Projet d'Algorithmique : Classification des séries temporelles à l'aide du 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 deux amélioration de cet algorithme. Enfin, nous choisirons le meilleur algorithme pour la classification dans le cadre financier.

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.

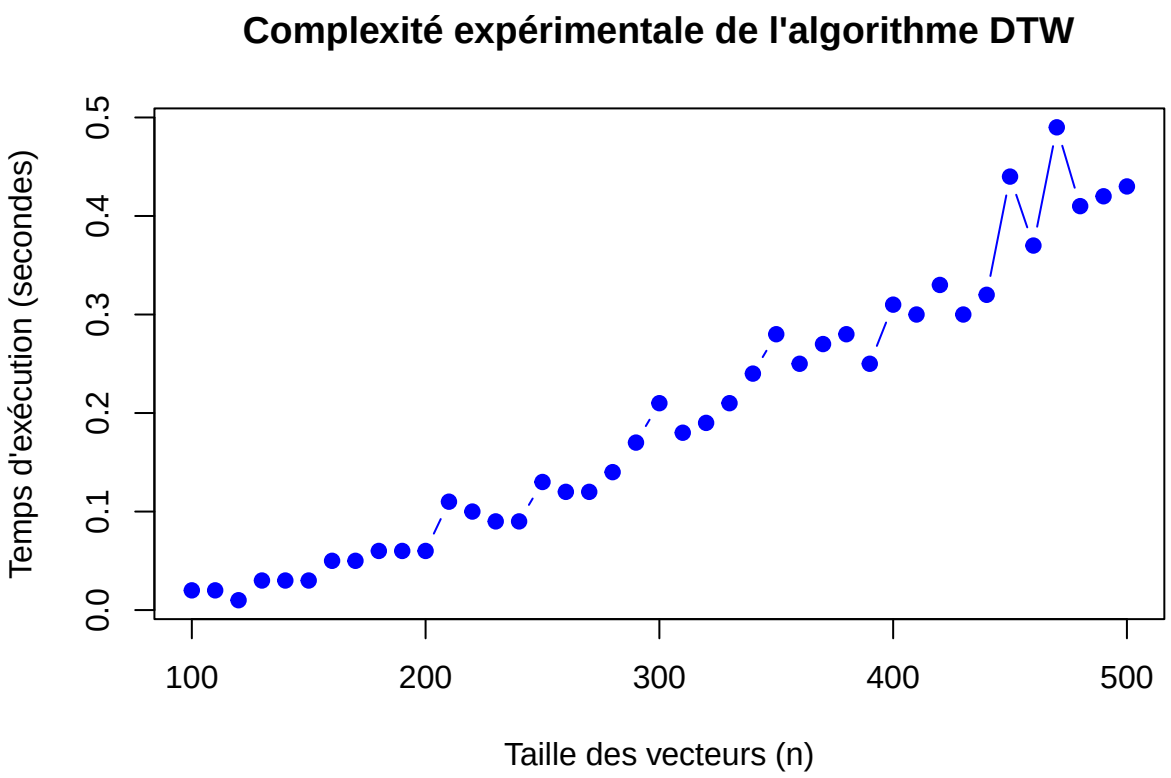
Pour analyser la complexité à partir des mesures expérimentales, nous traçons le graphique en échelle log-log. En prenant le logarithme de l'expression précédente, on obtient :

