Bons codes

- Facile et efficace à implémenter
  - Codage et décodage (détection/correction) peu coûteux
  - logiciel et/ou matériel
  - Exemple: codes linéaires cycliques
- Etant donné un taux de correction « t/n » donné, pouvoir facilement construire un code (n,k,d) qui permet ce taux de correction
  - Exemple: codes de Reed-Solomon

Codes par interpolation / Reed-Solomon

V : un corps (fini) => un mot de V^k est vu comme un polynome de degré k-1 à coefficients dans V.

1. **Rappel : interpolation/évaluation – Vandermonde et FFT.**
2. Code par interpolation : codage et décodage sans erreur. Exemple simple
3. Décodage avec erreurs (Berlekamp-Welch)
4. Extensions: décodage par Euclide étendu. Extension au cas entier

Evaluation / interpolation matrice de Vandermonde

- $[a_0, \ldots, a_{n-1}]$ n abscisses d'évaluation distinctes, fixées.
- Soit $P_S(X) = \Sigma_{i=0..n-1}\ s_i.X^i$ un polynôme : coefficients $[s_0, \ldots, s_{n-1}]$ et soit ses évaluations $y_j = P_S(a_j)$: évaluation $[y_0, \ldots, y_{n-1}]$
- Soit $\mathbf{G_{n,n}} =$

$$\begin{pmatrix} a_0^0 & a_1^0 & a_2^0 & \ldots & a_{n-1}^0 \\ a_0^1 & a_1^1 & a_2^1 & \ldots & a_{n-1}^1 \\ & & \ldots\ a_j^i\ \ldots & & \\ a_0^{k-1} & a_1^{k-1} & a_2^{k-1} & \ldots & a_{n-1}^{k-1} \\ a_0^{n-1} & a_1^{n-1} & a_2^{n-1} & \ldots & a_{n-1}^{n-1} \end{pmatrix}$$

(matrice de Vandermonde, inversible)

- Alors : $[y_0, \ldots, y_{n-1}] = [s_0, \ldots, s_{n-1}] \cdot G_{n,n}$

  et $[s_0, \ldots, s_{n-1}] = [y_0, \ldots, y_{n-1}] \cdot G_{n,n}^{-1}$

Interpolation aux racines de l'unité et FFT

- Soit ω une racine primitive n-ième de l'unité
  - ie $\omega_n = 1$ et $\omega_0, \omega_1, \omega_2, \ldots \omega_{n-1}$ sont n valeurs distinctes.
- Soit $\mathbf{G_{n,n}} = \Omega_n$ la matrice de Vandermonde : $\Omega_{i,j} = \omega^{i.j}$. Alors
  - $[y_0, \ldots, y_{n-1}] = [s_0, \ldots, s_{n-1}] \cdot \Omega_n$: calcul en O(n log n) ops [FFT]
  - $[s_0, \ldots, s_{n-1}] = [y_0, \ldots, y_{n-1}] \cdot \Omega_n^{-1}$: calcul en O(n log n) ops [FFT-inverse]
  -
- En choisissant comme abscisses d'évaluation les n racines de l'unité, évaluation et interpolation se calculent en O(n log n) opérations sur le corps de base.
- Dans la suite, on prend a_i distincts quelconques (et on peut choisir $a_i = \omega^i$ pour accélérer codage et décodage)

Quelques mots sur la FFT

DEBUT

« The Top 10 Algorithms of the 20th»

[J. Dongarra, F. Sullivan editors, Computing in Science and Engineering, Jan./ Feb. 2000]

[Science page 799, February 4, 2000]

- 1946: The Metropolis Algorithm for Monte Carlo.
- 1947: Simplex Method for Linear Programming.
- 1950: Krylov Subspace Iteration Method.
- 1951: The Decompositional Approach to Matrix Computations.
- 1957: The Fortran Optimizing Compiler.
- 1959: QR Algorithm for Computing Eigenvalues.
- 1962: Quicksort Algorithms for Sorting.

- **1965: Fast Fourier Transform.**
  « An algorithm the whole family can use »
  - « (…) *the most ubiquitous algorithm in use today to analyze and manipulate digital or discrete data. The FFT takes the operation count for discrete Fourier transform from $O(N^2)$ to $O(N \log N)$.* »
  - "Life as we know it would be very different without the FFT. » *Charles Van Loan*

- 1977: Integer Relation Detection.
- 1987: Fast Multipole Method.

