Chaines de Markov et modélisation

Kamil Stos

N° candidat: 54417

Juin 2022

FIG. 1 – *Andreï Markov*

FIG. 2 – *Paul Ehrenfest*

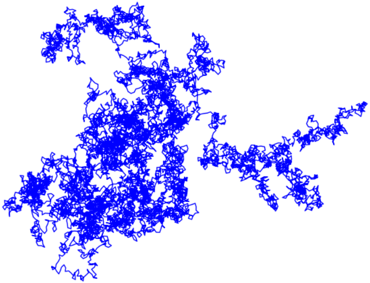


FIG. 3 – *Mouvement Brownien*

Qu'est ce qu'une chaine de Markov ?

- Espace probabilisé $(\Omega,\mathcal{F}, p)$
- $\{X_n\}_{n\geq 0}$ une suite de v.a. à valeur dans un ensemble $Q = \{i_1,...,i_n\}$

Propriété de Markov

$\forall (i_0,...,i_n) \in Q^{n+1}$:

$$P(X_{n+1} = i_{n+1} | X_n = i_n,...,X_0 = i_0) = P(X_{n+1} = i_{n+1} | X_n = i_n)$$

Exemple

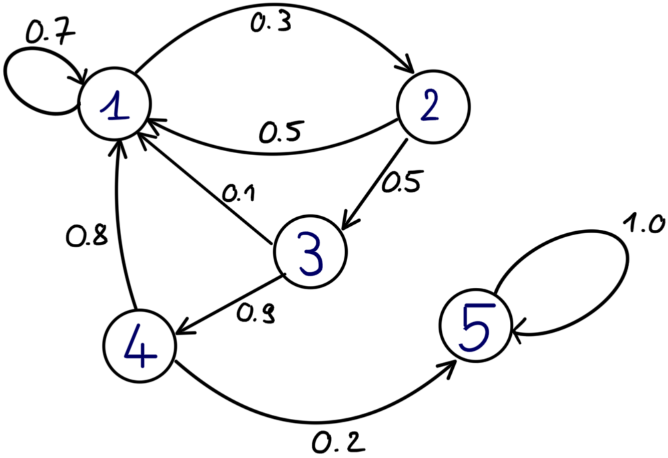
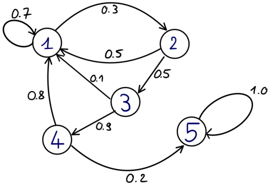


Fig. 4 – *Exemple d'une chaine de Markov : point de vue automate*



Matrice de transition

$$\forall (i,j) \in [\![1,n]\!]^2 : \qquad A_{i,j} = P(X_{n+1} = i_{n+1} | X_n = i_n)$$

Matrice de transition de l'exemple précédent

$$A = \begin{pmatrix} 0.7 & 0.3 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.1 & 0.9 & 0 \\ 0.8 & 0 & 0 & 0 & 0.2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Utilité

Position initiale: $\pi = \Big(P(X_0 = i_1), P(X_0 = i_2), \quad ..., P(X_0 = i_n) \Big)$

Après p itérations: $\pi \times A^p = \Big(P(X_p = i_1), P(X_p = i_2), ... , P(X_p = i_n) \Big)$

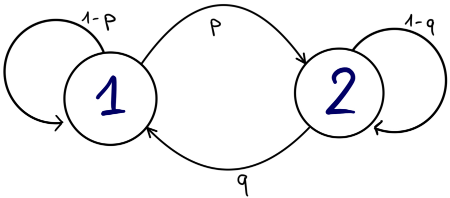
Position initiale: $\pi = \Big(P(X_0 = i_1),\ P(X_0 = i_2),\quad ...,\ P(X_0 = i_n) \Big)$

Après p itérations: $\pi \times A^p = \Big(P(X_p = i_1),\ P(X_p = i_2), ...\ ,\ P(X_p = i_n) \Big)$

- Est ce que le système converge vers une distribution stationnaire ?
- Si oui quelle est la configuration finale ?

Système à deux états

- $Q = \{u_0, u_1\}$
- $\exists p, q \in]0,1[, \quad A = \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix}$



$$A = \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix}$$

$\chi_A = det(XI_2 - A) = X^2 + (p+q-2)X + (1-p-q)$ est scincé à racines simples dans $\mathbb{R}$.

A possède deux valeurs propres 1 et 1 - p - q: $A \simeq \begin{pmatrix} 1 & 0 \\ 0 & (1-p-q) \end{pmatrix}$

et alors on a $\boxed{\lim_{n\to+\infty} A^n = \frac{1}{(p+q)} \begin{pmatrix} q & p \\ q & p \end{pmatrix}}$

Applications simples

Probabilité d'infection:

$$\frac{\color{blue}{moyenne\ de\ cas\ actifs}}{population}$$

Probabilité de guérison:

$$\frac{\color{red}{Morts\ + COVID\ long}}{nombre\ de\ cas}$$

Applications

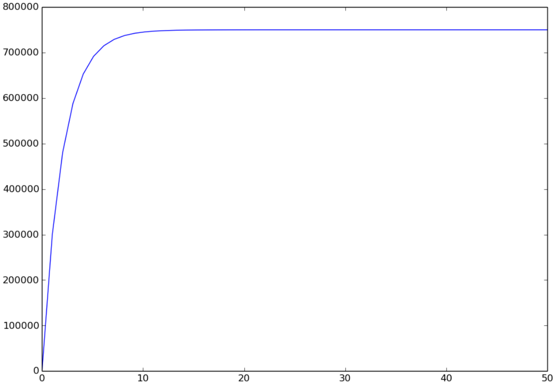


Fig. 5 – *Modèle à deux états*

Amélioration

Même démarche mais plus d'états:

