Projet d'Algorithmique : Classification des séries temporelles à l'aide de Dynamic Time Warping

Sarah Eliko, Fatima-Zahra Aklila , Samira Sonfack

10 December 2025

Contents

Introduction

Le mathématicien Al-Khwarizmi a rédigé de nombreux travaux qui furent traduits au XIIe siècle. C'est d'ailleurs de son nom que provient le mot « algorithme ». Un algorithme peut être défini comme un enchaînement d'instructions logiques permettant de résoudre un problème. Il constitue la traduction d'un raisonnement en un pseudo-code. Mais dans quelles circonstances construit-on un algorithme, et comment intervient l'algorithmique dans ce processus ? Lorsqu'une problématique nécessite une méthode de calcul efficace, il devient nécessaire d'adopter une démarche algorithmique. L'algorithmique regroupe l'ensemble des méthodes génériques permettant de résoudre des problèmes. Elle s'intéresse aussi à l'étude de la complexité de ces méthodes, c'est-à-dire à leur rapidité, leur efficacité et leur capacité à s'adapter à différents cas. Autrement dit, elle ne se limite pas à la simple écriture d'un algorithme, mais vise à concevoir la meilleure approche possible pour atteindre un résultat fiable et optimisé. Aujourd'hui, au XXIe siècle, de nombreux mathématiciens et informaticiens ont inventé, amélioré et publié diverses méthodes de résolution. Ces avancées nous permettent de disposer d'outils conceptuels pour résoudre des problèmes complexes.

La résolution d'une problématique passe d'abord par l'évaluation de la complexité du problème, puis par le choix d'une méthode adaptée, d'où l'utilisation de l'algorithmique. Vient ensuite la construction du programme à l'aide de l'algorithme et en utilisant un langage de programmation choisi. Après l'exécution du programme, un résultat concret est obtenu, traduisant la pertinence de la méthode employée.

Contexte

Une série temporelle est une suite chronologique de valeurs numériques. Les séries temporelles sont utilisées dans plusieurs domaines, comme en médecine, par exemple pour le suivi d'une constante vitale au cours du temps, ou encore en finance pour le suivi du cours d'une action. Dans ces deux disciplines, l'analyse de ces séries temporelles peut être cruciale, notamment pour le diagnostic d'une anomalie ou de la détection d'un comportement particulier. Dans notre cas, nous allons nous intéresser à une méthode permettant de classifier les séries temporelles. Dans un premier temps, nous allons utiliser deux algorithmes qui vont nous permettre de comparer deux séries entre elles. Plus précisément, nous allons utiliser l'algorithme DTW, puis une amélioration de cet algorithme appelée FastDTW. Enfin, nous choisirons le meilleur algorithme pour la classification.

1 Dynamic Time Warping

L'algorithme Dynamic Time Warping (DTW) ou Déformation Temporelle Dynamique, permet de mesurer la similarité entre deux séries temporelles qui peuvent différer en vitesse ou en phase.

Cette méthode aligne de manière non linéaire deux séquences temporelles pour obtenir la correspondance optimale entre leurs éléments.

Soient deux séries temporelles :

$$X = (x_1, x_2, \dots, x_n) \quad \text{et} \quad Y = (y_1, y_2, \dots, y_m)$$

On définit la distance D entre deux points correspondants :

$$D(i,j) = (x_i - y_j)^2$$

Le but du DTW est de construire une matrice de coût cumulatif C, où chaque élément $C(i,j)$ représente le coût minimal d'alignement des sous-séquences $(x_1, \dots, x_i)$ et $(y_1, \dots, y_j)$.
Cette matrice est calculée récursivement selon :

$$C(i,j) = \begin{cases} D(1,1), & \text{si } i = 1, j = 1, \\ D(i,1) + C(i-1,1), & \text{si } j = 1, i > 1, \\ D(1,j) + C(1,j-1), & \text{si } i = 1, j > 1, \\ D(i,j) + \min\big(C(i-1,j),\ C(i,j-1),\ C(i-1,j-1)\big), & \text{sinon.} \end{cases}$$

L'objectif de la méthode DTW est de trouver le chemin optimal qui minimise la distance totale entre les deux séries temporelles.

La distance DTW finale est alors donnée par :

$$DTW(n,m) = \min_{\pi} \sum_{(i,j)\in\pi} D(i,j)$$

où π désigne l'ensemble des chemins de warping valides reliant les points $(1,1)$ et (n,m).

Dans la pratique, cette distance correspond à la valeur du dernier élément de la matrice de coût cumulatif :

$$DTW(X,Y) = C(n,m)$$

1.1 Algorithme du DTW

L'algorithme DTW repose sur la programmation dynamique.

Il consiste à remplir progressivement la matrice de coût cumulé C.

```
Algorithme DTW(x, y):
1. Initialiser D[i,j] = |x_i - y_j|
2. Initialiser C[1,1] = D[1,1]
3. Remplir la première ligne et la première colonne de C
4. Pour i=2..n, j=2..m :
     C[i,j] = D[i,j] + min(C[i-1,j-1], C[i-1,j], C[i,j-1])
5. Rétrotraçage pour obtenir le chemin optimal
6. Retourner C, traceback, path
```

Path correspond au chemin optimal. La complexité en temps de l'algorithme DTW est de l'ordre de $O(NM)$.