Transformée de Fourier discrète (DFT)

Soit (A, +, ×, 0, 1) un anneau commutatif tel que
- $n = 1+ \ldots + 1$ (n fois) est inversible, d'inverse n^{-1},
- il existe dans A un élément ω qui est une racine n-ième primitive de l'unité:

$$\omega^0 = \omega^n = 1 \text{ et } \quad \text{pour } 0 < j < n : \omega^j \neq 1.$$

Définition : Soit $\boldsymbol{u = [u_0, \ldots, u_{n-1}]}$ un vecteur de A^n.
La transformée de Fourier discrète $\hat{u}$ *de* $\boldsymbol{u}$ *par rapport à* ω est

$$\mathbf{DFT}_\omega(u) = \hat{u} = [\hat{u}_0, \ldots, \hat{u}_{n-1}] \text{ avec } \hat{u}_j = \Sigma_{0 \le k < n}\, u_k . \omega^{k.j} .$$

Remarques:
- La valeur de chaque $\hat{u}_k$ dépend des n entrées $u_0, \ldots, u_{n-1}$
- « ***Transformée*** » car $\mathbf{DFT}_\omega$ est inversible :

$$\mathbf{DFT}_\omega(u) = \hat{u} \quad \Leftrightarrow \quad u = n^{-1} . \mathbf{DFT}_{\omega^{-1}}(\hat{u})$$

« *transformée de Fourier inverse* »

Algorithme FFT « Radix 2 » [Cooley-Tuckey 1965]

- On pose $n = 2.m$: $\omega^m = -1$;
  - $\hat{u}_{2k} = \Sigma_{0 \le j < m-1} (u_j + u_{j+m}) . \omega^{2.k..j}$
  - $\hat{u}_{2k+1} = \Sigma_{0 \le j < m-1} (u_j - u_{j+m}) . \omega^j . \omega^{2.k..j}$

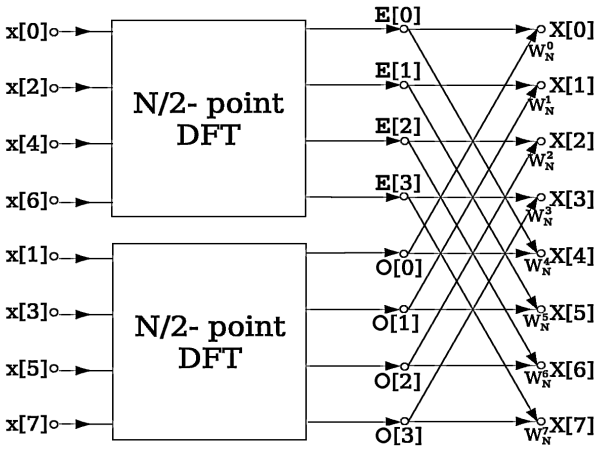
- D' où, en posant :
  - $\theta = \omega^2$ est une racine $m^{ième}$ de l'unité
  - $v_j = (u_j + u_{j+m}) \Rightarrow \hat{u}_{2k} = \Sigma_{0 \le j < m-1} v_j . \theta^{k..j}$
  - $w_j = (u_j - u_{j+m}) . \omega^j \Rightarrow \hat{u}_{2k+1} = \Sigma_{0 \le j < m-1} w_j . \theta^{k..j}$

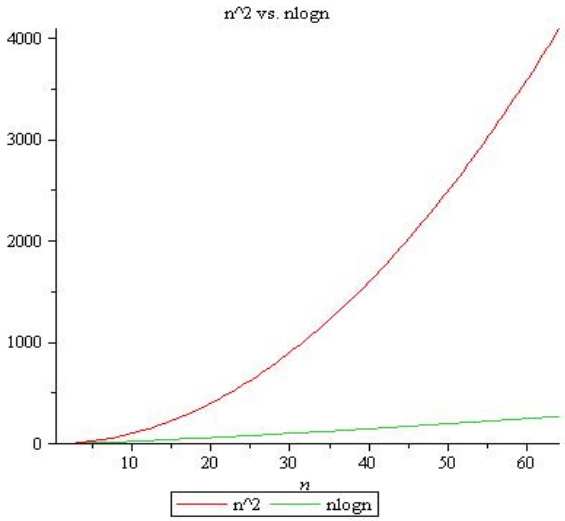
FFT de taille m=n/2

FFT de taille m=n/2

« **Twiddle factor** »

Algorithme FFT « Radix 2 » [Cooley-Tuckey 1965]





- $Coût_{FFT}(n) = 2.\ Coût_{FFT}(n/2) + \Theta(n) = \Theta(\ \boldsymbol{n.\log_2 n}\ )$ opérations.
- Remarque: généraliation: FFT de taille n=***k.m*** se ramène à
  [***k*** FFT de taille ***m***] + [transposition ***m*** × ***k*** → ***k*** × ***m***] + [***m*** FFT de taille ***k***]