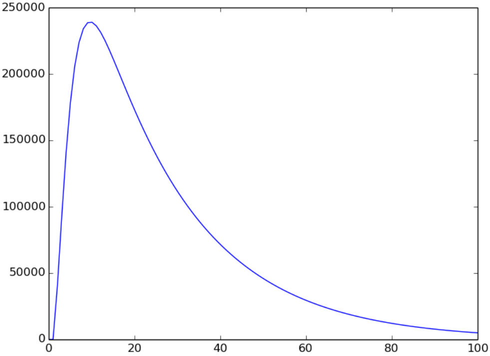


FIG. 6 – *Nombre d'infectés en France (gouvernement.fr)*

Jeux d'argent, probabilité de ruine

- Deux joueurs à position symétrique
-
-

Jeux d'argent, probabilité de ruine

- Deux joueurs à position symétrique
- N pièces : $Q = [\![0; 2N]\!]$
-

Jeux d'argent, probabilité de ruine

- Deux joueurs à position symétrique
- N pièces : $Q = [\![0; 2N]\!]$
- X_n : argent de A à la n^{ieme} partie

Jeux d'argent, probabilité de ruine

- Deux joueurs à position symétrique
- N pièces : $\quad Q = [\![0; 2N]\!]$
- X_n : argent de A à la n^{ieme} partie

Par exemple pour N $= 2$,

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & \\ 1-p & 0 & p & 0 & \\ 0 & 1-p & 0 & p & \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Ruine du joueur

$$On\ note\ u_i = P\Big(\exists t \in \mathbb{N} : X_t = 0 | X_0 = i\Big)$$

puis

$$u_i = pu_{i+1} + (1-p)u_{i-1} = p\ u_i + (1-p)\ u_i$$

d'où

$$(u_{i+1} - u_i) = \frac{1-p}{p}(u_i - u_{i-1})$$

$$u_i = \sum_{k=0}^{i-1} \left(\frac{1-p}{p}\right)^k (u_1 - u_0) + u_0$$

- Si $p = \frac{1}{2}$:
  $$\boxed{u_i = 1 - \frac{i}{N}}$$
- Sinon :
  $$\boxed{u_i = 1 - \frac{1-(\frac{1-p}{p})^i}{1-(\frac{1-p}{p})^N}}$$

Contre un casino: $N \longrightarrow +\infty$ et

- Si $p \leq \frac{1}{2}$: $u_i = 1$

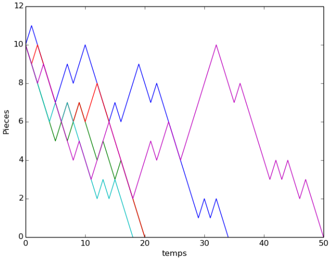


FIG. 7 – *cas* $p = 0.3$

- Sinon, $u_i = (\frac{1-p}{p})^i$

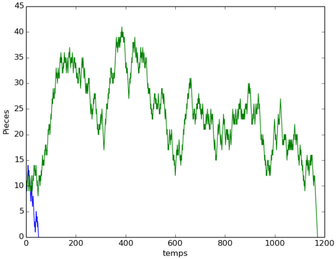


FIG. 8 – *cas* $p = 0.5001$

Espérance du temps d'une partie

On note T_i la variable aléatoire t.q.

$$(T_i = n) = \left(X_n = 0 \text{ ou } 2N \text{ et } \forall 0 < k < n : X_k \neq 0 \text{ ou } N | X_0 = i \right)$$

avec $v_i = \mathbb{E}(T_i)$:

$$\forall i \in [\![1; N-1]\!], \quad v_i = p v_{i+1} + (1-p) v_{i-1} + 1$$

Par la même méthode:

$$v_{i+1} = \frac{i}{2p-1} + (v_1 - \frac{1}{2p-1}) \times \frac{1-(\frac{1-p}{p})^i}{1-(\frac{1-p}{p})^N}$$

- $p = \frac{1}{2}$:

$$\forall i \in [\![1; N-1]\!], \quad v_i = \frac{1}{2} v_{i+1} + \frac{1}{2} v_{i-1} + 1$$