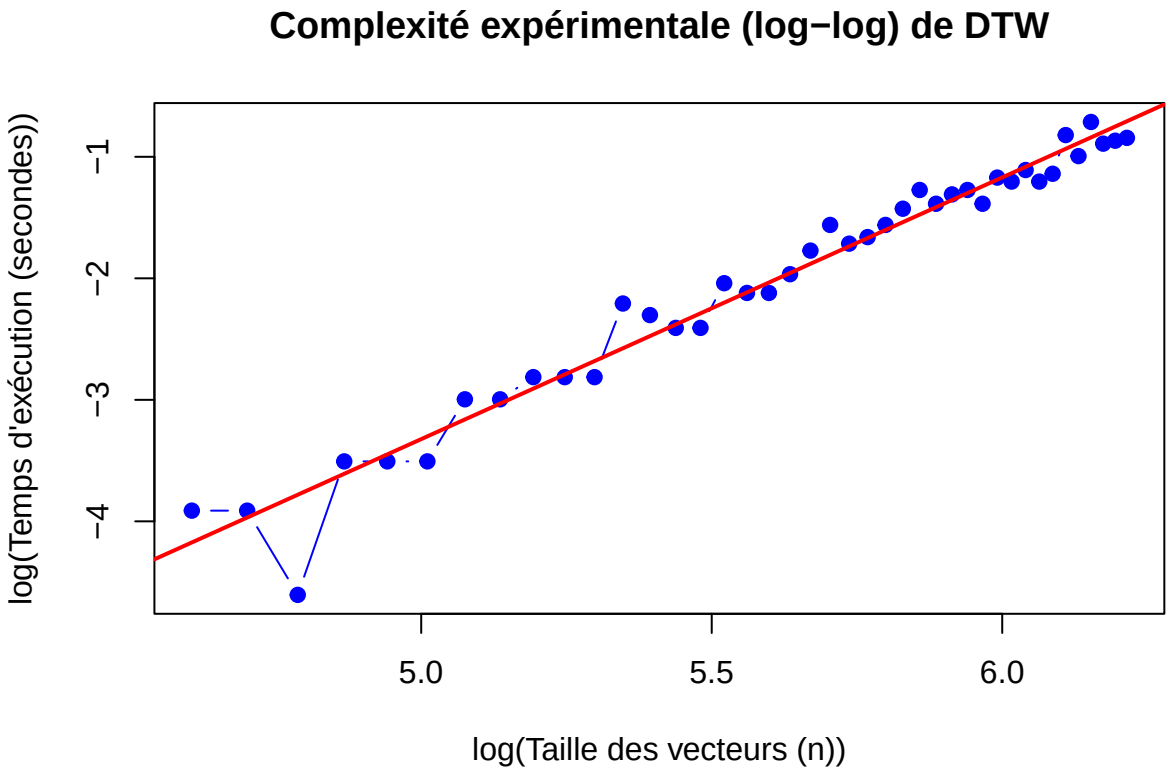
$$\log T(n) = \log C + 2 \log n.$$

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 = 100, fin = 500, pas = 10)
```


Complexité expérimentale de l'algorithme DTW
Temps d'exécution (secondes)
0.0
0.1
0.2
0.3
0.4
0.5
100
200
300
400
500
