

# Algorithmique M2 Data Science

cours 3  
Programmation dynamique  
Dynamic programming

Vincent Runge



Laboratoire de  
Mathématiques  
et Modélisation  
d'Évry



- 1 Découpage optimal
- 2 La suite de Fibonacci
- 3 Problème du rendu de monnaie
- 4 Détection de ruptures pour les séries temporelles
- 5 Qu'est ce que la programmation dynamique ?
- 6 TP

## Section 1

# Découpage optimal

# Découper des barres

Soit une barre d'acier de longueur  $n$  cm et la grille tarifaire suivante:

longueur $i$ cm	1	2	3	4	5	6	7	8	9	10
prix $p_i$	1	5	8	9	10	17	17	20	24	30

Déterminer le **revenu maximal**  $r_n$  que l'on peut obtenir en découpant la barre

# Découper des barres

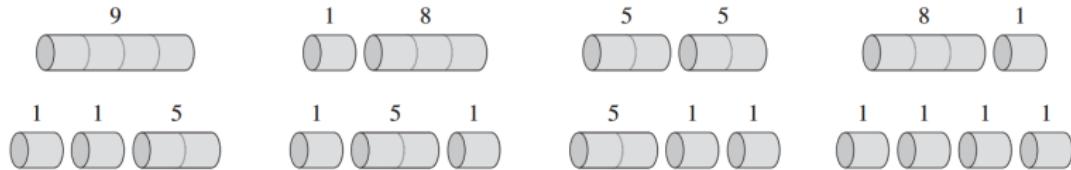
Soit une barre d'acier de longueur  $n$  cm et la grille tarifaire suivante:

longueur $i$ cm	1	2	3	4	5	6	7	8	9	10
prix $p_i$	1	5	8	9	10	17	17	20	24	30

Déterminer le **revenu maximal**  $r_n$  que l'on peut obtenir en découpant la barre

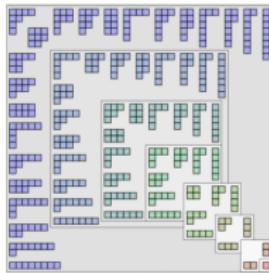
Pour une barre de longueur  $n$  :  $2^{n-1}$  découpages possibles!

Exemple avec  $n = 4$ :



# Découper des barres

Pour une barre de longueur  $n$  :  $2^{n-1}$  découpages possibles!



On peut **réduire ce nombre** de  $2^{n-1}$  aux nombres de **partitions noté p(n)**

Pour  $n = 4$  ce sera  $p(4) = \#\{1 + 1 + 1 + 1, 1 + 1 + 2, 1 + 3, 2 + 2, 4\}$

avec l'équivalent

$$p(n) \sim \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

découvert par [G. H. Hardy](#) et [Ramanujan](#) en 1918

# Découper des barres

longueur $i$ cm	1	2	3	4	5	6	7	8	9	10
prix $p_i$	1	5	8	9	10	17	17	20	24	30

Pour une découpe  $n = i_1 + i_2 + \dots + i_k$  le revenu est  $r = p_{i_1} + p_{i_2} + \dots + p_{i_k}$

On peut faire une recherche des premières solutions optimales:

- $r_1 = 1$  avec  $1 = 1$
- $r_2 = 5$  avec  $2 = 2$
- $r_3 = 8$  avec  $3 = 3$
- $r_4 = 10$  avec  $4 = 2 + 2$
- $r_5 = 13$  avec  $5 = 2 + 3$
- $r_6 = 17$  avec  $6 = 6$
- $r_7 = 18$  avec  $7 = 1 + 6$  ou  $7 = 2 + 2 + 3$
- $r_8 = 22$  avec  $8 = 2 + 6$
- $r_9 = 25$  avec  $9 = 3 + 6$
- $r_{10} = 30$  avec  $10 = 10$

# Relation de récurrence

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$

- $p_n$  : vente de la barre sans découpe
- $r_i + r_{n-i}$  : découpe de la barre en les morceaux  $i$  et  $n - i$

# Relation de récurrence

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$

- $p_n$  : vente de la barre sans découpe
- $r_i + r_{n-i}$  : découpe de la barre en les morceaux  $i$  et  $n - i$

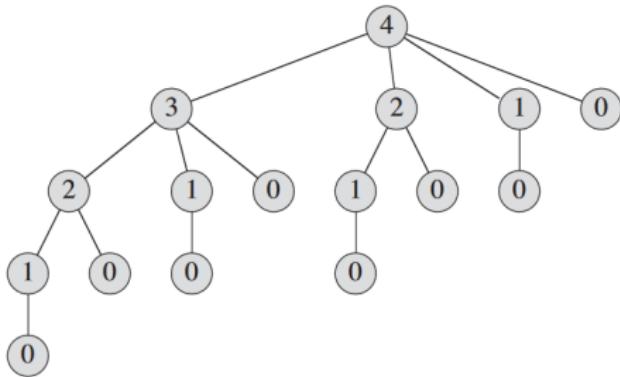
REMARQUE : pour trouver  $r_n$  on résout des problèmes du même type mais plus petits!