1.1.1 Exemple d'application

Considérons les deux séries temporelles suivantes :

$$X = (1, 2, 3, 4) \quad \text{et} \quad Y = (1, 1, 2, 3, 5)$$

Calcul de la matrice de distances locales $D(i, j)$:

$$D = \begin{bmatrix} 0 & 0 & 1 & 4 & 16 \\ 1 & 1 & 0 & 1 & 9 \\ 4 & 4 & 1 & 0 & 4 \\ 9 & 9 & 4 & 1 & 1 \end{bmatrix}$$

Calcul de la matrice de coût cumulatif $C(i, j)$:

En remplissant la matrice pas à pas, on obtient :

$$C = \begin{bmatrix} 0 & 0 & 1 & 5 & 21 \\ 1 & 1 & 0 & 1 & 10 \\ 5 & 5 & 1 & 0 & 4 \\ 14 & 14 & 5 & 1 & 2 \end{bmatrix}$$

La distance DTW entre X et Y est donnée par le dernier élément de la matrice :

$$DTW(X, Y) = C(4, 5) = 2$$

Cette valeur faible indique que les deux séries sont similaires, même si elles ne sont pas parfaitement alignées temporellement.

Le DTW a « déformé » le temps pour trouver un alignement optimal entre les points de X et Y.

```
x <- c(1, 3, 4, 9)
y <- c(1, 3, 7, 8, 9)
dtw_result <- dtw(x, y)
dtw_result
```

```
## $DTW_matrix
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    2    8   15   23
## [2,]    2    0    4    9   15
## [3,]    5    1    3    7   12
## [4,]   13    7    3    4    4
##
## $traceback_matrix
##      [,1] [,2]   [,3]   [,4]   [,5]
## [1,] ""   "left" "left" "left" "left"
```

```
## [2,] "up" "diag" "left" "left" "left"
## [3,] "up" "up"   "diag" "left" "left"
## [4,] "up" "up"   "diag" "diag" "left"
##
## $path
## $path[[1]]
## [1] 1 1
##
## $path[[2]]
## [1] 2 2
##
## $path[[3]]
## [1] 3 3
##
## $path[[4]]
## [1] 4 4
##
## $path[[5]]
## [1] 4 5
```

Nous cherchons maintenant à retrouver expérimentalement la complexité de l'algorithme DTW. Lorsque les deux séries ont la même longueur n, le temps d'exécution peut alors s'écrire sous la forme :

$$T(n) = C \cdot n^2,$$

où C est une constante correspondant au coût d'un calcul élémentaire. Cette constante ne dépend pas de n.

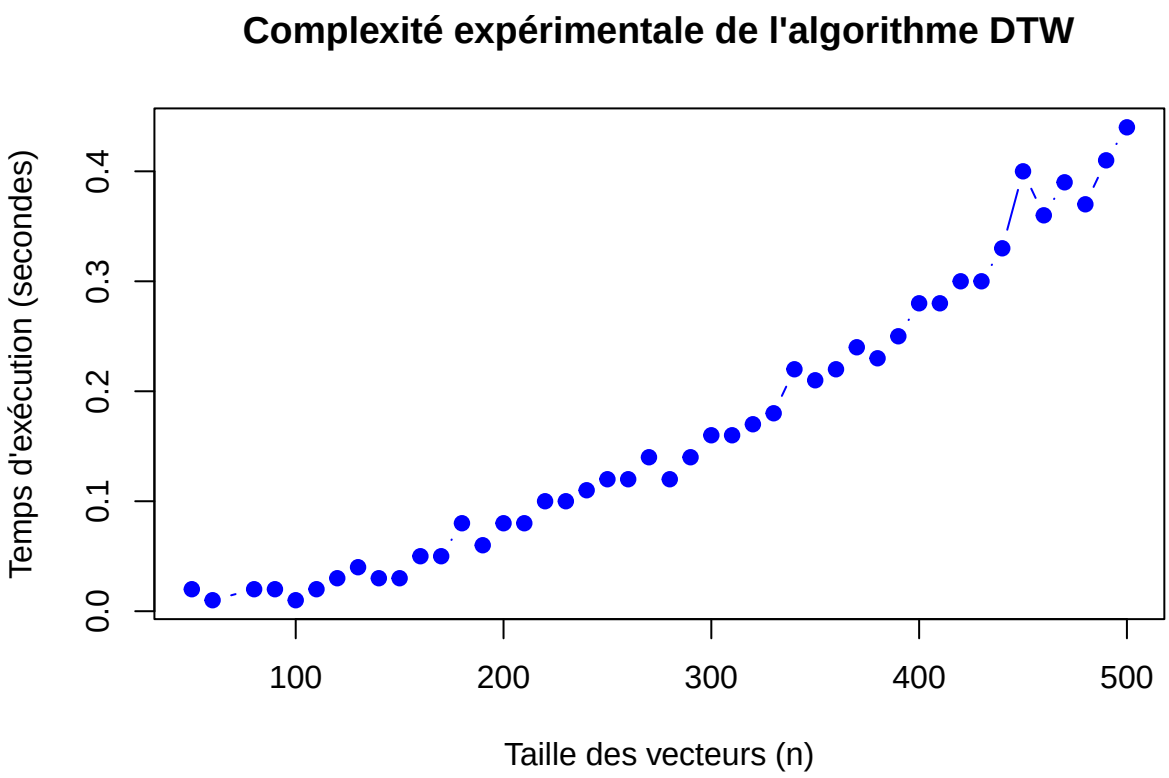
Pour analyser la complexité à partir des mesures expérimentales, nous traçons le graphique en échelle log-log. En prenant le logarithme de l'expression précédente, on obtient :