puis $\boxed{v_k = k(N-k))}$

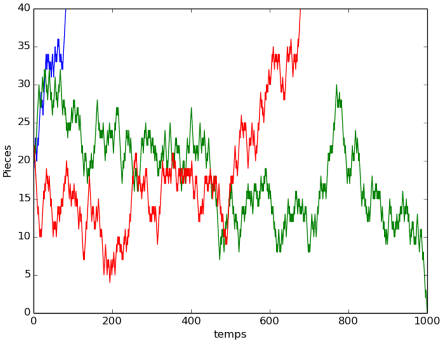


FIG. 9 – *cas* $p = 0.5$

Distribution des temps d'arrivées

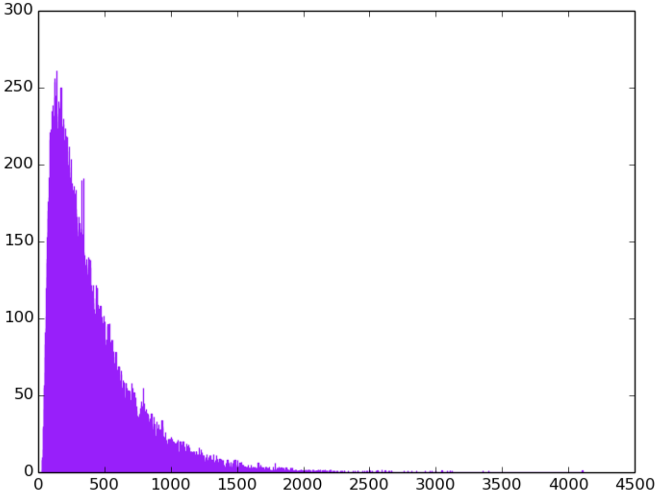


Fig. 10 – *Distribution des temps d'arrivée pour* $p = 0.5$, $N = 20$

Distribution des temps d'arrivées

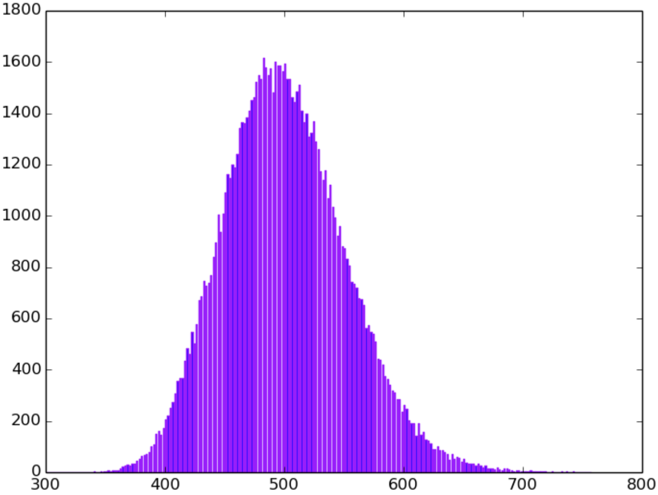


FIG. 11 – *Distribution des temps d'arrivée pour p = 0.3, N= 200*

Distribution des temps d'arrivées

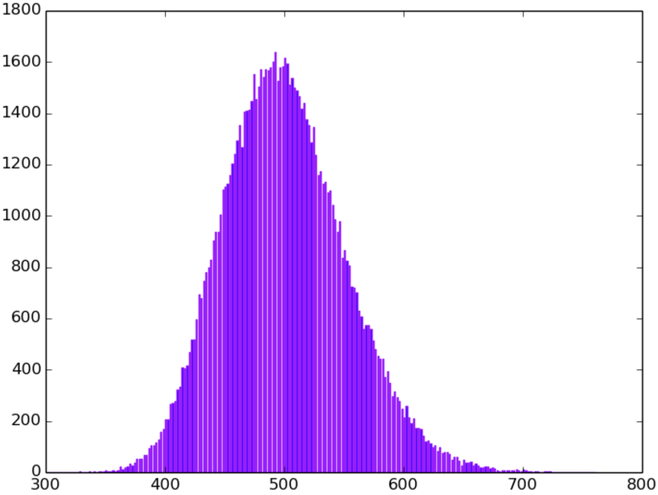


Fig. 12 – *Distribution des temps d'arrivée pour p = 0.7, N= 200*

Classification d'états

Etat récurrent

i est **récurrent** si:

$$f_{ii} := P(\text{Retour à i } |X_0 = i) = 1$$

Il est **transient** dans le cas contraire.

Etat récurrent positif

i est dit **récurrent positif** s'il est récurrent et que

$$\mu_i = E\left(min\{n \in \mathbb{N}^*, X_n = i\} \quad | \quad i_0 = i \right) \in \mathbb{R}^*$$

Il est dit **récurrent nul** si $\mu_i = +\infty$

Etat périodique

La **période** d'un état est:

$$d(i) = \begin{cases} pgcd\left(\{n \geq 1, \quad A_{ii}^{(n)} > 0\}\right) & \text{si cet ensemble est non vide} \\ 0 \quad sinon \end{cases}$$

Si d(i) = 0 ou d(i) = 1, l'état est **apériodique**

- Si $A_{ii}^{n} > 0$ *APCR*, i est **apériodique**.
- Si i **apériodique**:
  Quand $A_{ii}^{n} > 0$ et $A_{ii}^{m} > 0$, alors $A_{ii}^{n+m} > 0$

$$\exists (n_1,...,n_k) \quad : \quad \forall i \in [\![1,k]\!], \quad A_{i,i}^{n_k} > 0$$

$$n_1\mathbb{Z} + ... + n_k\mathbb{Z} = \mathbb{Z}$$

puis

$$\exists a \in \mathbb{N} \quad : \quad n_1\mathbb{N} + ... + n_k\mathbb{N} = \quad [a; \infty[$$

d'où $\underline{A_{ii}^{n} > 0}$ *à partir du rang a.*

Etat ergodique

i est **ergodique** s'il est récurrent positif et apériodique.

Exemple

On considère une situation modélisée par la matrice suivante
$P = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$.
Alors on voit que l'état 1 est récurrent:

$$Pr(\text{Non retour à } 1 | i_0 = 1) = \prod_{n=1}^{+\infty} \quad \frac{1}{2} = 0$$

Il est récurrent positif:

$$E(min\{n \in \mathbb{N}^*, X_n = 1\} \quad | \quad X_0 = 1) = \sum_{k=1}^{+\infty} k \times (\frac{1}{2})^k = 2$$

Il est apériodique : $\forall n \in \mathbb{N}^*, \quad P_{ii}^n > 0$
C'est finalement un état **ergodique**.

Théroème principal

Théorème ergodique

Pour j un état **ergodique**:

$$\lim_{n \to +\infty} P_{ij}^{(n)} = f_{ij} \times \frac{1}{\mu_j}$$

avec

$$\mu_i = E\left(min\{n \in \mathbb{N}^*, X_n = i\} \quad | \quad i_0 = i \right) \in \mathbb{R}^*$$

et

$$f_{ii} = Pr(\text{Visite en j } | X_0 = i)$$

Démonstration

- On a pour $n \in \mathbb{N}^*$
$$A_{i,j}^{(n)} = \sum_{k=1}^{n} f_{ij}^{(k)} A_{j,j}^{(n-k)}$$
- $\sum_{k \geq 1} f_{ij}^{(k)}$ converge.