**Autre version :**

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

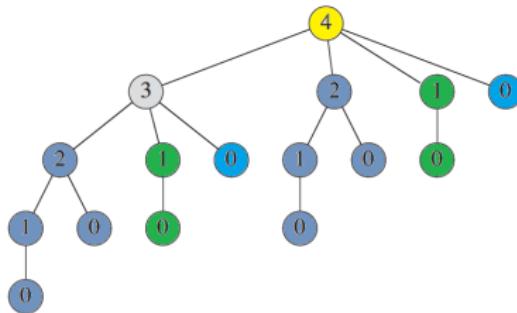
# Algorithme récursif naïf

```
CUT-ROD( $p, n$ )
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  return  $q$ 
```



Appels récursifs à cut-rod pour  $n = 4$

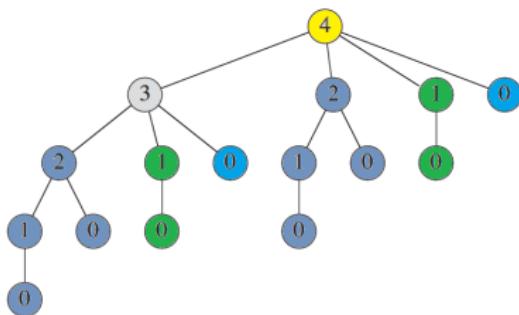
# Algorithme récursif naïf (analyse)



$NB(n)$  : nombre d'appels à cut-rod avec  $NB(0) = 1$

$$NB(n) = 1 + \sum_{j=0}^{n-1} NB(j)$$

# Algorithme récursif naïf (analyse)



$NB(n)$  : nombre d'appels à cut-rod avec  $NB(0) = 1$

$$NB(n) = 1 + \sum_{j=0}^{n-1} NB(j)$$

On peut montrer par récurrence que  $NB(n) = 2^n$

$$NB(0) = 2^0 \text{ et } NB(n) = 1 + \sum_{j=0}^{n-1} 2^j = 1 + \frac{2^n - 1}{2 - 1} = 2^n$$

# Algorithme récursif naïf (analyse)

=> la solution récursive est **mauvaise!** Il faut s'arranger pour ne résoudre les sous-problèmes qu'une seule fois (en sauvegardant leur solution)

- compromis temps/mémoire à faire
- on passe alors (souvent) d'une complexité en temps exponentielle à un temps polynomial.

# Méthode par mémoïsation (1)

MEMOIZED-CUT-ROD( $p, n$ )

```
1 let  $r[0..n]$  be a new array
2 for  $i = 0$  to  $n$ 
3    $r[i] = -\infty$ 
4 return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )
```

---

MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```
1 if  $r[n] \geq 0$ 
2   return  $r[n]$ 
3 if  $n == 0$ 
4    $q = 0$ 
5 else  $q = -\infty$ 
6 for  $i = 1$  to  $n$ 
7    $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r))$ 
8  $r[n] = q$ 
9 return  $q$ 
```

# Méthode par mémoïsation (1)

MEMOIZED-CUT-ROD( $p, n$ )

```

1 let  $r[0..n]$  be a new array
2 for  $i = 0$  to  $n$ 
3    $r[i] = -\infty$ 
4 return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```

---

MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```

1 if  $r[n] \geq 0$ 
2   return  $r[n]$ 
3 if  $n == 0$ 
4    $q = 0$ 
5 else  $q = -\infty$ 
6 for  $i = 1$  to  $n$ 
7    $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r))$ 
8  $r[n] = q$ 
9 return  $q$ 

```

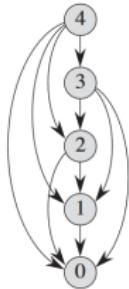
# Méthode ascendante (2)

**BOTTOM-UP-CUT-ROD( $p, n$ )**

```

1 let  $r[0..n]$  be a new array
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4      $q = -\infty$ 
5     for  $i = 1$  to  $j$ 
6          $q = \max(q, p[i] + r[j - i])$ 
7      $r[j] = q$ 
8 return  $r[n]$ 
```

- On parcourt l'ordre naturel des sous-problèmes
- Avec la double boucle, complexité en  $\Theta(n^2)$



# Méthode ascendante + backtracking

**EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )**

```

1 let  $r[0..n]$  and  $s[0..n]$  be new arrays
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4    $q = -\infty$ 
5   for  $i = 1$  to  $j$ 
6     if  $q < p[i] + r[j-i]$ 
7        $q = p[i] + r[j-i]$ 
8        $s[j] = i$ 
9    $r[j] = q$ 
10 return  $r$  and  $s$ 
```

**PRINT-CUT-ROD-SOLUTION( $p, n$ )**