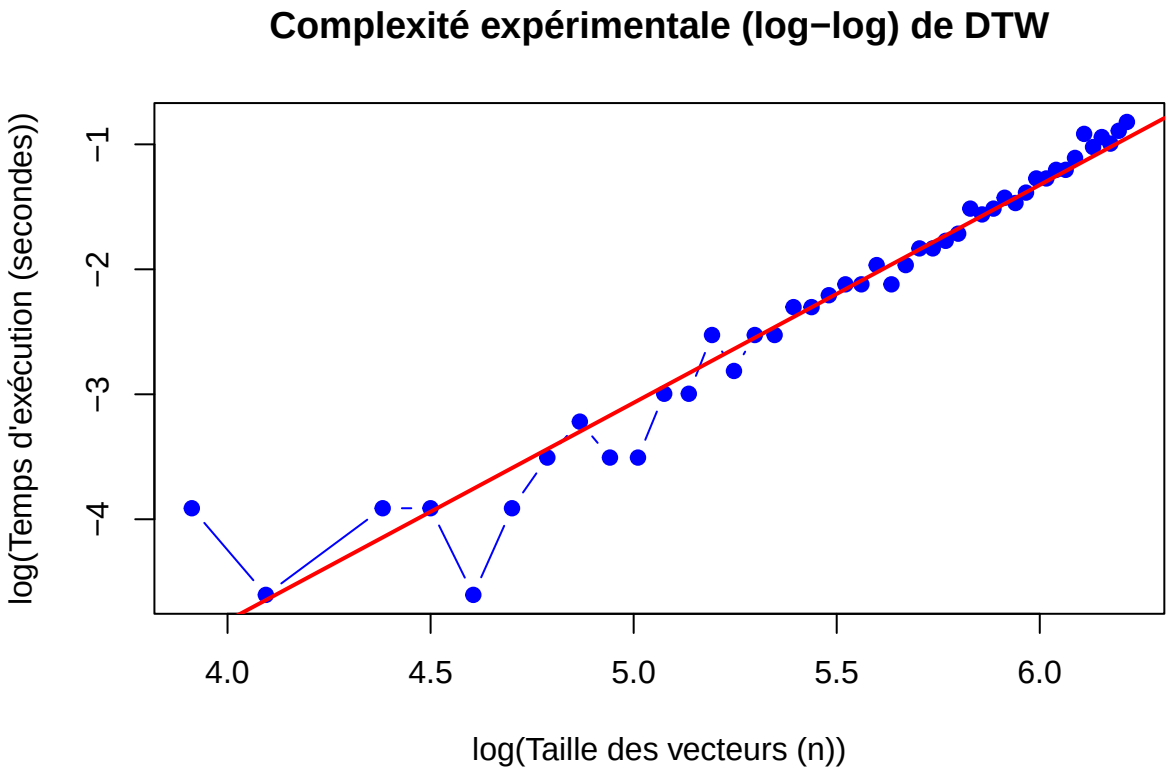
$$\log T(n) = \log C + 2 \log n.$$

Cette équation est de la forme d'une droite, dont l'ordonnée à l'origine est $\log C$ et dont la pente vaut 2. Ainsi, si le graphique $\log(n)$ – $\log(T)$ présente une pente proche de 2, cela confirme expérimentalement la complexité quadratique théorique du DTW.

1.1.2 Simulation de la complexité

```r
# R
res <- simulation_dtw(debut = 10, fin = 500, pas = 10)
```


Complexité expérimentale de l'algorithme DTW
Temps d'exécution (secondes)
0.0
0.1
0.2
0.3
0.4
100
200
300
400
500
Taille des vecteurs (n)

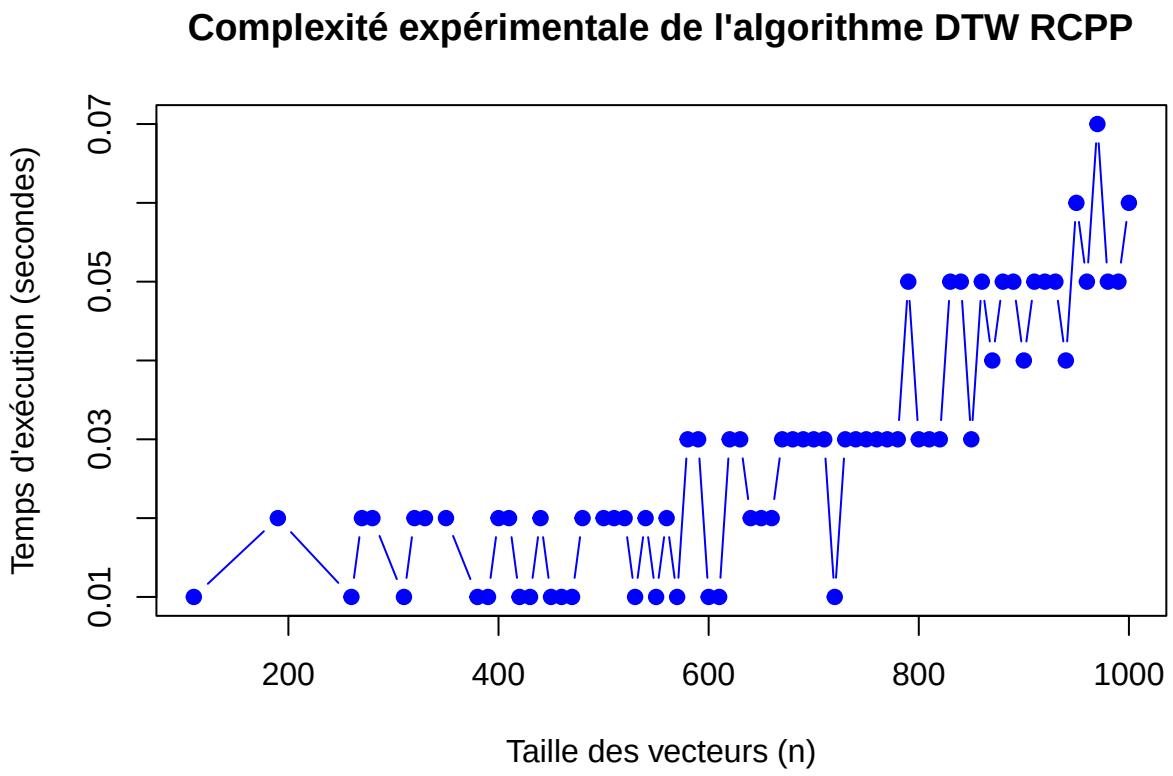

Complexité expérimentale (log−log) de DTW
log(Temps d'exécution (secondes))
−4
−3
−2
−1
4.0
4.5
5.0
5.5
6.0
log(Taille des vecteurs (n))