Principal Discoveries of Efficient Methods of Computing the DFT

Researcher(s)	Date	Lengths of Sequence	Number of DFT Values	Application
C. F. Gauss [10]	1805	Any composite integer	All	Interpolation of orbits of celestial bodies
F. Carlini [28]	1828	12	7	Harmonic analysis of barometric pressure variations
A. Smith [25]	1846	4, 8, 16, 32	5 or 9	Correcting deviations in compasses on ships
J. D. Everett [23]	1860	12	5	Modeling underground temperature deviations
C. Runge [7]	1903	2^nK	All	Harmonic analysis of functions
K. Stumpff [16]	1939	2^nK, 3^nK	All	Harmonic analysis of functions
Danielson & Lanczos [5]	1942	2^n	All	X-ray diffraction in crystals
L. H. Thomas [13]	1948	Any integer with relatively prime factors	All	Harmonic analysis of functions
I. J. Good [3]	1958	Any integer with relatively prime factors	All	Harmonic analysis of functions
Cooley & Tukey [1]	1965	Any composite integer	All	Harmonic analysis of functions
S. Winograd [14]	1976	Any integer with relatively prime factors	All	Use of complexity theory for harmonic analysis

M. T. Heideman, D. H. Johnson, C. S. Burrus, *Gauss and the history of the fast Fourier transform*,

Quelques mots sur la FFT

FIN

Codes par interpolation / Reed-Solomon

V : un corps (fini) => un mot de V^k est vu comme un polynome de degré k-1 à coefficients dans V.

1. Rappel : interpolation/évaluation – Vandermonde et FFT.
2. **Code par interpolation : codage et décodage sans erreur. Exemple simple**
3. Décodage avec erreurs (Berlekamp-Welch)
4. Extensions: décodage par Euclide étendu.
   Extension au cas entier

Codage par évaluation / Interpolation

- Soit S=[s_0, …, s_{k-1}] mot source
- $P_S(X) = \Sigma_{i=0..k-1}\ s_i.X^i$ de degré k-1 sur V
  - caractérisé de manière unique par sa valeur en k points a_i distincts

- Pour **n ≥k** (et n ≤ #V), et pour n points a_j distincts fixés
  Alors **$[s_0, …, s_{k-1}] \Leftrightarrow [y_0, …, y_{n-1}]$** avec $y_j=P_S(a_j)$
- Dém:
  - $(s_j)=>(y_j)$: **évaluation** Pour j=0, … n-1: $y_j=\Sigma_{i=0..k-1} s_i.a_j^i$
  - $(y_j)=>(s_j)$: **interpolation**: $\Sigma_{i=0..k-1} s_i.X^i = \Sigma_{i=0..n-1}\ y_i\ L_i.(X)$
    avec $L_i.(X) = \Pi_{j\neq i}\ (a_i - a_j)^{-1}.(X- a_j)$: polynôme de Lagrange

Ex. Code d'interpolation (6, 2) dans V=GF(11)

- Code (6, 2) sur GF(11) par évaluation aux abscisses (0, 1, 2, 3, 4, 5)
  (code de distance 5 donc 2 correcteur !)

Codage Mot source: **[7,2]**
Polynôme associé au mot source: P = 7 + 2 X
Mot de code transmis : **[7, 9, 0, 2, 4, 6]** i.e. évaluation de P aux abscisses (a_i)= (0, 1, 2, 3, 4, 5)

Décodage: (si il n'y a pas d'erreurs) Mot de code reçu : **[7, 9, 0, 2, 4, 6]**
Calcul du polynôme d'interpolation aux valeurs reçues:
(calcul par Lagrange, ou par résolution du système Vandermonde
ou mieux par remontée récursive)
- Interpolation en (a_0=0, y_0=7) et (a_1=1, y_1=9) : P_{01} = 9.X - 7.(X-1) = 2.X + 7 mod 11
- Interpolation en (a_2=2, y_2=0) et (a_3=3, y_3=2) : P_{23} = 2.(X-2) + 0.(X-3) = 2X – 4 = 2X + 7 mod 11
- Interpolation en (a_4=4, y_4=4) et (a_5=5, y_5=6) : P_{23} = 6.(X-4) - 4.(X-5) = 2X – 4 = 2X + 7 mod 11

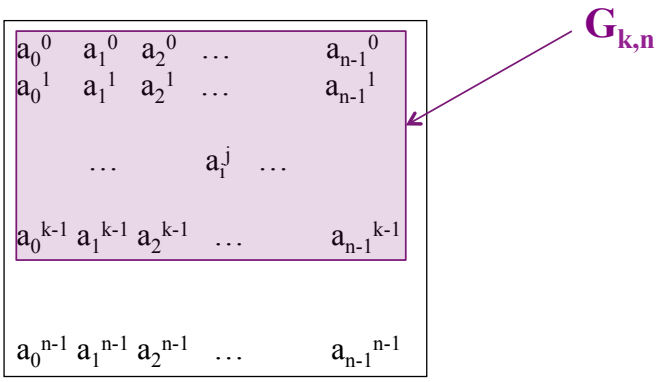
Etc: on fusionne les polynômes récursivement: direct ici.

