche du segment $s_{min} = (x_{min}, [b_{min}, b_{min} + l_s[)$ de taille l_s de telle sorte que, pour tout segment s de taille l_s :

$$\mathsf{Err}(s_{min}, T) \leq \mathsf{Err}(s, T)$$

Question 7 *Implémenter une fonction* computeMinimalCost(T, l_s) *qui identifie le segment* s_{min} *et renvoie* b_{min} *modulo* 1000. *Évaluer votre implémentation sur les* T *(de taille* l*) et* l_s *suivants :*

a) T_0 *avec* $l = 10^3$ *et* $l_s = 10^2$

b) T_5 *avec* $l = 10^4$ *et* $l_s = 10^3$

c) T_{10} *avec* $l = 10^5$ *et* $l_s = 10^4$

d) T_{20} *avec* $l = 10^6$ *et* $l_s = 10^5$

Question à développer pendant l'oral 6 *Quelle est la complexité en temps de votre implémentation de* computeMinimalCost *dans le pire cas en fonction de* l *?*

3.1 Une première approche : un algorithme glouton

Nous pouvons utiliser un algorithme glouton pour calculer une représentation APCA qui n'est pas nécessairement optimale. Cet algorithme, nommée ***Bottom-up*** APCA, considère dans un premier temps la partition la plus fine, c'est-à-dire l segments de taille 1, puis, à chaque itération, fusionne les deux segments consécutifs conduisant à la plus faible erreur d'approximation (c'est-à-dire $\mathsf{Err}(s, T)$ avec s le nouveau segment fusionné). Pour cette sous-partie exclusivement, nous considérons seulement la partie entière de l'erreur d'approximation. L'algorithme s'arrête une fois que le nombre de segments atteint N. Le pseudo-code d'une telle solution est le suivant :

fonction BuAPCA(T, N)
  Créer la liste *segments* contenant l segments de taille 1.
  Calculer la liste *mergeCost* contenant la partie entière de l'erreur d'approximation si on fusionne deux segments consécutif ($l - 1$ elements)
  tant que $|segments| > N$ **:**
    Fusionner les deux premiers segments consécutifs (dans l'ordre d'apparition dans *mergeCost*) correspondant à min(*mergeCost*)
    Mettre à jour *mergeCost* (toujours avec la partie entière de l'erreur d'approximation) et *segments*
  fin
  renvoyer *segments*
fin

On rappelle que les fonctions `math.floor(x)` en Python et `Float.floor x` en OCaml permettent de calculer la partie entière d'un flottant x.

Pour la question suivante uniquement, afin de tenir compte des erreurs d'arrondi inhérentes au calcul en virgule flottante, nous accepterons des résultats avec une marge d'erreur de plus ou moins 10.

Question 8 *Implémenter la fonction* BuAPCA(T, N) *renvoyant l'*APCA *approximée de* T *de* N *segments. Pour des séries temporelles de taille* l*, calculer la longueur du segment le plus long, c'est-à-dire* $\max_{(x_i,[e_i,b_i])\in \text{APCA}(T,N)}(e_i - b_i + 1)$*, modulo* 1000*, pour :*

- ***a)*** T_0 *avec* $l = 10^3$ *et* $N = 10$
- ***b)*** T_5 *avec* $l = 10^4$ *et* $N = 7$
- ***c)*** T_{10} *avec* $l = 2 \times 10^4$ *et* $N = 20$
- ***d)*** T_{50} *avec* $l = 5 \times 10^4$ *et* $N = 50$

Question à développer pendant l'oral 7 *Quelle est la complexité en temps dans le pire cas votre implémentation de* BuAPCA *?*

Question à développer pendant l'oral 8 *Sans l'implémenter, quelle stratégie pourriez-vous mettre en place pour avoir une implémentation de* BuAPCA *avec une complexité en temps dans le pire cas en* $O(l \times \log l)$ *?*

3.2 Une solution optimale

L'algorithme BuAPCA ne permet pas toujours d'obtenir une approximation optimale d'une série temporelle. Pour un nombre de segments N donné, une APCA est optimale si elle minimise la somme des erreurs de ses segments, pour la fonction Err définie précédemment. La Figure 5 montre quatre exemples de représentation APCA optimales. Pour les séries temporelles de la question suivante, l'APCA optimale est unique.

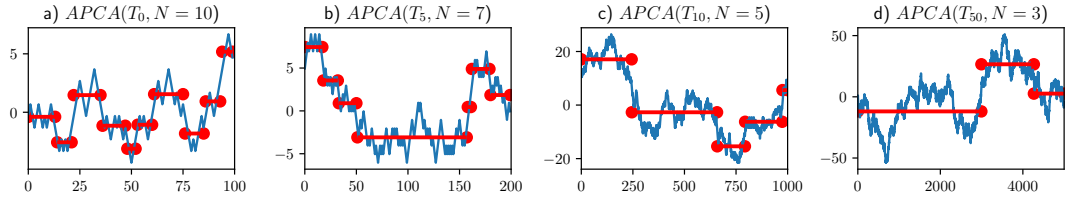


FIGURE 5 – Représentation APCA pour les séries temporelles (en utilisant $\widetilde{u_0}$) de la Question 9

Question 9 *Implémenter la fonction* APCA(T, N) *renvoyant l'*APCA *optimale de* T *avec* N *segments. Pour ces séries temporelles de taille* l*, calculer la longueur du segment le plus long, c'est-à-dire* $\max_{(x_i,[e_i,b_i])\in \text{APCA}(T,N)}(e_i - b_i + 1)$*, modulo* 1000*, pour :*

- ***a)*** T_0 *avec* $l = 100$ *et* $N = 10$
- ***b)*** T_5 *avec* $l = 200$ *et* $N = 7$
- ***c)*** T_{10} *avec* $l = 1000$ *et* $N = 5$
- ***d)*** T_{50} *avec* $l = 5000$ *et* $N = 3$

Question à développer pendant l'oral 9 *Quelle est la complexité en temps de votre implémentation de* APCA *dans le pire cas en fonction de* l *et* N *? Comparer avec la complexité en temps de la solution triviale consistant à tester toutes les segmentations possibles.*

Fiche réponse type : Recherche de Similarités Temporelles

$\widetilde{u_0}$: 79

Question 1

a)	564
b)	561
c)	491
d)	772

Question 2

a)	6
b)	14
c)	7
d)	10

Question 3

a)	0.74
b)	0.76
c)	0.80
d)	0.90

Question 4

a)	48
b)	157
c)	148
d)	57

Question 5

a)	84
b)	140
c)	281
d)	887

Question 6

a)	170
b)	340
c)	483
d)	631

Question 7

a) 594

b) 426

c) 329

d) 904

Question 8

a) 188

b) 430

c) 830

d) 256

Question 9

a) 15

b) 107

c) 414

d) 995

Fiche réponse : Recherche de Similarités Temporelles

Nom, prénom, u_0 : .....

Question 1

a)

b)

c)

d)

Question 2

a)

b)

c)

d)

Question 3

a)

b)

c)

d)

Question 4

a)

b)

c)

d)

Question 5

a)

b)

c)

d)

Question 6

a)

b)

c)

d)

Question 7

a)

b)

c)

d)

Question 8

a)

b)

c)

d)

Question 9

a)

b)

c)

d)