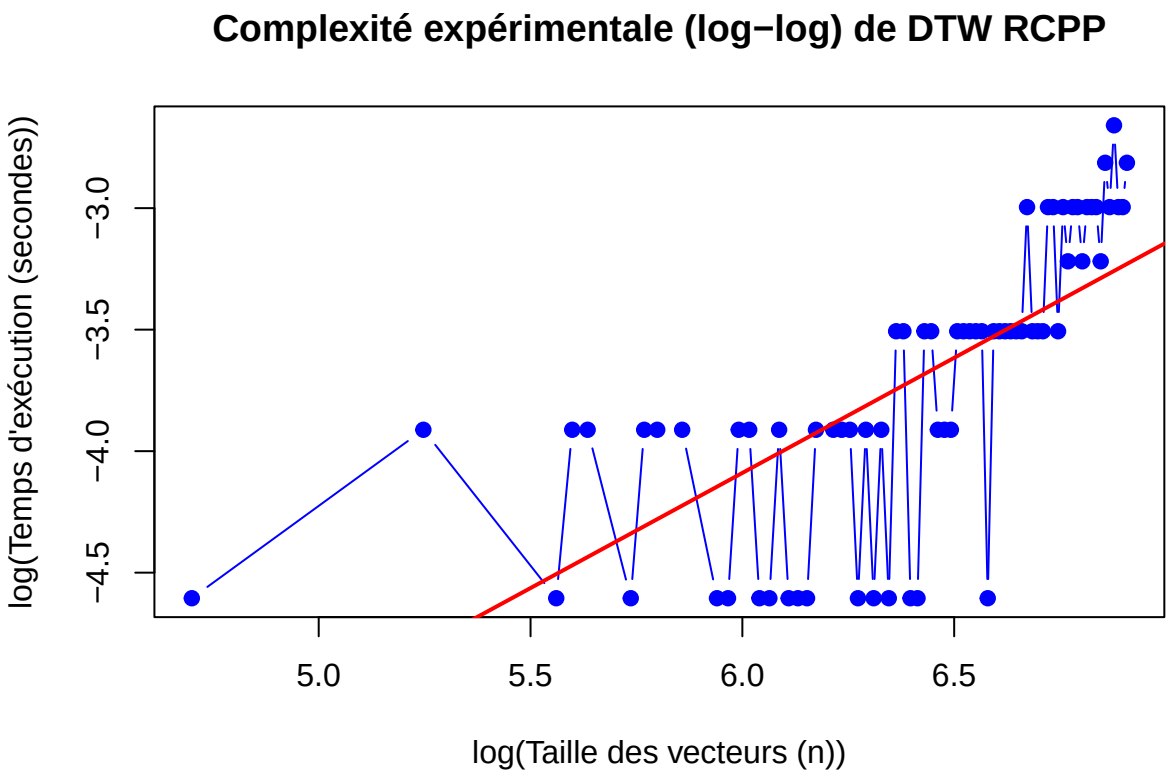
```
## Pente estimée sur le graphique log-log : 1.74
```

```
res$pente
```

```
## log(Taille)
##    1.743386
```

```
# C++
res <- simulation_dtw_rcpp(debut = 10, fin = 1000, pas = 10)
```






```
## Pente estimée sur le graphique log-log : 0.95
```

```
res$pente
```

```
## log(Taille)
##   0.9474182
```

1.2 Algorithme du DTW avec bande de Sakoe Chiba fixe - Sakoe Chiba Band

L'algorithme de **Sakoe–Chiba** introduit une contrainte locale appelée **bande de Chiba** afin de limiter les déformations autorisées lors du calcul du *Dynamic Time Warping* (DTW). Dans la formulation classique de la DTW, la matrice de coût est entièrement parcourue, ce qui conduit à une complexité temporelle en $O(N^2)$ pour deux signaux de longueur N.

Or lorsqu'on applique l'algorithme sur la même série on voit que le chemin optimal correspond à la diagonale et s'en éloigne d'autant plus que les séries diffèrent. La bande de Chiba impose que l'alignement optimal reste à l'intérieur d'une fenêtre de largeur prédéfinie autour de la diagonale principale.Une variante propose une fenêtre mobile sous forme de matrice dde "région": - pour un point (i, j), seules les cellules telles que

$$|i - j| \leq \text{radius}$$

sont considérées ;

On considèrera ici à **une taille de bande fixe radius** dans toutes les directions. Ainsi, seuls les coûts des cellules situées dans cette zone restreinte sont évalués.

Grâce à cette contrainte, le nombre d'opérations nécessaires est considérablement réduit, faisant passer la complexité temporelle à :

$$O(N \cdot w)$$

où w est la demi-largeur de la bande, généralement beaucoup plus petite que N.

En limitant les correspondances possibles, l'algorithme améliore non seulement l'efficacité computationnelle, mais également la robustesse de l'alignement, tout en conservant une bonne précision lorsque les signaux ne présentent pas de fortes distorsions temporelles.