Taille des vecteurs (n)

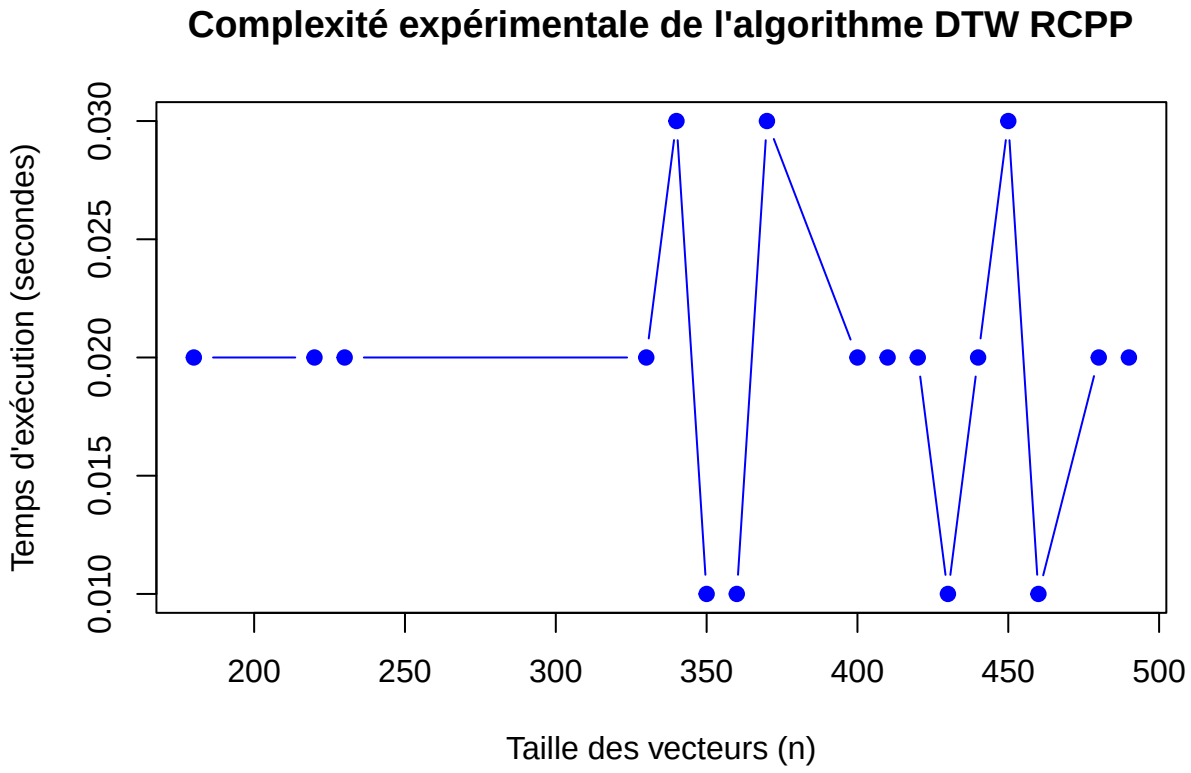

Complexité expérimentale (log−log) de DTW
log(Temps d'exécution (secondes))
−4
−3
−2
−1
5.0
5.5
6.0
log(Taille des vecteurs (n))

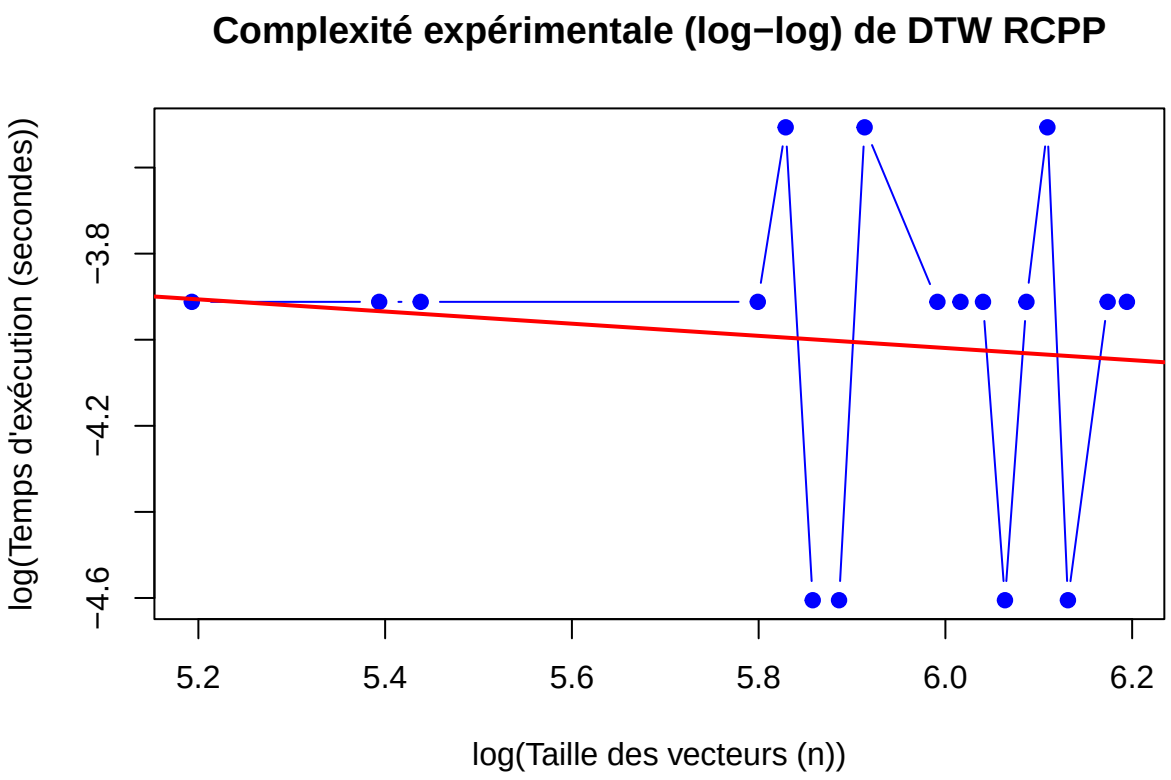
```
## Pente estimée sur le graphique log-log : 2.15
```