```

1  $(r, s) = \text{EXTENDED-BOTTOM-UP-CUT-ROD}(p, n)$ 
2 while  $n > 0$ 
3   print  $s[n]$ 
4    $n = n - s[n]$ 
```

# Méthode ascendante + backtracking

**EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )**

```

1 let  $r[0..n]$  and  $s[0..n]$  be new arrays
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4    $q = -\infty$ 
5   for  $i = 1$  to  $j$ 
6     if  $q < p[i] + r[j-i]$ 
7        $q = p[i] + r[j-i]$ 
8        $s[j] = i$ 
9    $r[j] = q$ 
10 return  $r$  and  $s$ 
```

**PRINT-CUT-ROD-SOLUTION( $p, n$ )**

```

1  $(r, s) = \text{EXTENDED-BOTTOM-UP-CUT-ROD}(p, n)$ 
2 while  $n > 0$ 
3   print  $s[n]$ 
4    $n = n - s[n]$ 
```

## Section 2

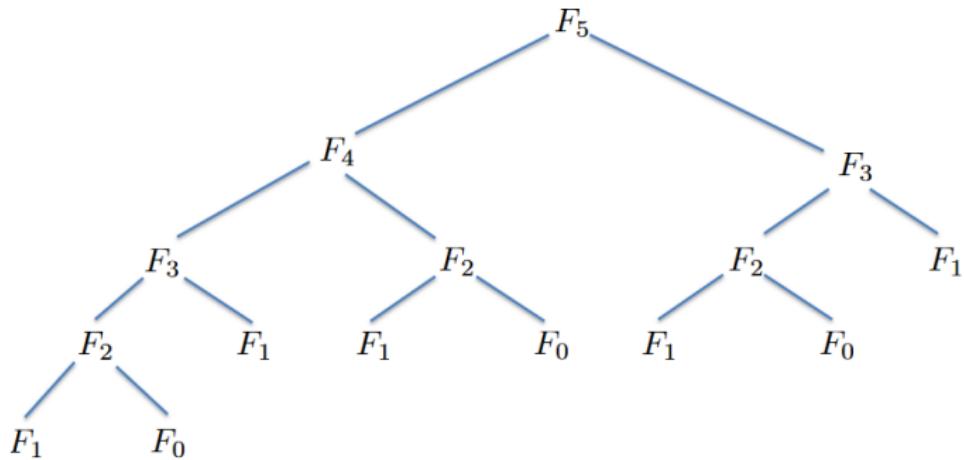
# La suite de Fibonacci

# La solution naïve

$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$  pour  $n \geq 2$

naive\_fibo( $n$ ):

```
if  $n = 0$ : return 0
else if  $n = 1$ : return 1
else: return naive_fibo( $n - 1$ ) + naive_fibo( $n - 2$ )
```



# La solution naïve

$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$  pour  $n \geq 2$

Complexité:

$$T(n) = T(n - 1) + T(n - 2) + c$$

$c$ : temps pour sommer deux nombres à  $n$  digits.

# La solution naïve

$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$  pour  $n \geq 2$

Complexité:

$$T(n) = T(n-1) + T(n-2) + c$$

$c$ : temps pour sommer deux nombres à  $n$  digits.

$$T(n) \geq 2T(n-2) + c$$

$$T(n) \geq 2^k T(n-2k) + c(2^{k-1} + 2^{k-2} + \cdots + 2 + 1) = \Omega(c2^{\frac{n}{2}})$$

# La solution naïve

$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$  pour  $n \geq 2$

Complexité:

$$T(n) = T(n-1) + T(n-2) + c$$

$c$ : temps pour sommer deux nombres à  $n$  digits.

$$T(n) \geq 2T(n-2) + c$$

$$T(n) \geq 2^k T(n-2k) + c(2^{k-1} + 2^{k-2} + \dots + 2 + 1) = \Omega(c2^{\frac{n}{2}})$$

Mais  $c = \Omega(n)$  car avec  $\phi = \frac{\sqrt{5}+1}{2}$ ,  $F_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}} \approx n \log_{10} \phi \approx \frac{n}{5}$

DONC :

$$T(n) = \Omega(n \times 2^{\frac{n}{2}})$$

# La solution mémoïsation

MEMFIBO( $n$ ):

    if  $n = 0$

        return 0

    else if  $n = 1$

        return 1

    else

        if  $F[n]$  is undefined

$F[n] \leftarrow \text{MEMFIBO}(n - 1) + \text{MEMFIBO}(n - 2)$

        return  $F[n]$

# La solution ascendante

ITERFIBO( $n$ ):

```
 $F[0] \leftarrow 0$ 
 $F[1] \leftarrow 1$ 
for  $i \leftarrow 2$  to  $n$ 
     $F[i] \leftarrow F[i - 1] + F[i - 2]$ 
return  $F[n]$ 
```

ITERFIBO2( $n$ ):