Polynôme interpolé mod 11 = **7 + 2.X** => Mot décodé: **[7, 2]**

Matrice génératrice du code = matrice de Vandermonde

- k premières lignes de la matrice (n,n) :

$\mathbf{G_{n,n}} =$



Codage / décodage sans erreur par Vandermonde

- Code (6, 2) sur GF(11) par évaluation aux abscisses (0, 1, 2, 3, 4, 5)

- $\mathbf{G_{2,6}}$= Matrice génératrice
  = 2 premières lignes de la matrice de Vandermonde $G_{6,6}$ =

1	1	1	1	1	1
0	1	2	3	4	5
0	1	4	9	5	3
0	1	8	5	9	4
0	1	5	4	5	9
0	1	10	1	9	1

- **Codage** : **multplication par $\mathbf{G_{2,6}}$**
  codage ([7, 2]) = [7 2]. $G_{2,6}$ = [7 9 0 2 4 6]

- **Décodage sans erreur : multiplication par $\mathbf{G_{6,6}^{-1}}$**
  decodage ([7 9 0 2 4 6]) = [7 9 0 2 4 6] . $G_{6,6}^{-1}$ = [7 2 0 0 0 0] => [7 2]

  On reçoit [y_1 y_2 y_3 y_4 y_5 y_6]
  - 1/ Calcul de [s_1 s_2 s_3 s_4 s_5 s_6] = [y_1 y_2 y_3 y_4 y_5 y_6] . $G_{6,6}^{-1}$
  - 2/ Si s_3=s_4=s_5=s_6=0 : on a reçu un mot de code ! => le mot décodé est [s_1 s_2]
    sinon: détection d'erreurs ([s_1 s_2 s_3 s_4 s_5 s_6])

Codes par interpolation / Reed-Solomon

V : un corps (fini) => un mot de V^k est vu comme un polynome de degré k-1 à coefficients dans V.

1. Rappel : interpolation/évaluation – Vandermonde et FFT.
2. Code par interpolation : codage et décodage sans erreur. Exemple simple
3. **Décodage avec erreurs (Berlekamp-Welch)**
4. Extensions: décodage par Euclide étendu. Extension au cas entier

Interpolation et correction

- **Théorème fondamental:**
  Le code (n,k) par évaluation/interpolation est de distance d=n-k+1.
  (NB donc d=r+1 : distance maximale – MDS -).

- Preuve:
  - *Soient n évaluations (y_j) d'un polynôme P de degré ≤ k-1. Parmi ces n évauations, e sont erronées.*
  - *Si 2e ≤ n-k, alors **P est l'unique polynôme de degré < k** qui corresponde à au moins n-e évaluations correctes, i.e.*
    $$\#\{ j=0..n-1 : y_j \neq P(a_j) \} \leq e$$
  - *Corrige donc tout nombre e d'erreurs vérifiant* e≤(n-k)/2 *donc est de distance d = n - k + 1*

- Intérêt*: construction de codes correcteurs de distance d arbitraire :*
  - *Distance maximale, mais il faut suffisament de points d'interpolation*

Décodage du code d'interpolation avec correction [par Berlekamp-Welch]

on reçoit un mot [$y_0 \ldots y_{n-1}$] avec des erreurs

- **Problème décodage avec e≤(n-k)/2=r/2 erreurs et t=n-e≥k+r/2 valeurs correctes.**
  - Entrée: on reçoit $y_0, \ldots, y_{n-1} \in F^n$ vérifiant $\exists P \in F[X]$ de degré k-1 tel que $\#\{i / y_i = P(a_i)\} \geq t$.
  - Sortie : le polynôme P unique tel que $\#\{i / y_i = P(a_i)\} \geq t$.

- **Principe** :
  - Soit $T=\{ i / y_i = P(a_i) = L(a_i) \}$ les indices des valeurs correctes;
  - Les indices $i \notin T$ sont des erreurs; soit $E(X)= \Pi_{i \notin T} (X-a_i)$ le **polynôme localisateur d'erreur.** On a Degré(E) ≤ e [**NB** $(i \notin T) \Leftrightarrow$ (erreur en position i) $\Leftrightarrow (y_i \neq P(a_i)) \Leftrightarrow (E(a_i) = 0)$].
  - Posons $N(X)=P(X).E(X)$ qui est de degré k+e-1: on a donc pour i=0..n-1 : $N(a_i) = P(a_i).E(a_i)$;
  - P et E sont inconnus mais pour $i \in T$: $N(a_i) = E(a_i).P(a_i) = E(a_i).y_i$
    et pour $i \notin T$: $N(a_i) = 0 = E(a_i).y_i$
  - donc, pour i=0..n-1 : $N(a_i) = y_i.E(a_i)$ ie un système de n équations à k+2.e ≤ n inconnues (les inconnues sont les e coefs de E et k+e coefs de N);
  - Ce système admet une solution unique (car P est unique) !