1.2.1 Application : comparaison de cours de bourse des stocks d'une entreprise et d'un stock de matière première (pétrole)

Problématique : Les séries sont à des fréquences différentes. Déterminer l'appartenance d'une entreprise à un secteur , évaluer analytiquement la corrélation entre les cours de bourse

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(gridExtra)
```

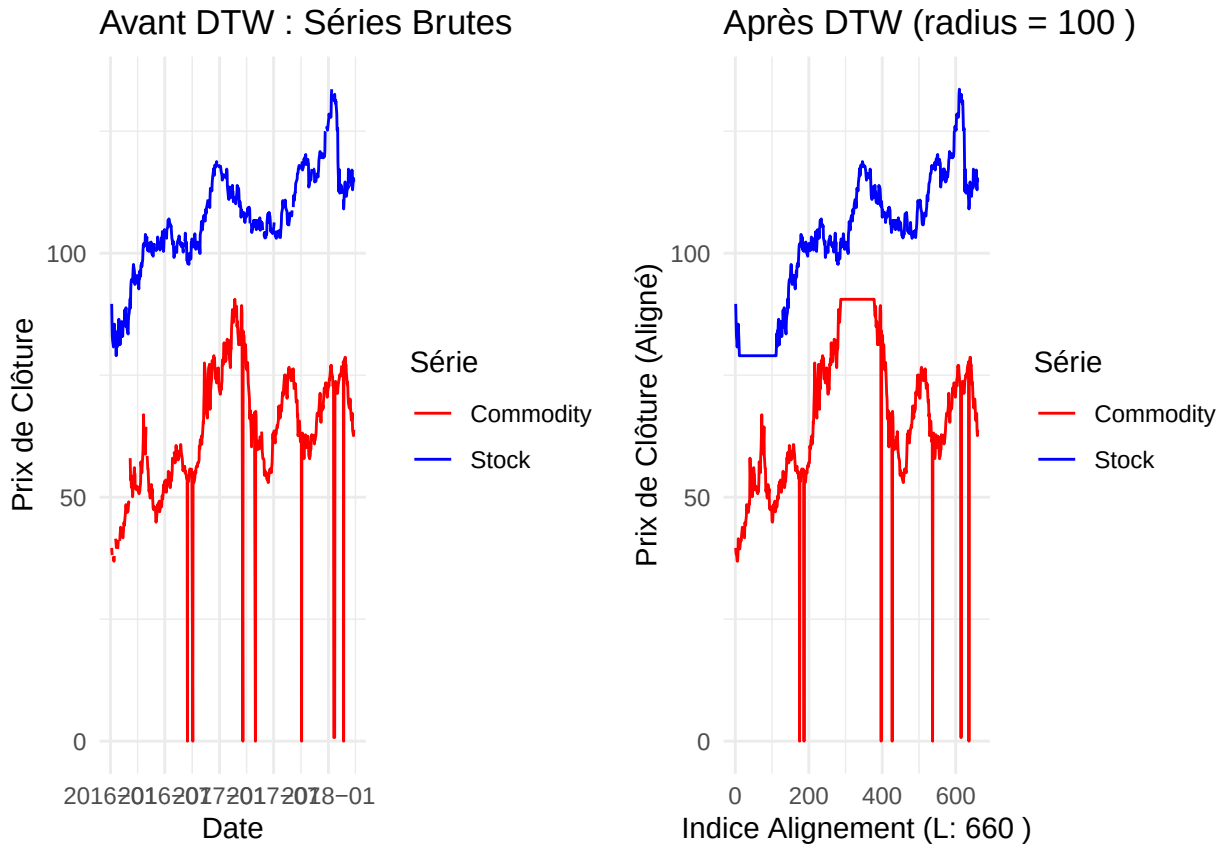
```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
###  Charge les données depuis le package
path_stock <- system.file("extdata", "stock.csv", package = "ProjetM2Algo")
path_com <- system.file("extdata", "commodity.csv", package = "ProjetM2Algo")
```

```r
df_stock <- read.csv(path_stock)
df_com <- read.csv(path_com)

df_stock$Date <- as.Date(df_stock$Date)
df_com$Date <- as.Date(df_com$Date)



X <- df_stock$Close
Y <- df_com$Close

radius <- 100
dtw_result <- dtw_sakoe_chiba_rcpp(X, Y, radius = radius)


#Indices pour radius
path <- dtw_result$path
idx_x <- sapply(path, `[[`, 1)
idx_y <- sapply(path, `[[`, 2)

aligned_X <- X[idx_x]
aligned_Y <- Y[idx_y]

df_aligned <- data.frame(
  Index = seq_along(aligned_X),
  Stock = aligned_X,
  Commodity = aligned_Y
)



df_before <- full_join(
  df_stock %>% rename(Stock = Close),
  df_com %>% rename(Commodity = Close),
  by = "Date"
)

plot_before <- ggplot(df_before, aes(x = Date)) +
  geom_line(aes(y = Stock, color = "Stock")) +
  geom_line(aes(y = Commodity, color = "Commodity")) +
  labs(title = "Avant DTW : Séries Brutes",
       y = "Prix de Clôture",
       color = "Série") +
  theme_minimal() +
  scale_color_manual(values = c("Stock" = "blue", "Commodity" = "red"))
```

```r
#radius1
plot_after <- ggplot(df_aligned, aes(x = Index)) +
  geom_line(aes(y = Stock, color = "Stock")) +
  geom_line(aes(y = Commodity, color = "Commodity")) +
  labs(title = paste("Après DTW (radius =", radius, ")"),
       x = paste("Indice Alignement (L:", length(aligned_X), ")"),
       y = "Prix de Clôture (Aligné)",
       color = "Série") +
  theme_minimal() +
  scale_color_manual(values = c("Stock" = "blue", "Commodity" = "red"))