```
 $prev \leftarrow 1$ 
 $curr \leftarrow 0$ 
for  $i \leftarrow 1$  to  $n$ 
     $next \leftarrow curr + prev$ 
     $prev \leftarrow curr$ 
     $curr \leftarrow next$ 
return  $curr$ 
```

# Even faster !

On peut montrer que:

$$F_{2n-1} = F_{n-1}^2 + F_n^2$$

$$F_{2n} = F_n(2F_{n-1} + F_n)$$

```

⟨⟨Compute the pair  $F_{n-1}, F_n
FASTRECFIBO( $n$ ) :
    if  $n = 1$ 
        return 0, 1
     $m \leftarrow \lfloor n/2 \rfloor$ 
     $hprv, hcur \leftarrow \text{FASTRECFIBO}(m)$    ⟨⟨ $F_{m-1}, F_m$ ⟩⟩
     $prev \leftarrow hprv^2 + hcur^2$            ⟨⟨ $F_{2m-1}$ ⟩⟩
     $curr \leftarrow hcur \cdot (2 \cdot hprv + hcur)$    ⟨⟨ $F_{2m}$ ⟩⟩
     $next \leftarrow prev + curr$            ⟨⟨ $F_{2m+1}$ ⟩⟩
    if  $n$  is even
        return  $prev, curr$ 
    else
        return  $curr, next$$ 
```

# Even faster (analyse)

$$T(n) = T\left(\frac{n}{2}\right) + M(n)$$

$M(n)$  : temps pour multiplier 2 nombres à  $n$  bits

=> master theorem :  $T(n) = \Theta(M(n)) = \Theta(n \log(n))$

# Even faster (analyse)

$$T(n) = T\left(\frac{n}{2}\right) + M(n)$$

$M(n)$  : temps pour multiplier 2 nombres à  $n$  bits

=> master theorem :  $T(n) = \Theta(M(n)) = \Theta(n \log(n))$

En 2019, il a été montré que la multiplication de deux entiers de  $n$ -bits peut être réalisée en temps  $O(n \log(n))$ , voir la [référence](#)

## Section 3

# Problème du rendu de monnaie

# Problème du rendu de monnaie (#1)

lien wikipédia

- Soit un système de monnaie (pièces et billets) donné
  - Comment rendre une somme donnée de façon optimale, i.e. avec **un nombre minimal de pièces et billets?**
- ① Cas général → problème **NP complet** (en le nombre de pièce différentes)
  - ② Cas des **systèmes canoniques** → l'algorithme glouton est optimal



- système (1,3,4) :  $6 = 4+1+1$  (algorithme glouton)

$$6 = 3+3 \text{ (programmation dynamique)}$$

# Problème du rendu de monnaie (#2)

- L'**Euro** est un système canonique
- Pas la **livre sterling** avant 1971 !



Exercice : montrer pourquoi?

# Problème du rendu de monnaie (#3)

Soit le système de pièces  $S = \{c_1, \dots, c_n\}$ .

Problématique : nombre minimal de pièces pour faire la valeur  $v$

$$\text{Minimiser} \quad z = \sum_{j=1}^n x_j$$

sous contraintes

$$\sum_{j=1}^n x_j c_j = v \quad x_j \geq 0 \quad \text{entiers}$$

On peut montrer que le problème de la faisabilité (= existence d'une solution) est un problème NP-complet (cf **Lueker 1975 two np-complete problems in nonnegative integer programming**)

# Problème du rendu de monnaie (#4)

- **lien youtube** pour la solution en programmation dynamique (+ le backtracking bien expliqué!)
- Nombre minimal  $M_S(v)$  de pièces permettant de rendre la valeur  $v$  dans le système de pièces  $S = \{c_1, \dots, c_n\}$ .

## RécurSION de la programmation dynamique :

$$M_S(v) = 1 + \min_{1 \leq i \leq n} M_S(v - c_i)$$

- *Il n'existe pas, à ce jour, de caractérisation générale des systèmes canoniques, mais il existe une méthode efficace pour déterminer si un système donné est canonique.* (source wikipédia)

**article (2005)** A polynomial-time algorithm for the change-making problem

## Section 4

# Détection de ruptures pour les séries temporelles

# Détection de ruptures pour les séries temporelles

*Extrait d'une présentation à un séminaire scientifique*