Décodage unique de Reed-Solomon par Berlekamp-Welch

- **Algorithme: on reçoit un mot [$y_0 \ldots y_{n-1}$] avec des erreurs**

1. // Calculer le polynôme d'interpolation $L(X) = \Sigma_{i=0..n-1} b_i.X^i$
   Calculer $\mathbf{[c_0, \ldots, c_{n-1}] := [y_0, \ldots, y_{n-1}] . G_{n,n}^{-1}}$
   Si $[c_k, \ldots, c_{n-1}]$ sont tous nuls, retourner $\mathbf{[c_0, \ldots, c_{k-1}]}$

2. **Sinon correction :** // Résoudre pour i=0..n-1 : $N(a_i) = y_i.E(a_i)$
   // Soient $E(X) = u_0 + u_1X + \ldots + u_{e-1} X^{e-1} + 1. X^e$
   // et $N(X) = v_0 + v_1X + \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots + v_{k+e-1} X^{k+e-1}$
   Résoudre $[v_0\ v_1 \ldots \ldots v_{k+e-1}\ \ 0 \ldots\ 0] \mathbf{. G_{n,n} =} [u_0\ u_1 \ldots u_{e-1}\ \ 1\ 0 \ldots \ldots\ 0] \mathbf{. G_{n,n} . Diag(y_0, \ldots, y_{n-1})}$.

3. Si E(X) divise N(X) : calculer P(X) := N(X) div E(X) et retourner les coefficients $\mathbf{[s_0, \ldots, s_{k-1}]}$ de P.
   Sinon retourner « **Erreur détectée non corrigeable** (plus de (n-k)/2 erreurs) »

- **Coût : O(n log n) si pas d'erreurs; $O(n^3)$ si erreurs.**
  Remarque: il existe des algorithmes plus rapides en $O(n.\log^2 n)$.
  [Berlekamp-Massey, ou calcul de pgcd tronqué en utilisant un algorithme de pgcd rapide].

Résolution système linéaire

- Soit **D** = **Diag(y_0, …, y_{n-1})**
- $[v_0 v_1 \dots \dots v_{k+e-1} \ 0 \dots 0].\mathbf{G_{n,n}} = [u_0 u_1 \dots u_{e-1} \ 1 \ 0 \dots \dots 0].\ \mathbf{G_{n,n}}.\mathbf{D}$.
- Soit $M = \mathbf{G_{n,n}}.\mathbf{D}.\mathbf{G_{n,n}^{-1}}$ et soient
  - $M_{e,n}$ = sous-matrice formée des e premières lignes de M
  - m_{e+1} = ligne e+1 de M
  - Alors: $[v_0 v_1 \dots \dots v_{k+e-1} \ 0 \dots 0] = [u_0 u_1 \dots u_{e-1}].\ \mathbf{M_{e,n}} + m_{e+1}$.

- Soit Q = matrice formée des e premières lignes et e dernières colonnes de M
  $[u_0 u_1 \dots u_{e-1}] . Q = - [m_{n-e}, \dots , m_{n-1}]$, d'où $[u_0 u_1 \dots u_{e-1}]$.
- Puis $[v_0 v_1 \dots \dots v_{k+e-1}] = [u_0 u_1 \dots u_{e-1} \ 1].\ \mathbf{M_{e+1,n}}$.
  ou bien: isoler les a_i incorrects grâce aux u_i (en testant si $E(a_i)=0$) puis interpoler (ou approximer aux moindres carrés) P aux valeurs correctes restantes
- NB On peut trouver aussi un vecteur u qui minimise la norme 2 en ponderant avec un gros poids les composantes qui doivent être nulles.

Lien avec Reed Solomon

- Produit matrice-vecteur rapide => FFT
  - Choix pour x_i = racines de l'unité
  - Dans GF(q) => q-1 racines de l'unité distinctes = ω^i
  - Donc limite le degré $n \leq q-1$

- Reed-Solomon: code d'interpolation de rendement maximal avec les racines de l'unité comme absicces :
  - n = q-1
  - $k \leq n$ arbitraire. Permet de corriger t = (n-k)/2 erreurs.
  - Codage: en O(n log n) [FFT]
  - Décodage en $O(n \log^2 n)$ [Euclide rapide ou Berlekamp-Massey]

Exercice : le code RS (255, 223)

- Consultative Commitee for Space Data Standard :
  échange de données spatiales codées avec le code de Reed-Solomon
  RS(255,223)
  - Code utilisé pour système Galileo

- Pour le code de RS(255,223), préciser:
  - Le corps de base
  - Les caractéristiques (n, k et r) et la forme d'une matrice génératrice
  - La distance et le nombre d'erreurs corrigées
  - Le rendement
  - (vu comme un code binaire), Le nombre de bits d'un mot source et d'un mot de code la distance (en nombre de bits différents) et le nombre d'erreurs de bits corrigées.