Si $\lim_{n\to\infty} \delta_{\{k \leq n\}} A_{j,j}^{(n-k)} = \lim_{n\to\infty} A_{j,j}^{(n)} = \frac{1}{\mu_i}$

Alors
$$A_{i,j}^{(n)} = \sum_{k=1}^{n} f_{ij}^{(k)} A_{j,j}^{(n-k)} = \sum_{k=0}^{+\infty} f_{ij}^{(k)} A_{j,j}^{(n-k)} \delta_{\{k \leq n\}}$$

Lemme

Si l'on a $\lim\limits_{n\to\infty} b_{n,k} = b$ pour tout k avec $0 \leq \ b_{n,k} \leq 1$ et $a_k \geq 0$, alors

$$\lim_{n\to\infty} \sum_{k=1}^{+\infty} a_k b_{n,k} = b \sum_{k=1}^{+\infty} a_k$$

Par le lemme:

$$\sum_{k=0}^{+\infty} f_{ij}^{(k)} A_{j,j}^{(n-k)} \delta_{\{k \leq n\}} \quad \underset{n \longrightarrow +\infty}{\longrightarrow} \quad (\sum_{k=0}^{+\infty} f_{ij}^{(k)}) \times \lim_{n \to \infty} A_{j,j}^{(n-k)} = \frac{f_{i,j}}{\mu_i}$$

Montrons maintenant que

$$\lim_{n \to \infty} A_{j,j}^{(n)} = \frac{1}{\mu_i}$$

On appelle « équation de renouvellement » :

$$\sum_{k\geq 1} \delta_{\{k\leq n\}} \; A_{j,j}^{(n-k)} \times (\sum_{m\geq k} f_{j,j}^{(m)}) = 1$$

Soit ϕ une *extractrice* telle que

$$A_{j,j}^{\phi(n)} \underset{n\longrightarrow\infty}{\longrightarrow} \quad \ell$$

Montrons que $\ell = \frac{1}{\mu_j}$

Prenons ψ une *extractrice* telle que $A_{j,j}^{\psi(n)} \underset{n \longrightarrow \infty}{\longrightarrow} \ell'$.

$$A_{j,j}^{\psi(n)} = \sum_{i \in Q} A_{j,i}^{(k)} \; A_{i,j}^{(n-k)} = \sum_{i \in Q} A_{j,i}^{(k)} \; (A_{i,j}^{(n-k)} - A_{j,j}^{(n-k)}) + \sum_{i \in Q} A_{j,i}^{(k)} \; A_{j,j}^{(n-k)}$$

On montre que pour i **communiquant avec j**,

$$A_{i,j}^{(n)} - A_{j,j}^{(n)} \underset{n \longrightarrow \infty}{\longrightarrow} 0$$

Par le lemme;

$$\ell = \ell' \; \sum_{i \in Q} A_{j,i}(k) = \ell'$$

i.e. $\ell = \ell'$

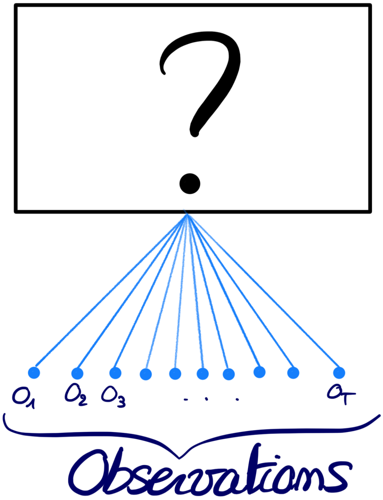
On applique alors l'**équation de renouvellement** aux instants $\phi(n)$ pour avoir

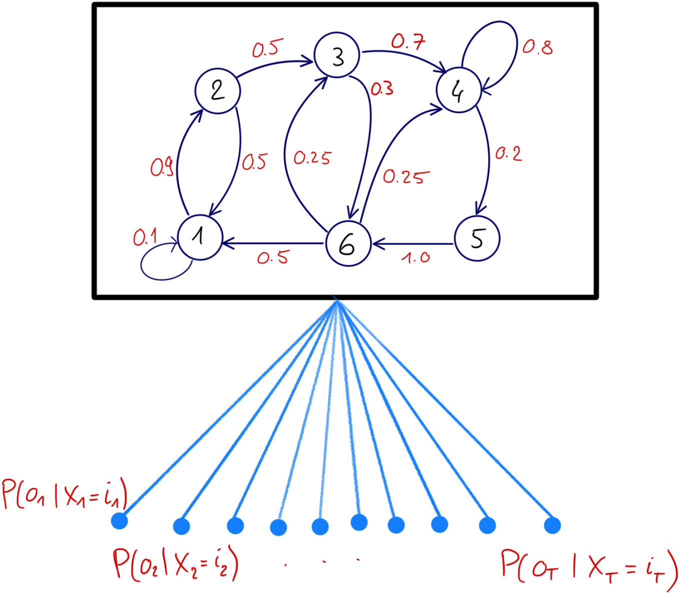
$$\ell \times \sum_{k \geq 1} \sum_{m \geq k} \ f_{j,j}^{(k)} = \ell \times \sum_{k \geq 1} m \ f_{j,j}^{(k)} = \ell \times \mu_j = 1$$

d'où

$$\ell = \frac{1}{\mu_j}$$

Qu'est-ce qu'une chaine de Markov Cachée





1
2
3
4
5
6
0.1
0.9
0.5
0.5
0.7
0.8
0.3
0.25
0.25
0.2
0.5
1.0
P(o1 | X1 = i1)
P(o2 | X2 = i2)
. . .
P(oT | XT = iT)

HMM: Hidden Markov Model

- $(X_n)_{n \in \mathbb{N}}$ une chaine de Markov
- A sa matrice de transition
- $\mathcal{O}$ un ensemble d'observables
- $(vsm_i(\mu))_{i \ \in Q}$ pour tout $\mu \in \mathcal{O}$

Utilité du modèle

- Etant donné un modèle $\theta = \left(A, vsm\right)$, calculer $P(O|\theta)$.
  ALGORITHME **FORWARD**

- Etant donné θ et O, trouver $(q_1, ..., q_n)$ optimal.
  ALGORITHME DE **VITERBI**