# Multiple change-point problem

Data:  $y_{1:n} = (y_1, \dots, y_n)^T \in \mathbb{R}^n$

**Problem:** cluster the data in groups  $\{y_{\tau_i+1}, \dots, y_{\tau_{i+1}}\}$ ,  $i = 0, \dots, k$   
called **segments**

# Multiple change-point problem

Data:  $y_{1:n} = (y_1, \dots, y_n)^T \in \mathbb{R}^n$

**Problem:** cluster the data in groups  $\{y_{\tau_i+1}, \dots, y_{\tau_{i+1}}\}$ ,  $i = 0, \dots, k$   
called **segments**

Two unknowns:

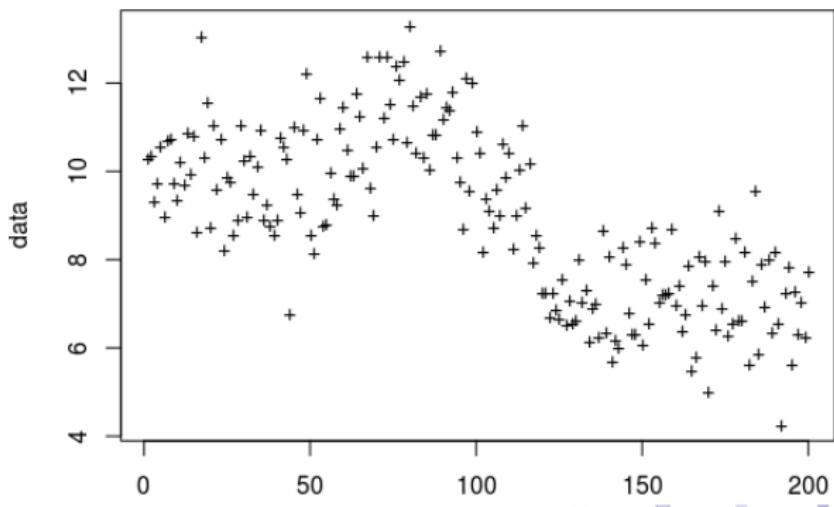
- The **number  $k$**  of change-points
- Their **location** ( $0 = \tau_0 < \tau_1 < \dots < \tau_k < \tau_{k+1} = n$ )

# Multiple change-point problem

Data:  $y_{1:n} = (y_1, \dots, y_n)^T \in \mathbb{R}^n$

Two unknowns:

- The **number  $k$**  of change-points
- Their **location** ( $0 = \tau_0 < \tau_1 < \dots < \tau_k < \tau_{k+1} = n$ )

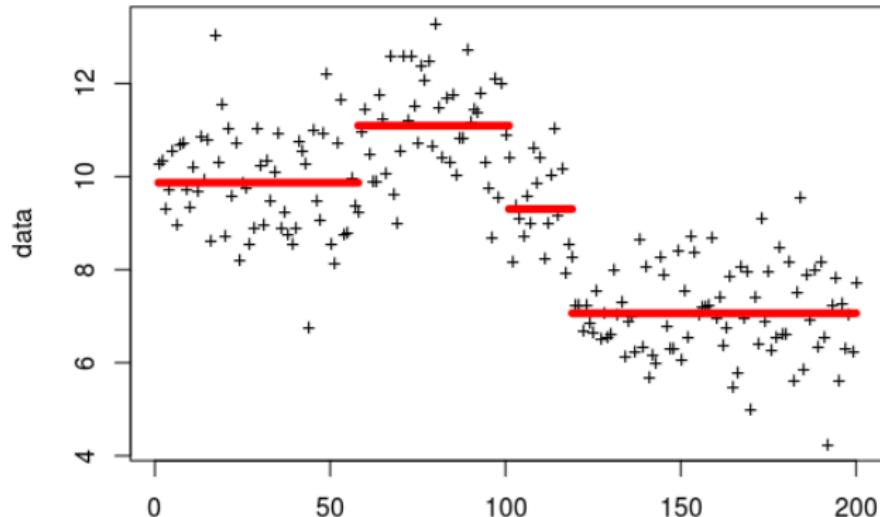


# Multiple change-point problem

Data:  $y_{1:n} = (y_1, \dots, y_n)^T \in \mathbb{R}^n$

Two unknowns:

- The **number  $k$**  of change-points
- Their **location** ( $0 = \tau_0 < \tau_1 < \dots < \tau_k < \tau_{k+1} = n$ )



# How to find the best segmentation?

## 3 standard steps

- (1) Choose a **modelisation** (goodness of fit with a cost function  $\mathcal{C}$ )
- (2) Choose a **statistic** criterion (model selection)
- (3) Choose an **algorithm**

# How to find the best segmentation?

## 3 standard steps

- (1) Choose a **modelisation** (goodness of fit with a cost function  $\mathcal{C}$ )
- (2) Choose a **statistic** criterion (model selection)
- (3) Choose an **algorithm**

## Parametric model

- $y_t$  are realizations of random variables  $Y_t$  i.i.d on each segment

$$Y_t \sim \mathcal{L}(\theta_i) \quad t \in \{\tau_i + 1, \dots, \tau_{i+1}\}, \quad i = 0, \dots, k$$

One-parameter law : Gauss  $\mathcal{G}(\theta)$ , Poisson  $\mathcal{P}(\theta)$ , Binomial  $\mathcal{B}(\theta)$

## Parametric model

- $y_t$  are realizations of random variables  $Y_t$  i.i.d on each segment

$$Y_t \sim \mathcal{L}(\theta_i) \quad t \in \{\tau_i + 1, \dots, \tau_{i+1}\}, \quad i = 0, \dots, k$$

One-parameter law : Gauss  $\mathcal{G}(\theta)$ , Poisson  $\mathcal{P}(\theta)$ , Binomial  $\mathcal{B}(\theta)$

- with cost on each segment :

$$\mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) = \min_{\theta} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \theta) = \min_{\theta} \sum_{t=\tau_i+1}^{\tau_{i+1}} \gamma(y_t, \theta)$$

and

$$\gamma(y_t, \theta) = \begin{cases} (y_t - \theta)^2 & \text{Gauss} \quad \theta \in \mathbb{R} \\ \theta - y_t \log(\theta) & \text{Poisson} \quad \theta > 0 \\ y_t \log(\theta) + (1 - y_t) \log(1 - \theta) & \text{Binomial} \quad 0 < \theta < 1 \end{cases}$$

# How to find the best segmentation?

## 3 standard steps

- (1) Choose a **modelisation** (goodness of fit with a cost function  $\mathcal{C}$ )
- (2) Choose a **statistic** criterion (model selection)
- (3) Choose an **algorithm**

# Statistic criterion ?

**Penalized cost by  $\beta > 0$**

$$Q_n(\beta) = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^k \{ \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) + \beta \} \right]$$