Extension: Power decoding

- Généralisation du schéma précédent en élevant y_i à la puissance q
  - Posons $N^{(q)}(X)=P^q(X).E(X)$ qui est de degré $q.(k-1)+e$: donc pour i=0..n-1: $N^{(q)}(a_i)=P^q(a_i).E(a_i)$
  - P et E sont inconnus mais pour $i \in T$: $N^{(q)}(a_i) = E(a_i).P^q(a_i) = E(a_i).y_i^q$
    et pour $i \notin T$: $N^{(q)}(a_i) = 0 = E(a_i).y_i^q$
    ie un système de n équations dont les inconnues sont les e coefs de E et les $q.(k-1)+e+1$ de $N^{(q)}$.

  - En prenant successivement q=1 (cas de base précédent), q=2, q=3, … q= b : on obtient un système d'équations:
    $$N^{(q)}(a_i) = y_i^q.E(a_i) \quad \text{pour } i=0..n-1,\ q=1, \ldots, b$$
  - soit en tout n.b équations (linéairement indépendantes en caractéristique nulle) dont les inconnues sont les coefs de E et ceux de $N^{(1)}$, $N^{(2)}$, …, $N^{(b)}$:
    - à savoir, en tout: $e + \Sigma_{q=1..b}\ q.(k-1)+e+1 = e(b+1) + 1 + (k-1).b(b+1)/2$ inconnues

  - Pour que le nombre d'inconnues soit plus petit que le nombre d'équations (nb), il faut:
    $$e(b+1) + 1 + (k-1).b(b+1)/2 < n.b$$

Codes par interpolation / Reed-Solomon

V : un corps (fini) => un mot de V^k est vu comme un polynome de degré k-1 à coefficients dans V.

1. Rappel : interpolation/évaluation – Vandermonde et FFT.
2. Code par interpolation : codage et décodage sans erreur. Exemple simple
3. Décodage avec erreurs (Berlekamp-Welch)
4. **Extensions: décodage par Berlekamp-Massey**
   Euclide étendu. Extension au cas entier

Autre algorithme de correction

- Euclide étendu tronqué / Berlekamp-Massey

Correction

- Soit Q(X) le polynôme qui interpole les (y_j) en les n points:
  $Q(X) = P(X) + E(X)$ avec E = polynôme d'erreur

- Soit $\Pi(X)=\Pi_{j=0..n-1}(X - x_j)$
- Soit I le sous ensemble des n-t points x_j tels que $P(x_j) = y_j$ [corrects]

- Pour tout x_j de I : $E(x_j)=0$
- Donc E(X) est un multiple de $\Pi_V(X)= \Pi_{j \text{ dans } I}(X - x_j)$:
  $E(X) = Z(X).\ \Pi_V(X)$
  et PGCD (E(X) , $\Pi(X)$) = $\Pi_V(X)$ donc de degré n-t ≥ k+t

- Astuce: les premières étapes du calcul PGCD(Q(X)=P+E, $\Pi(X)$) donnent les mêmes quotients que PGCD(E(X), $\Pi(X)$) !!!

Exemple

- Q = E + P : degré(P) ≤ k-1 ; degre(Q) =degré(E) ≥ n-k
- Séquence des restes R_i dans l'algorithme d'Euclide (eténdu) : $R_i = A_i.\Pi + B_i.Q$
  Initialement $R_0 = \Pi$; $R_1 = Q$; $R_2 = \Pi$ rem Q (reste Euclidien)

$$\mathbf{R_i = A_i.\Pi + B_i.Q}$$

$\mathbf{B_i}$

Algorithme de correction par Euclide tronqué

- Entrée: Π , Q = polynôme d'interpolation aux n points y
- Sortie: P de degré ≤ k-1 qui correspond à n-t évaluations

- $A_0=1.X^0$; $B_0 = 0$; $R_0=\Pi$;
  $A_1=0$; $B_1=1.X^0$; $R_1=Q$;
- For (i=1 ; deg(R_i) ≥ n - t ; i+=1)
  - Soit q_i le quotient euclidien de R_{i-1} par R_i.
  - $R_{i+1} = R_{i-1} - q_i.R_i$;
  - $A_{i+1} = A_{i-1} - q_i.A_i$;
  - $B_{i+1} = B_{i-1} - q_i.B_i$;