```
res$pente
```

```
## log(Taille)
##    2.153982
```

```
# C++
res <- simulation_dtw_rcpp(debut = 100, fin = 500, pas = 10)
```






```
## Pente estimée sur le graphique log-log : -0.14
```

```
res$pente
```

```
## log(Taille)
##  -0.1411961
```

Finalement, nous constatons que le temps d'exécution de l'algorithme en R et en C++ n'est pas du même ordre de grandeur: pour une série de taille 500, le programme en R prend 0,1 seconde, tandis que celui en C++ ne prend que 0,005 seconde.

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

Problématique : Les séries sont à des fréquences différentes. Evaluer analytiquement des similarités structurelles entre des séries.

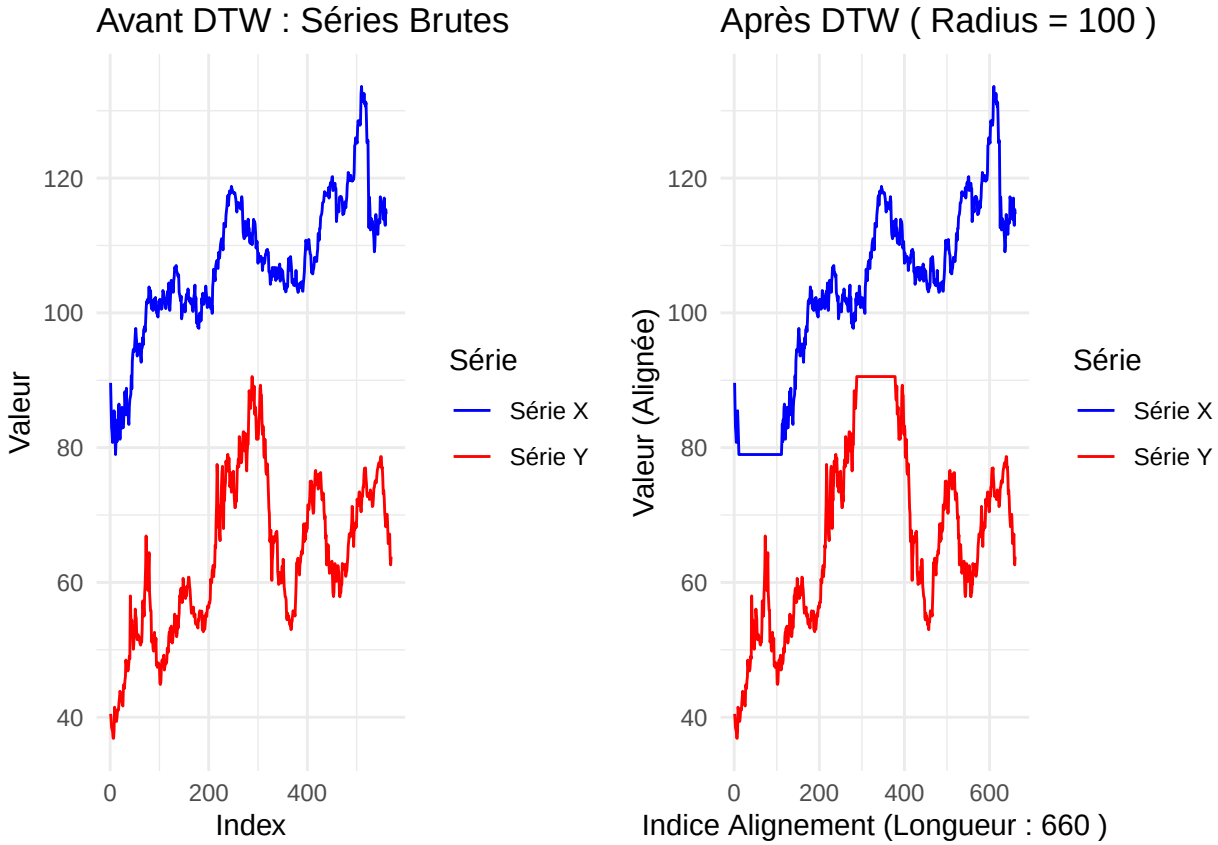
```
library(tidyr)
###  Charge les données depuis le package
path_stock <- system.file("extdata", "stock.csv", package = "ProjetM2Algo")
path_com <- system.file("extdata", "commodity.csv", package = "ProjetM2Algo")

df_stock <- read.csv(path_stock , sep=",")
df_com <- read.csv(path_com, sep =",")
```