⇒ Choose the right  $\beta$

$$\beta = 2\hat{\sigma}^2 \log(n) \quad \text{Yao \& Au (1989)}$$

**Fixed model size =  $K$**

$$Q_n^K = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^K \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) \right]$$

⇒ Choose the right integer  $K$

$$\min_{K=1,\dots,M} \{ Q_n^K + \text{Penalty}(K) \}$$

Penalty = slope heuristic, mBIC,...

# Statistic criterion ?

**Penalized cost by  $\beta > 0$**

$$Q_n(\beta) = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^k \{ \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) + \beta \} \right]$$

⇒ Choose the right  $\beta$

$$\beta = 2\hat{\sigma}^2 \log(n) \quad \text{Yao \& Au (1989)}$$

**Fixed model size =  $K$**

$$Q_n^K = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^K \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) \right]$$

⇒ Choose the right integer  $K$

$$\min_{K=1,\dots,M} \{ Q_n^K + \text{Penalty}(K) \}$$

Penalty = slope heuristic, mBIC,...

## References :

- Guyon and Yao (1999)
- Baraud, Giraud, Huet et al. (2009)
- Lebarbier (2005)
- Zhang and Siegmund (2007)
- Birge & Massart (2007)
- ...

### 3 standard steps

- (1) Choose a **modelisation** (goodness of fit with a cost function  $\mathcal{C}$ )
- (2) Choose a **statistic** criterion (model selection)
- (3) Choose an **algorithm**

# Algorithm

**Penalized cost by  $\beta > 0$**

$$Q_n(\beta) = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^k \{ \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) + \beta \} \right]$$

**Fixed model size =  $K$**

$$Q_n^K = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^K \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) \right]$$

How to efficiently compute  $Q_n$ ?

There are  $2^{n-1}$  different segmentations  
⇒ Combinatorial explosion !

**Question:** How to get the best one ?

Find the global minimum? or accept easily computable approximate solution ?

⇒ battle : Exactness vs. Time complexity

# Change-point detection algorithms (quick review)

## Two main approaches

- Binary segmentation:

# Change-point detection algorithms (quick review)

## Two main approaches

- Binary segmentation:
  - ▶ Status : approximate
  - ▶ Time complexity :  $O(n \log(n))$

# Change-point detection algorithms (quick review)

## Two main approaches

- Binary segmentation:
  - ▶ Status : approximate
  - ▶ Time complexity :  $O(n \log(n))$
- Dynamic programming:

# Change-point detection algorithms (quick review)

## Two main approaches

- Binary segmentation:
  - ▶ Status : approximate
  - ▶ Time complexity :  $O(n \log(n))$
- Dynamic programming:
  - ▶ Status : exact
  - ▶ Time complexity :  $O(n^2)$

# Binary Segmentation

Scott and Knott (1974)

## Principle

- Iteratively apply a **single change-point method**
- Stop when a **condition** is met

# Binary Segmentation

Scott and Knott (1974)

## Principle

- Iteratively apply a **single change-point method**
- Stop when a **condition** is met

Example :  $y_{1:n}$  and  $B > 0$  given

- (1)  $\{a, b\} = \{1, \dots, n\}$
- (2) find  $\tau^*$  :

$$\tau^* = \operatorname{Argmin}_{\tau \in \{a, \dots, b-1\}} (\mathcal{C}(y_{a:\tau}) + \mathcal{C}(y_{\tau+1:b}))$$