- Return P = R_i / B_i ; // B_i est un multiple de R_i

Codage et Décodage en Maple

```
##### Fonction de CODAGE d'un mot a
#####  (i.e. evaluation du polynome Pa(X) aux abscisses x[i] i=1..n
CodageInterpol := proc( a::list)::list ;
  local Pa, y, i ; # Pa est le polynome associé à a; y le vecteur des évaluations
  description "Codage par interpolation aux n points x[i] du mot source a de longueur k »;
  Pa := sum( op(i+1,a)*'X'^i, i=0.. k-1) mod q :
  y := [seq( eval(Pa, X=x[i]) mod q, i=1..n )] :
  print("Polynome associé au mot source:", Pa) :
  print("Mot de code transmis :", y) :
  y
end proc;
```

Polynôme d'interpolation (formule de Lagrange)

Algorithme d'Euclide tronqué (reconstruction rationnelle)

Polynôme source après correction

```
##### Fonction de DECODAGE ET CORRECTION par algorithme d'Euclide tronqué
DecodageInterpol := proc( yrecu::list)::list ;
  local
    Precu,  # Le polynome d'interpolation de degré n associé au mot recu
    Pcorr, motcorrige, # Le polynome corrigé et le mot associé
    A0, B0, A1, B1, # Les coefficients de Bezout dans Euclide
    i, aux ;# Variables internes pour les boucles et  pour les permutations
  description "Decodage par interpolation aux n points x[i] du mot recu de longueur n";
  Precu:= expand( sum(  op(i,yrecu)*op(i,LL), i=1..n )) mod q;
  print("Mot reçu : yrecu = ", yrecu ) :
  print("Interpolation de yrecu  :", Precu ) :
  A0:=1: B0:=0: R0 := PI:  print("Reste Euclide tronqué:",  R0) :
  A1:=0: B1:=1: R1 := Precu: print("Reste Euclide tronqué:",  R1):
  while (degree(R1) >= n-t) do
    quotient := Quo(R0, R1, X) mod q:
    aux := Expand(R0 - quotient*R1) mod q: R0:=R1: R1:=aux : print("Reste Euclide tronqué:",  R1):
    aux := Expand(A0 - quotient*A1) mod q: A0:=A1: A1:=aux :
    aux := Expand(B0 - quotient*B1) mod q: B0:=B1: B1:=aux :
  end do :
  if ( Rem(R1,B1,X) mod q <> 0) then
    print ("*** ERREUR: CORRECTION IMPOSSIBLE *** Reste non nul=", Rem(R1, B1, X) mod q  ) :
   motcorrige :=[ "ECHEC DECODAGE car trop d'erreurs!" ]
  else
    Pcorr := Quo(R1, B1, X) mod q:
    motcorrige := [seq(coeff(Pcorr,'X',i), i=0..max(degree(Pcorr),k-1) ) ] :
    print( "Polynome interpolé corrigé=", Pcorr, " qui correspond au mot source:", motcorrige)
  end if :
  motcorrige
end proc;
```

Un exemple dans GF(11)

- Evaluation aux abscisses (0, 1, 2, 3, 4, 5) donc Code (6, 2, 5) qui est t=2 correcteur

Codage Mot source: **[7,2]**
Polynôme associé au mot source: $P = 7 + 2X$
Mot de code transmis : **[7, 9, 0, 2, 4, 6]** (i.e. évaluation de P aux absisses X= 0, 1, 2, 3, 4, 5)

Décodage: Mot reçu avec 2 erreurs: **[7, 4, 0, 2, 6, 6]**
1/ Polynôme d'interpolation aux valeurs reçues: $Q := 2X^5 + 2X^4 + 7X^3 + 6X^2 + 2X + 7$

2/ Séquence des restes dans l'algorithme d'Euclide étendu (et coefficient de Bezout B associé):
Étape 0: $R_0 = L(X) = X(X-1)(X-2)(X-3)(X-4)(X-5)$ $[B_0 = 0]$
Étape 1: $R_1 = Q(X) = 2X^5 + 2X^4 + 7X^3 + 6X^2 + 2X + 7$ $[B_1 = 1]$
Étape 2: $R_2 = R_0 \bmod R_1 = 4X^4 + 4X^3 + 2X^2 + 8X + 1$ $[B_2 = 5X + 8]$
Étape 3: $R_3 = R_1 \bmod R_2 = 6X^3 + 2X^2 + 7X + 7$ $[B_3 = 3X^2 + 7X + 1]$
Arrêt de l'algorithme d'Euclide car degré inférieur ou égal à n-t = 4.