#affichage
gridExtra::grid.arrange(plot_before, plot_after, ncol = 2)
```



1.2.2 Calcul théorique de complexité

Soit deux séries temporelles de longueur égale N.

Avec **une bande de Chiba de radius** r (fenêtre fixe autour de la diagonale), seules les cellules telles que $|i - j| \leq r$ sont calculées.

Chaque ligne de la matrice contient au maximum $2r + 1$ cellules évaluées. La complexité

devient donc :

$$T_{\text{DTW\_bande}}(N, r) = O(N \cdot (2r + 1)) \approx O(N \cdot r)$$

où r est généralement beaucoup plus petit que N.

Ainsi, la réduction de la complexité par rapport au DTW classique est significative, surtout pour de grandes séries temporelles.

Représentation log-log Pour visualiser la complexité, on peut utiliser une **échelle log-log** :

$$x = \log(N), \quad y = \log(T(N))$$

- Pour le **DTW avec bande fixe** :

$$y = \log(N \cdot r) = \log(r) + \log(N) \approx \log(N) \quad \text{si } r \text{ constant}$$

En pratique :

Sur un graphique log-log, la **pente de la droite** correspond à l'exposant de N. Donc une pente 1

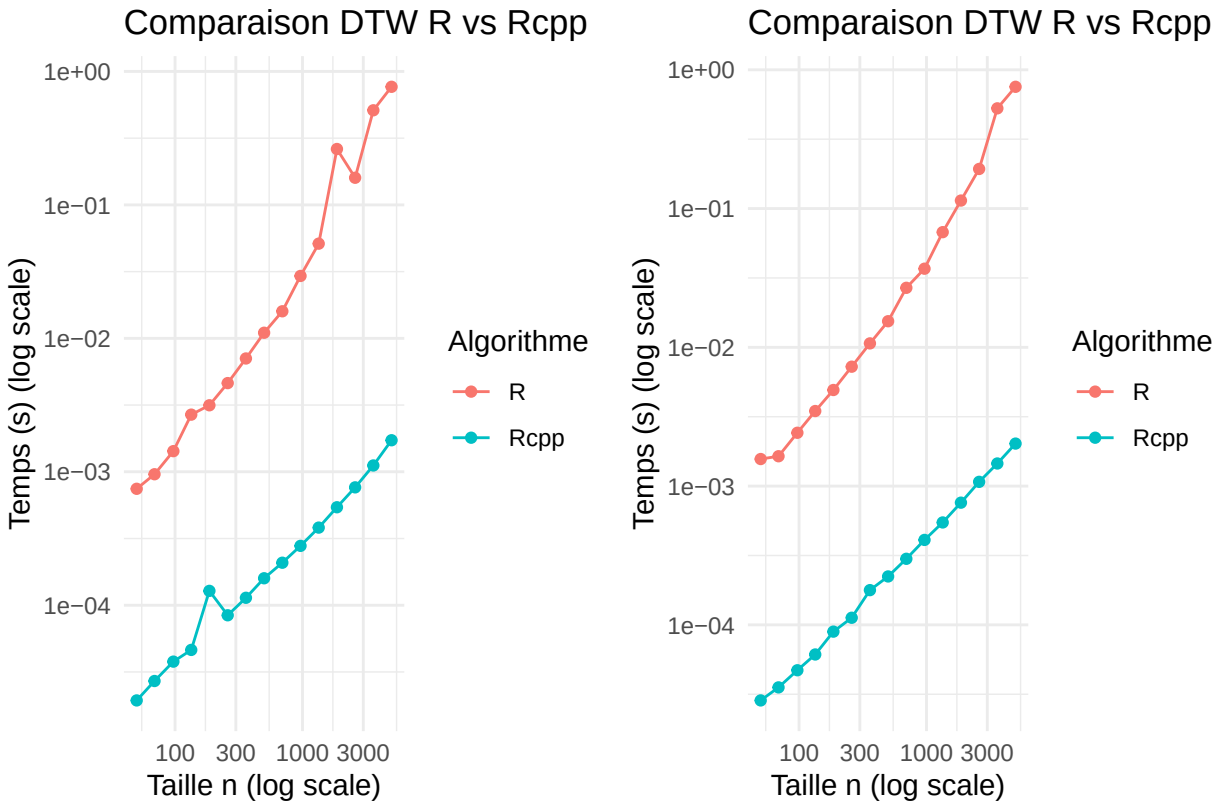
###Simulation de la complexité temporelle

```
n_values <- round(exp(seq(log(50), log(5000), length.out = 15)))
res1 <- simulate_dtw_sakoe_chiba(
      n_values,
      algo_r = dtw_sakoe_chiba,
      algo_rcpp = dtw_sakoe_chiba_rcpp,
      radius = 2,
      plot = TRUE
      )
```

```
# Simulation 2 (Radius = 5)
res2 <- simulate_dtw_sakoe_chiba(
    n_values,
    algo_r = dtw_sakoe_chiba,
    algo_rcpp = dtw_sakoe_chiba_rcpp,
    radius = 5, # Changer le rayon
    plot = TRUE
)
```

```r
library(gridExtra)

gridExtra::grid.arrange(
    res1$plot,
    res2$plot,
    ncol = 2,
    top = "Comparaison de la Complexité R vs Rcpp sous différentes contraintes radius=1
)
```

omparaison de la Complexité R vs Rcpp sous différentes contraintes radius=1 et radius=




```r
# Affichage des pentes
cat(" Pentes de Complexité par Implémentation :\n")
```