- (3) if  $\mathcal{C}(y_{a:\tau^*}) + \mathcal{C}(y_{\tau^*+1:b}) + B < \mathcal{C}(y_{a:b})$   
then repeat (2) with  $\{a, b\} = \{a, \dots, \tau^*\}$   
and with  $\{a, b\} = \{\tau^* + 1, \dots, b\}$

# Dynamic programming

## References

- Bellman and Dreyfus (1962)
- Segment Neighbourhood method. Auger and Lawrence (1989)
- Optimal Partitioning. Yao (1984), Jackson et al. (2005)

# Dynamic programming

## References

- Bellman and Dreyfus (1962)
- Segment Neighbourhood method. Auger and Lawrence (1989)
- **Optimal Partitioning.** Yao (1984), Jackson et al. (2005)

# Our setting

1.  $\mathcal{C}(\cdot)$  cost function: **Gaussian cost**, constant variance  $\sigma^2$

$$\mathcal{C}(y_{u:v}) = \min_{\theta \in \mathbb{R}} \left( \sum_{j=u}^v (y_j - \theta)^2 \right)$$

2. **Penalty**  $\beta > 0$ : cost for adding a new segment ( $\beta = 2\hat{\sigma}^2 \ln(n)$ )

# Our setting

1.  $\mathcal{C}(\cdot)$  cost function: Gaussian cost, constant variance  $\sigma^2$

$$\mathcal{C}(y_{u:v}) = \min_{\theta \in \mathbb{R}} \left( \sum_{j=u}^v (y_j - \theta)^2 \right)$$

2. Penalty  $\beta > 0$ : cost for adding a new segment ( $\beta = 2\hat{\sigma}^2 \ln(n)$ )

**Problem:** Find  $Q_n = \min_{\bar{\tau} \in S_n} \left[ \sum_{i=0}^k \{\mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) + \beta\} \right]$

$$S_n = \{\bar{\tau} \in \mathbb{N}^{k+2}, 0 = \tau_0 < \tau_1 < \dots < \tau_k < \tau_{k+1} = n\}$$

# Dynamic programming

## 3. Optimal Partitioning

### Principle

- (1) Solve sub-problems: find **the last change-point** in  $y_{1:t}$ ,  $t = 2, \dots, n$ .
- (2) Backtrack a global solution

# Dynamic programming

## 3. Optimal Partitioning

### Principle

- (1) Solve sub-problems: find **the last change-point** in  $y_{1:t}$ ,  $t = 2, \dots, n$ .
- (2) Backtrack a global solution

$$Q_t = \min_{\bar{\tau} \in S_t} \left[ \sum_{i=0}^k \{C(y_{\tau_i+1:\tau_{i+1}}) + \beta\} \right]$$

# Dynamic programming

## 3. Optimal Partitioning

### Principle

- (1) Solve sub-problems: find **the last change-point** in  $y_{1:t}$ ,  $t = 2, \dots, n$ .
- (2) Backtrack a global solution

$$Q_t = \min_{\bar{\tau} \in S_t} \left[ \sum_{i=0}^k \{C(y_{\tau_i+1:\tau_{i+1}}) + \beta\} \right]$$

$$Q_t = \min_{\bar{\tau} \in S_t} \left[ \sum_{i=0}^{k-1} \{C(y_{\tau_i+1:\tau_{i+1}}) + \beta\} + C(y_{\tau_k+1:\tau_{k+1}}) + \beta \right]$$

# Dynamic programming

## 3. Optimal Partitioning

### Principle

- (1) Solve sub-problems: find **the last change-point** in  $y_{1:t}$ ,  $t = 2, \dots, n$ .
- (2) Backtrack a global solution

$$Q_t = \min_{\bar{\tau} \in S_t} \left[ \sum_{i=0}^k \{ \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) + \beta \} \right]$$

$$Q_t = \min_{\bar{\tau} \in S_t} \left[ \sum_{i=0}^{k-1} \{ \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}) + \beta \} + \mathcal{C}(y_{\tau_k+1:\tau_{k+1}}) + \beta \right]$$

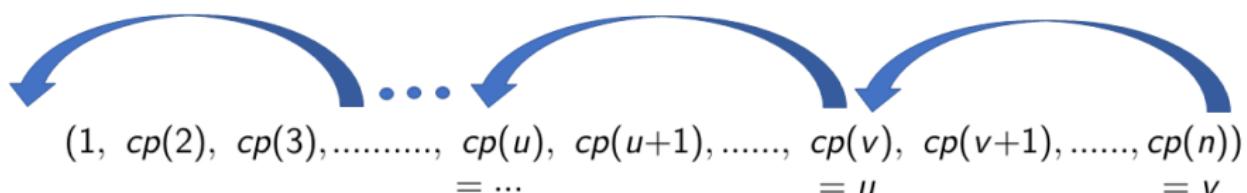
$$Q_t = \min_{s \in \{0, \dots, t-1\}} \left( Q_s + \mathcal{C}(y_{s+1:t}) + \beta \right)$$

# Dynamic programming

## 3. Optimal Partitioning

Algorithm  $\Rightarrow O(n^2)$  Time complexity

- 1:  $Q_0 = 0$
- 2: **for**  $t = 1$  to  $n$  **do**