- Déterminer θ à partir d'une observation.
  ALGORITHME DE **BAUM-WELCH**

Algorithme forward

Idée naïve

$$P(O = o_1, \ldots, o_T) = \sum_{(q_1, \ldots, q_T) \in Q^T} \prod_{t=1}^{T} P(o_t | X_t = q_t)$$

$$= \sum_{(q_1, \ldots, q_T) \in Q^T} \prod_{t=1}^{T} vsm_{q_t}(o_t)$$

Problème: $O(T \times |Q|^T)$

Solution : programmation dynamique

IProbabilité forward

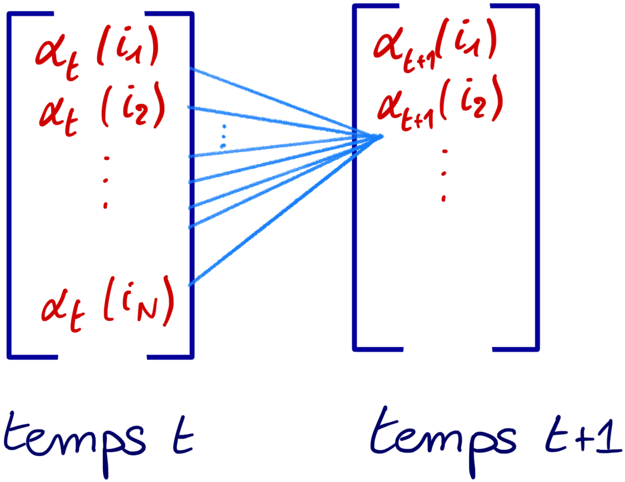
Etant donné un moment t et un état j, on nomme

$$\alpha_t(j) = P(o_1 \ ,\ldots,\ o_t,\ q_t \ = j)$$

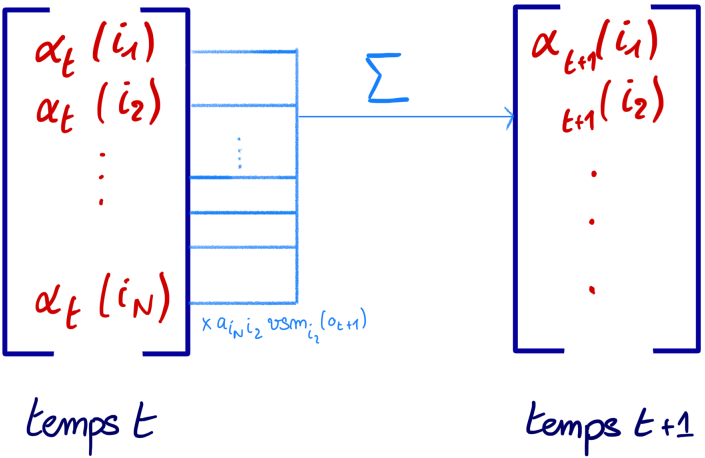
Le résultat viendra par:

$$P(O = o_1 \ ,\ldots,\ o_T) = \sum_{j \in Q} P(o_1 \ ,\ldots,\ o_t,\ q_T \ = j)$$

$$= \sum_{j \in Q} \alpha_T(j)$$



$\alpha_t(i_1)$
$\alpha_t(i_2)$
$\alpha_t(i_N)$
$\alpha_{t+1}(i_1)$
$\alpha_{t+1}(i_2)$
temps t
temps $t+1$


αt(i1)
αt(i2)
αt(iN)
Σ
× a iN i2 vsm i2 (ot+1)
αt+1(i1)
t+1(i2)
temps t
temps t+1

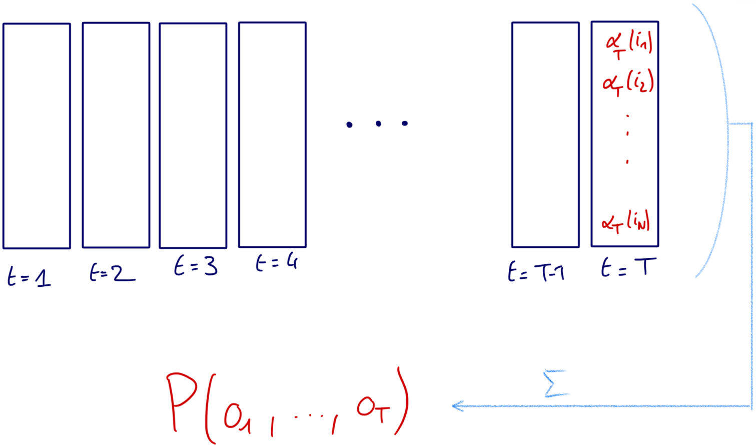
Relation de récurrence

On a au début

$$\forall j \in Q, \quad \alpha_1(j) = vsm_j(o_1)$$

et $\forall t \in [\![2, T]\!]$, $\forall j \in Q$,

$$\alpha_t(j) = vsm_j(o_t) \sum_{i \in Q} \alpha_{t-1}(i)\ a_{ij}$$


$\alpha_T(i_1)$
$\alpha_T(i_2)$
$\alpha_T(i_N)$
$t=1$
$t=2$
$t=3$
$t=4$
$t=T-1$
$t=T$
Σ
$P(o_1, \dots, o_T)$