```
##  Pentes de Complexité par Implémentation :
```

```r
cat("\n* Pente R (Radius 1) : **", round(res1$slope_r, 4), "**")
```

```
##
## * Pente R (Radius 1) : ** 1.5307 **
```

```r
cat("\n* Pente R (Radius 2) : **", round(res2$slope_r, 4), "**")
```

```
##
## * Pente R (Radius 2) : ** 1.3651 **
```

```r
cat("\n* Pente Rcpp (Radius 1) : **", round(res1$slope_rcpp, 4), "**")
```

```
##
## * Pente Rcpp (Radius 1) : ** 0.9214 **
```

```r
cat("\n* Pente Rcpp (Radius 2) : **", round(res2$slope_rcpp, 4), "**")
```

```
##
## * Pente Rcpp (Radius 2) : ** 0.9375 **
```

On voit conformément au résultat théorique attendu qu'on a une complexité $O(N)$ avec un pente très proche de 1 pour le code Rcpp et légèrement supérieure à 1 en R. On vérifie également que la **complexité varie peu selon le radius** . Les résultats autour de **1.45** pour le code R mettent en évidence les faiblesses de l'interpréteur R qui introduit un temps de calcul supplémentaire qui augmente avec la valeur de N. Rcpp à contrario est compilé et facilite les calculs lourds tels que les boucles imbriquées. De plus les type Rcpp optimise la gestion de la mémoire.

1.3 Segmental DTW

Le segmental DTW (SDTW) est une variante du DTW classique utilisée pour comparer deux séries temporelles par segments plutôt que globalement.

Il repose sur 3 idées principales : - Au lieu de comparer toute la série X avec toute la série Y, on découpe Y en segments de longueur fixe. - On effectue le DTW sur chaque segment séparément. - Ensuite, on combine les distances segmentales pour obtenir une distance globale ou un alignement segment par segment.

1.3.1 Etapes de Segmental DTW

1.3.1.1 Segmentation

Dans le code, la première étape consiste à diviser la série serie2 en segments de longueur fixe segment_length :

```r
serie1 <- c(1, 3, 4, 9, 8)
serie2 <- c(1, 2, 3, 4, 7, 8, 9, 10)
segment_length <- 2
segments <- segment_series(serie2, segment_length)
```

Cette étape permet de traiter serie2 par blocs, ce qui est plus efficace et facilite l'analyse locale.

1.3.1.2 Calcul du DTW pour chaque segment

Ensuite, pour chaque segment, le code calcule le DTW entre serie1 complète et les segments courants de la serie2 :

```r
dtw_segments <- compute_segment_dtw(serie1, serie2, segments)
dtw_segments[1]
```

```
## [[1]]
```

```
## [[1]]$DTW_matrix
##      [,1] [,2]
## [1,]    0    1
## [2,]    2    1
## [3,]    5    3
## [4,]   13   10
## [5,]   20   16
##
## [[1]]$traceback_matrix
##      [,1] [,2]
## [1,] ""   "left"
## [2,] "up" "diag"
## [3,] "up" "up"
## [4,] "up" "up"
## [5,] "up" "up"
##
## [[1]]$path
## [[1]]$path[[1]]
## [1] 1 1
##
## [[1]]$path[[2]]
## [1] 2 2
##
## [[1]]$path[[3]]
## [1] 3 2
##
## [[1]]$path[[4]]
## [1] 4 2
##
## [[1]]$path[[5]]
## [1] 5 2
```

Chaque élément de dtw_segments contient la matrice des coûts cumulés frame-level, le chemin optimal et la matrice de traces pour reconstruire le chemin. Cette étape correspond à l'alignement local segment par segment, au lieu d'un DTW global sur toute la série 2.

1.3.1.3 Calcul des coûts segmentaux

Pour combiner les distances locales, le code extrait la dernière ligne de chaque matrice DTW pour chaque segment :

```
seg_cost_list <- compute_segment_level_cost(dtw_segments)
seg_cost_list
```

```
## [[1]]
## [1] 20 16
##
## [[2]]
```

```
## [1] 14 11
##
## [[3]]
## [1] 16 14
##
## [[4]]
## [1] 20 21
```

La dernière ligne de DTW_matrix représente le coût pour atteindre chaque frame de serie1 à la fin du segment. seg_cost_list devient la base pour calculer le cumulative cost global segment-level.

1.3.1.4 Cumulative cost des segments

Le code cumule les coûts des segments pour obtenir la distance globale :

```
n_seg <- length(segments)
n_len <- length(serie1)
cum_cost <- matrix(Inf, nrow=n_seg, ncol=n_len)
cum_cost[1, 1:length(seg_cost_list[[1]])] <- seg_cost_list[[1]]

for(i in 2:n_seg){
  seg_cost <- seg_cost_list[[i]]
  for(j in 1:length(seg_cost)){
    min_prev <- min(cum_cost[i-1, j:n_len])
    cum_cost[i,j] <- seg_cost[j] + min_prev
  }
}

cum_cost
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   20   16  Inf  Inf  Inf
## [2,]   30   27  Inf  Inf  Inf
## [3,]   43   41  Inf  Inf  Inf
## [4,]   61   62  Inf  Inf  Inf
```

cum_cost[i,j] : coût minimal pour atteindre la position j dans serie1 après avoir traité i segments de serie2.

On effectue une recherche du minimum cumulatif sur les positions restantes pour connecter les segments.

1.3.1.5 Backtrace segment-level

Une fois la matrice cumulative calculée, on retrace le chemin optimal segment par segment. Cette étape permet de retrouver la séquence optimale de segments qui minimise la distance globale.

```
path_seg <- list()
i <- n_seg
j <- which.min(cum_cost[n_seg, ])
while(i > 0){
  path_seg <- append(list(c(i,j)), path_seg)
  if(i==1) break
  j <- which.min(cum_cost[i-1, j:n_len])
  i <- i-1
}
path_seg
```

```
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 2 1
##
## [[3]]
## [1] 3 2
##
## [[4]]
## [1] 4 1
```

1.3.1.6 Reconstruction du chemin frame-level final et calcul de la distance de DTW

Enfin, le code reconstitue le chemin frame-level global à partir des chemins de chaque segment et retourne la distance DTW finale.