```
df_com <- df_com %>%

  mutate(Close = as.numeric(Close)) %>%

  mutate(Close = ifelse(is.na(Close) | Close == 0.0000 | Close<1, NA, Close)) %>%

  fill(Close, .direction = "down") %>%
  fill(Close, .direction = "up")

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

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

```r
plot_dtw_comparison_simple(X,Y,radius=100 , dtw_sakoe_chiba_rcpp)
```

```
## Calcul DTW avec radius = 100 ...
```



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
)
```

```
library(gridExtra)
```

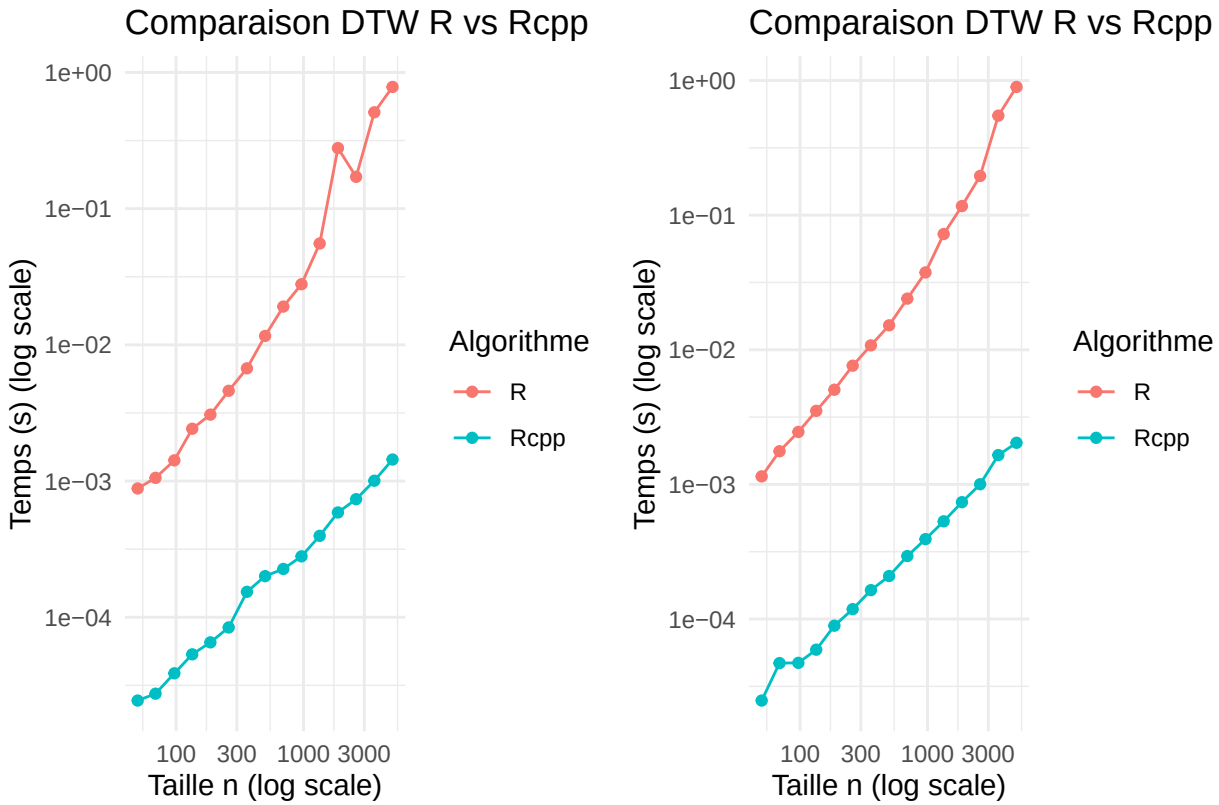
```
##
## Attaching package: 'gridExtra'
```

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

```
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=




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

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

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

```
##
## * Pente R (Radius 1) : ** 1.5291 **
```

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

```
##
## * Pente R (Radius 2) : ** 1.3996 **
```

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

```
##
## * Pente Rcpp (Radius 1) : ** 0.897 **
```