- 3:    $Q_t = \min_{s \in \{0, \dots, t-1\}} \{Q_s + \mathcal{C}(y_{s+1:t}) + \beta\}$
- 4:    $cp(t) = \operatorname{Argmin}_{s \in \{0, \dots, t-1\}} \{Q_s + \mathcal{C}(y_{s+1:t}) + \beta\}$
- 5: **end for**



# Pruning methods

Solve

$$Q_t = \min_{s \in \{0, \dots, t-1\}} \{Q_s + \mathcal{C}(y_{s+1:t}) + \beta\} = \min_{s \in \{0, \dots, t-1\}} \{Q_t^s\}$$

for all  $t = 1, \dots, n$ .

## Principle

Find conditions to remove integers from set  $\{0, \dots, t-1\}$

$$\left[ \begin{array}{lllllll} t=1 : & Q_1^0 \\ t=2 : & Q_2^0 & Q_2^1 \\ t=3 : & Q_3^0 & Q_3^1 & Q_3^2 \\ t=4 : & Q_4^0 & Q_4^1 & Q_4^2 & Q_4^3 \\ t=5 : & Q_5^0 & Q_5^1 & Q_5^2 & Q_5^3 & Q_5^4 \\ t=6 : & Q_6^0 & Q_6^1 & Q_6^2 & Q_6^3 & Q_6^4 & Q_6^5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array} \right]$$

# Pruning methods

Solve

$$Q_t = \min_{s \in \{0, \dots, t-1\}} \{ Q_s + \mathcal{C}(y_{s+1:t}) + \beta \} = \min_{s \in \{0, \dots, t-1\}} \{ Q_t^s \}$$

for all  $t = 1, \dots, n$ .

## Principle

Find conditions to remove integers from set  $\{0, \dots, t-1\}$

$t = 1 :$	$Q_1^0$								
$t = 2 :$	$Q_2^0$	$Q_2^1$							
$t = 3 :$	$\times$	$Q_3^1$	$Q_3^2$						
$t = 4 :$	$\times$	$Q_4^1$	$Q_4^2$	$Q_4^3$					
$t = 5 :$	$\times$	$Q_5^1$	$Q_5^2$	$\times$	$Q_5^4$				
$t = 6 :$	$\times$	$\times$	$Q_6^2$	$\times$	$Q_6^4$	$Q_6^5$			
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	

# Pruning methods

Solve

$$Q_t = \min_{s \in \{0, \dots, t-1\}} \{Q_s + \mathcal{C}(y_{s+1:t}) + \beta\} = \min_{s \in \{0, \dots, t-1\}} \{\textcolor{orange}{Q}_t^s\}$$

for all  $t = 1, \dots, n$ .

## Principle

Find conditions to remove integers from set  $\{0, \dots, t-1\}$

With pruning :

$$Q_t = \min_{s \in P_t} \{Q_s + \mathcal{C}(y_{s+1:t}) + \beta\}$$

with  $P_t \subset \{0, \dots, t-1\}$  and  $P_{t+1} \subset P_t \cup \{t\}$ .

# PELT

Inequality based pruning (Killick et al.(2012))

$$Q_t = \min_{s \in P_t} \{ Q_s + \mathcal{C}(y_{(s+1):t}) + \beta \}$$

with  $P_t \subset \{0, \dots, t-1\}$  and  $P_{t+1} \subset P_t \cup \{t\}$ .

If

$$Q_t \leq Q_s + \mathcal{C}(y_{(s+1):t})$$

we prune position  $s$  in  $P_u$  for all  $u > t$

Proof.

$$\begin{aligned} Q_t + \mathcal{C}(y_{(t+1):u}) + \beta &\leq Q_s + \mathcal{C}(y_{(s+1):t}) + \mathcal{C}(y_{(t+1):u}) + \beta \\ &\leq Q_s + \mathcal{C}(y_{(s+1):u}) + \beta \end{aligned}$$

□

⇒ Time complexity :  $O(n)$  if a lot of change-points, otherwise  $O(n^2)$

*fin de l'extrait*

## Section 5

**Qu'est ce que la programmation dynamique ?**

# Qu'est ce que la programmation dynamique ?

- ① On découpe un problème en des **sous-problèmes imbriqués**. Contrairement au "diviser pour régner" qui découpe les données en  $k$  problèmes "indépendants"
- ② La solution optimale utilise l'**optimalité des sous-problèmes**
- ③ On reconstruit la solution globale en parcourant dans l'autre sens les problèmes résolus (**backtracking step**)

# Citation de Bellman

lu sur wikipédia:

Bellman explique l'idée du terme programmation dynamique dans son autobiographie, *Eye of the Hurricane: An Autobiography*. Il dit :

« I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities. »

## Section 6

TP

# TP

Au programme de la semaine prochaine !