```
result <- segmental_dtw_two_series(serie1, serie2, segment_length = 2)

cat("Distance DTW finale segmentale :", result$dtw_distance, "\n")
```

```
## Distance DTW finale segmentale : 61
```

```
cat("Chemin frame-level final :\n")
```

```
## Chemin frame-level final :
```

```
print(result$final_path)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    1    3
## [4,]    1    5
## [5,]    2    5
## [6,]    1    7
```

1.3.1.7 Evaluation de la compléxité de Segmental DTW

Soient deux séries de longueurs n et m, et un segment de longueur L. Si la série cible est partitionnée en

$$S = \lceil m/L \rceil$$

segments, la complexité totale du **Segmental DTW** s'écrit :

$$O(S \cdot n \cdot L) = O(n \cdot m)$$

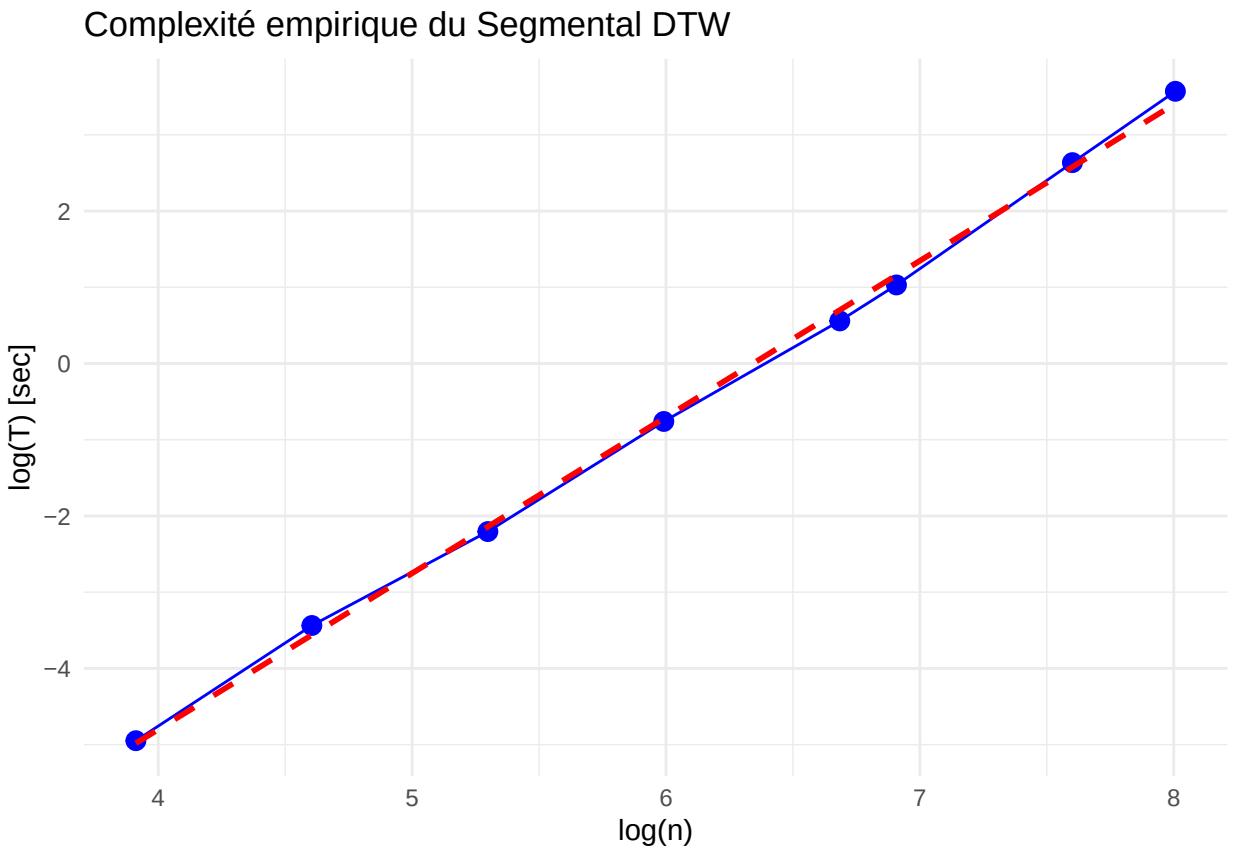
Ce qui correspond à la complexité théorique du DTW global, mais avec plusieurs avantages pratiques : - **Mémoire optimisée :** seules des matrices de taille n×L sont allouées à chaque étape, limitant l'usage de la mémoire vive.

- **Possibilité de parallélisation :** chaque segment peut être traité indépendamment, ouvrant la voie à des accélérations sur architectures multi-cœurs.
- **Alignements locaux précis :** le découpage segmental permet de capturer des correspondances locales qui pourraient être diluées dans un DTW global.

```
library(ggplot2)
res_complexity <- segmental_dtw_complexity()
```

1.3.1.8 Test de la complexité :

```
## n = 50 -> Temps = 0.0071 sec
## n = 100 -> Temps = 0.0322 sec
## n = 200 -> Temps = 0.1103 sec
## n = 400 -> Temps = 0.4675 sec
## n = 800 -> Temps = 1.7487 sec
## n = 1000 -> Temps = 2.8022 sec
## n = 2000 -> Temps = 13.9535 sec
## n = 3000 -> Temps = 35.5527 sec
## `geom_smooth()` using formula = 'y ~ x'
```




```
print(paste("Exposant empirique estimé :", round(res_complexity$slope, 2)))
```

```
## [1] "Exposant empirique estimé : 2.05"
```

Cette valeur de pente confirme que la méthode est très efficace pour de longues séries segmentées, tout en restant fidèle à sa complexité théorique.

2 Conclusion

L'exploration en profondeur de plusieurs varaintes du **Dynamic Time Wraping** nous a permis de vérifier la théorie et voir à quel point le Rcpp peut l'améliorer largement. Nous avons confirmé expérimentalement la complexité quadratique du DTW classique, ainsi qu el'apport majeur des contraintes locales telles que **la bande de Sakoe-Chibe**, qui réduisent la complexité à un ordre linéaire tout en préservant la qualité de l'alignement. Enfin, le **Segmental DTW** offre un compromis intéressant: il conserve la complexité théorique globale tout en améliorant lagestion de mémoire, la précision locale et le potentiel de prallélisation.