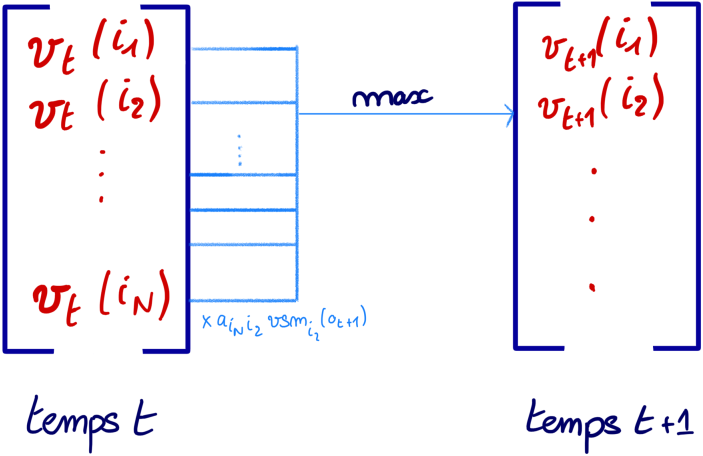
Algorithme de Viterbi

Données: $A\ ,\ (vsm_i(\mu))_{J\ \in Q},\ O = (o_1,\ ...,o_T)$

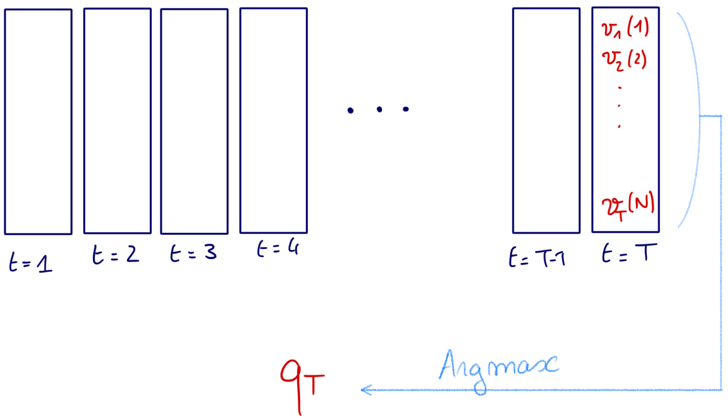
Objectif: $(q_1,...,q_T) \in Q^T\ \ :\quad P(O|\ (q_1,\ ...|q_T))$ est maximale

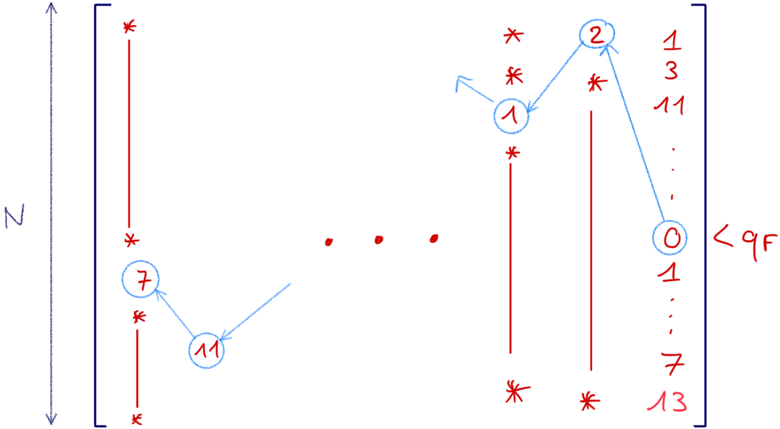
Etant donné un moment t et un état j, on nomme

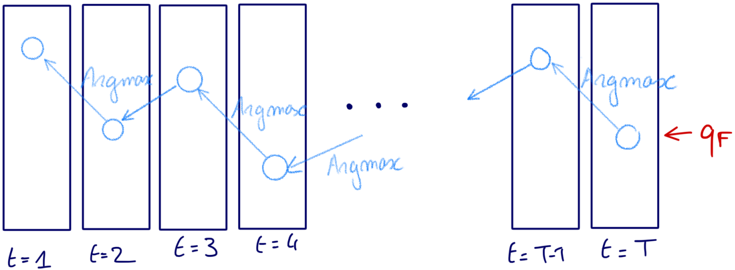
$$v_t(j) = \max_{(q_0,\dots,q_{t-1})} P(o_1,\dots,o_t|q_0,\dots,q_{t-1},q_t = j)$$



$$\boxed{v_{t+1}(j) = vsm_j(o_t) \times \max_{i\ \in Q} \left(v_t(i)\ a_{ij}\right)}$$


$v_1(1)$
$v_2(2)$
$v_T(N)$
$t=1$
$t=2$
$t=3$
$t=4$
$t=T-1$
$t=T$
Argmax
q_T


N
7
11
1
2
1
3
11
0
< qF
1
7
13



Argmax
Argmax
Argmax
Argmax
q_F
$t=1$
$t=2$
$t=3$
$t=4$
$t=T-1$
$t=T$

Estimation des paramètres

Algorithme de Baum-Welch

On fixe Q et $\mathcal{O}$.
Étant donné une observation $O = (o_1, \dots, o_T)$,
quel est θ maximisant $P(O|\theta)$?

Probabilité de transition:

$$a_{ij} = \frac{\text{le nombre de fois ou la transition a été faite}}{\text{le nombre de fois où l'on était en i}}$$

Probabilité d'observation:

$$vsm_i(o) = \frac{\text{le nombre de fois ou l'observation o a été faite en i}}{\text{le nombre de fois où l'on était en i}}$$

Idée:

- Premier modèle « à la main »
- Réestimation via le modèle
- Actualisation

Condition d'arrêt

$$|P(O|\theta_{n+1}) - P(O|\theta_n)| < \mathcal{E}$$

Probabilité backward

Etant donné un moment t et un état j, on nomme

$$\beta_t(i) = P(o_{t+1},....o_T | q_t = i)$$

et comme avec les $(\alpha_t(i))$, on a une relation de récurrence

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\ vsm_j(o_{t+1})\beta_{t+1}(j) \quad \text{et} \quad \beta_T(i) = a_{iF}$$

On a alors pour tout t:

$$P(O|\theta) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j)$$

On note pour $t \leq T$, $(i,j) \in Q^2$,

$$\zeta_t(i,j) = P(q_t = i, q_{t+1} = j | O, \theta)$$

pour pouvoir ensuite estimer

$$\boxed{a'_{i,j} = \frac{\displaystyle\sum_{t=1}^{T} \zeta_t(i,j)}{\displaystyle\sum_{k=1}^{N} \sum_{t=1}^{T} \zeta_t(i,k)}}$$

Comment estimer $\zeta_t(i,j)$?

Etapes:

- D'abord

$$P(q_t = i, q_{t+1} = j, O \mid \lambda) = \alpha_t(i)\ a_{i,j}\ vsm_j(o_{t+1})\ \beta_t(j)$$

- *avec*

$$P(A|B,C) = \frac{P(A,B|\ C)}{P(B|C)}$$

- On obtient:

$$\zeta_t(i,j) = \frac{P(q_t = i, q_{t+1} = j, O \mid \lambda)}{P(O|\lambda)}$$

- Algorithme **forward** $\longrightarrow P(O \mid \theta) = \sum_{i=0}^{N} \alpha_T(i)$

- Finalement

$$\boxed{\zeta_t(i,j) = \frac{\alpha_t(i)\ a_{i,j}\ vsm_j(o_{t+1})\ \beta_t(j)}{\sum_{i=0}^{N} \alpha_T(i)}}$$

Probabilités d'observation

On passe par

$$P(q_t = j|O,\theta) = \frac{P(q_t = j, O|\theta)}{P(O|\theta)} = \frac{\alpha_t(j)\beta_t(j)}{P(O|\theta)}$$

Comme avec $a_{i,j}$, pour une observation O:

$$\boxed{vsm_j(\nu) = \frac{\displaystyle\sum_{t\ \ tel\ \ que\ \ o_t=\nu} P(q_t = j|O,\theta)}{\displaystyle\sum_{t=1}^{T} P(q_t = j|O,\theta)}}$$

Fonctionnement de l'ADN

Nucléotides: [A, U, C, G]
Brin d'ADN: ACGUAGCUGA
Codon: AUG, CGA, etc...
Protéine: [codon1][codon2]etc...

Problème:

- Début de séquence ?
- Faux départs ?
- Pourquoi une chaine de Markov Cachée ?

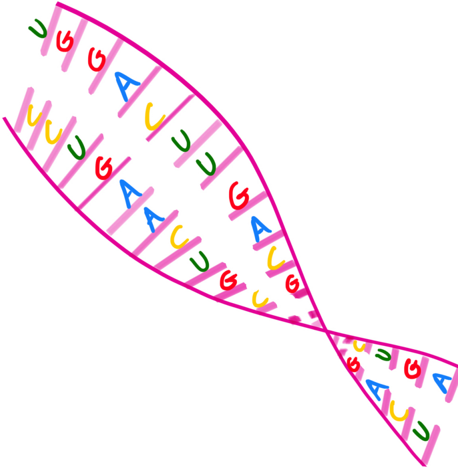


Fig. 13 – *Brin d'ADN*

Modélisation

- Observations: séquence de nucléotides
- Etats cachés: [intron, exon, entrée, sortie]
- Taches:
  1) Entrainer notre chaine de Markov cachée sur un gène
  2) Appliquer l'algorithme de Viterbi à de l'ADN du même animal

Résultats

- **Echantillon d'entrainement:** 4110 nucléotides contenant une partie codante et non codante
- **Echantillon d'essai:** gène de la même espèce précédé d'une centaine de nucléotides (non codants)
  Source: NIH (National Institute of Health)

- **Constat:**
  1) Bon début, arrêt prématuré
  2) de nombreux faux départs

Explication

- **Probabilité de rester consécutivement sur le même état**

- **Par le théorème ergodique:**

$$a_{i,i}^{(n)} \underset{n \longrightarrow \infty}{\longrightarrow} \frac{1}{\mu_i}$$

Potentielle solution à explorer

- **Semi-Chaines de Markov**
- **Chaines non homogènes**

Code

```python
import random as rd
def vol_du_joueur(N, p):
    for _ in range(3):
        X = [0]
        Y = [N]
        temps = 0
        A = N
        while A > 0 and A < 2*N:
            temps += 1
            if rd.random()< p:
                A += 1
            else:
                A -= 1
            X.append(temps)
            Y.append(A)
        plt.plot(X, Y)
    
    plt.xlabel("temps")
    plt.ylabel("Pieces")
    plt.show()
```

Code

```python
def temps_de_vol(N,p):
    Y = []
    nb_essais = 100000
    for k in range(nb_essais):
        temps = 0
        A = N
        while A > 0 and A < 2*N:
            temps += 1
            if rd.random()< p:
                A += 1
            else:
                A -= 1
        Y.append(temps)
    Yp = [0]*(max(Y)+1)
    for y in Y:
        Yp[y] += 1
    plt.bar(list(range(max(Y)+1)), Yp, width=1.0, edgecolor = "#981FFA")
    plt.show()
```

Code

```python
def ruine_du_joueur(N, p, T = 100):
    X_t = np.zeros(2*N+1)
    X_t[N] = 1.0
    T = list(range(1, T))
    A = np.reshape(np.zeros((2*N+1)**2), ((2*N+1),(2*N+1)))
    A[0][0] = 1
    A[-1][-1] = 1

    for i in range(1, 2*N):
        A[i][i-1] = 1-p
        A[i][i+1] = p

    print(A)
    Argent = []
    for t in T:
        m = max(X_t)
        for k in range(2*N+1):
            if X_t[k] == m:
                Argent.append(k)
                break
        X_t = np.dot(X_t, A)
    plt.plot(T, Argent)
```

Code

```python
def epidemie_1(temps = 50, population = 10**6):
    propagation = np.array( [[0.9 , 0.1], [0.3, 0.7]]) # 0 -> infecte, 1 -> sain
    popu = np.array([0, 1])
    X_temps = np.linspace(0, temps, temps)
    Y_infectes = []
    for t in range(temps):
        Y_infectes.append(popu[0]*population)
        popu = np.dot(popu, propagation)
    plt.plot(X_temps, Y_infectes)
```

Code

```python
def epidemie_2(temps = 100, population = 10**6):
    propagation = np.array( [
        [0.2, 0.2, 0, 0, 0.0001, 0.0999], # 0 -> infecte vaccine
        [0.2, 0.8, 0, 0, 0, 0], # 1 -> sain vaccine
        [0 , 0.2, 0.1, 0.7, 0, 0], # 2 -> sain non vaccin
        [0,0.2, 0, 0.7, 0.001, 0.099], # 3 -> infecte non vaccine
        [0, 0, 0, 0, 1, 0 ], # 4 -> mort
        [0, 0, 0, 0, 0, 1 ] # 5 -> immunise
    ])
    popu = np.array([0, 0, 1, 0, 0, 0])
    X_temps = np.linspace(0, temps, temps)
    Y_infectes = []
    for t in range(temps):
        Y_infectes.append(popu[0]*population)
        popu = np.dot(popu, propagation)
    plt.plot(X_temps, Y_infectes)
```

Code

```python
def forward(A, B, Obs):
    N = len(A)
    T = len(B)
    alpha_prec = np.array(B[:][Obs[0]])
    alpha_suiv = np.zeros(N)
    for t in range(T):
        nouv_alpha = np.zeros(N)
        for j in range(N):
            nouv_alpha[j] = B[j][Obs[t]] * sum( alpha_suiv[i] * A[i][j] for i in range(N))
        alpha_prec = alpha_suiv[:]
        alpha_suiv = nouv_alpha[:]
    return sum(alpha_suiv)
```

Code

```python
def Viterbi(A, B, Obs):
    # A matrice de transition
    # B les probabilites d'observation tq b_j(o_t) = B[j][t]
    # On travaille avec des logarithmes
    logA = np.log(A)
    logB = np.log(B)
    N = len(A)
    T = len(Obs)
    pointeurs =  np.reshape(np.zeros(T*N), (N,T)) # sert a retracer le chemin a la fin
    alpha_prec =  np.array(B[:][Obs[0]])
    alpha_suiv = np.zeros(N)
    for t in range(T):
        nouv_alpha = np.zeros(N)
        for j in range(N):
            nouv_alpha[j], pointeurs[j][t] = max_arg( np.array( np.log(alpha_suiv[i]) + logA[i][j] + logB[j][Obs
            # log est croissante, conserve donc le max
            # on met en pointeur l'etat i qui realise le maximum : c'etait l'etat precedent
        alpha_prec = alpha_suiv[:]
        alpha_suiv = nouv_alpha[:]

    pmax, i_final = max_arg(alpha_suiv)
    pmax = np.exp(pmax)
    etats_successifs = np.zeros(T)

    i = i_final
    for t in range(1, T+1, -1):
        etats_successifs[t] = i
        i = pointeurs[i][t-1]
    return pmax, etats_successifs
```

Code

```python
def max_arg(liste):
    """ renvoie le max de la liste et le plus petit indice ou il a ete realise"""
    m = liste[0]
    i_max = 0
    for n in range(len(liste)):
        if liste[n] > m:
            m = liste[n]
            i_max = n
    return m, i_max
```

Code

```python
def baum_welch_naif(A, B, Obs):
    N = len(A)
    T = len(Obs)
    alphas = np.reshape(np.zeros(N*T), (T, N))
    betas = np.reshape(np.zeros(N*T), (T, N))
    # trouver toutes les valeurs des alphas et betas
    alphas[:][0] = B[:][Obs[0]]
    betas[T-1][:] = np.ones(N)

    for t in range(1, T-2):
        for j in range(N):
            alphas[t][j] =  B[j][Obs[t]] * sum( alphas[t-1][i] * A[i][j]  for i in range(N))
            betas[T-1-t][j] = B[Obs[T-t]][j] * sum( betas[T-t][i] * A[j][i] for i in range(N))

    Pobs = sum(alphas[T-1][:])
    # step Expectations
    zeta = np.reshape(np.zeros(N*N*T), (T,N, N))
    gamma = np.reshape(np.zeros(N*T), (T,N))

```

```python
    for t in range(T-1):
        for i in range(N):
            for j in range(N):
                zeta[t][i][j] = alphas[t][i] * betas[t+1][j] * A[i][j] * B[Obs[t]][j] / Pobs
    for t in range(T):
        for i in range(N):
            gamma[t][i] = (alphas[t][j] * betas[t][j]) / Pobs

    #step S

    nouvA = np.reshape(np.zeros(N**2), (N,N))
    nouvB = np.reshape(np.zeros(N * len(B[0])), (N, len(B[0])))

    for i in range(N):
        denom = sum( sum( zeta[t][i][k] for k in range(N)) for t in range(T))
        for j in range(N):
            nouvA[i][j] = sum( zeta[t][i][j] for t in range(T)) / denom

    for j in range(N):
        for k in range(len(B)):
            denom = sum(gamma[t][j] for t in range(T))
            for t in range(T):
                if Obs[t] == k:
                    nouvB[j][k] = nouvB[j][k] + gamma[t][j] / denom

    return nouvA, nouvB
```

Code

```python
def traite_fichier_adn():
    nucleotide = open("adn_pur.txt", "r")
    nombres = open("adn_traite", "a")
    lignes = nucleotide.readlines()
    N = ['a', 'c', 't', 'g']

    for l in lignes:
        for carac in l:
            if carac == 'a':
                nombres.write("0 ")
            if carac == 'c':
                nombres.write("1 ")
            if carac == 't':
                nombres.write("2 ")
            if carac == 'g':
                nombres.write("3 ")
    nucleotide.close()
    nombres.close()

adn = open("adn_traite", "r")
sequence = adn.readlines()
Ob = []
for ligne in sequence:
    for nclt in ligne:
        if nclt in ['0', '1', '2', '3']:
            Ob.append(int(nclt))
adn.close()
```