3/ Correction: Calcul de $R_3/B_3 = (6X^3 + 2X^2 + 7X + 7) / (3X^2 + 7X + 1) \bmod 11 = 7X + 2$
=> Mot décodé: **[7, 2]**

De code d'interpolation à Reed-Solomon

Construction du code et test (en Maple)

```
##### CARACTERISTIQUES DU CODE - Constantes globales
q := 11 :  # Le corps de base : Z/11Z
n:=6 : k:=2 : t := (n-k)/2 :
print(cat("Code Interpolation ( ", n, ", ", k, ", ", n-k+1, " ) sur GF(", q, ")qui est ", t,"-correcteur.")):

X := 'X' ; # X est l'indeterminee pour les polynomes a 1 variable.
x:= [seq(i mod q,i=0..n-1)]: print("Abscisses d'evaluation: ", x) :  # Les points d"evaluation
# La base de Lagrange: LL(i), i=1..n
PI:= product( (X-op(i,x)), i=1..n ) mod q ;
LL:=[seq( (expand(PI/(X-op(i,x)mod q)) mod q) * (eval(expand(PI/(X-op(i,x)mod q))mod q, X=op(i,x))^(-1) mod q), i=1..n )] mod q:


#### Fonction de GENERATEUR ALEATOIRE D'ERREUR DANS UN MOT DE TAILLE n
GenereErreurCanal := proc( mot::list, nberreurs::integer)::list ;
 local   err,  # err est le vecteur d'erreur généré aléatoirement et de poids = nberreurs
         poserreurs, # l'ensemble de toutes les positions des erreurs
         pos, j, tmp ; # variable locale
 description "Renvoie (mot + err), mot de longueur n, où err est un vecteur d'erreur de poids de Hamming=nberreurs";
 err := [seq( 0, i=1..n)]: # Pour generer un vecteur d'erreur aleatoire initialisé à 0 erreurs
 poserreurs := {} :
 while ( nops(poserreurs) < nberreurs)  do
   pos := RandomTools[Generate](integer(range=1..n)) :
   tmp := poserreurs union {pos}  : poserreurs := tmp ;
 end do:
 for j in poserreurs do err[j] := RandomTools[Generate](integer(range=1..q-1)) end do:
 (mot + err ) mod q
end proc;

TestCodageDecodage := proc( source::list, nberreurs::integer)::list :
 local     motcode, motrecu, motcorrige:
 motcode := CodageInterpol( source ) :
 motrecu := GenereErreurCanal( motcode, nberreurs ) :
 motcorrige := DecodageInterpol( motrecu ) :
 print("Mot source=", source, " -- Mot transmis=", motcode, " -- Mot recu=", motrecu, " -- Apres correction=", motcorrige):
 motcorrige
end proc
```

Exemple:
TestCodageDecodage([7,2], 2) ;

```
#Reed Solomon: abscisses = racines de l'unité
n := q-1;
k := n-4;
alpha := 7; # [seq( 7**i mod 11, i=1..10)]= [7, 5, 2, 3, 10, 4, 6, 9, 8, 1]
x:= [setCodageDecodage(alpha^i mod q,i=0..n-1)]:
print("Abscisses d'evaluation: ", x) :  # Les points d"evaluation
# La base de Lagrange pour ReedSolomon: LL(i), i=1..n
PI:= product( (X-op(i,x)), i=1..n ) mod q ;
LL:=[seq( (expand(PI/(X-op(i,x)mod q)) mod q)
    * (eval(expand(PI/(X-op(i,x)mod q))mod q, X=op(i,x))^(-1) mod q), i=1..n )] mod q:
```


Reed-Solomon et Codes cycliques

- Pour Reed Solomon, on pourrait prendre 0 en plus des racines de l'unité => Code MDS (n, k, n-k+1) sur V de card $n=p^m$

☺ Si on ne prend que les n-1 racines de l'unité, alors si P(X) est un polynôme de degré k-1, $[\ P(\omega^i)\ ]_{i=0..n-2}$ est un mot de code.
Soit Q(X) = P(ωX) : $[\ Q(\omega^i)\ ]_{i=0..n-2}$ = shift ($[\ P(\omega^i)\ ]_{i=0..n-2}$).
Tout décalage circulaire d'un mot de code est un mot de code!

☺ Plus généralement, il est possible de construire des codes linéaires avec une distance minimale donnée : les codes cycliques.

- Exemple les codes de Reed-Solomon sont cycliques, mais imposent n=q-1.

- Ex: les x_i sont des octets (m=8), il y en a 2^8-1=255
  ⇒ RS(255,253) est alors 2-détecteur et 1-correcteur pour un ajout de 2 octets ($16=\lfloor \log_2(2^8 \times 255) \rfloor + 1$ bits ➔ c'est Hamming(2040,2024))

Généralisation: anneau euclidien

- Remplacer degré(M) par log(M) [attention: log *réel*]

Évaluer P en a $\leftrightarrow$ Reduire P modulo $X - a$

Polynômes	**Entiers**
Evaluation: $P \mod X - a$ Évaluer P en a	$N \mod m$ "Évaluer" N en m
Interpolation: $P = \sum_{i=1}^{k} \frac{\prod_{j \neq i}(X - a_j)}{\prod_{j \neq i}(a_i - a_j)}$	$N = \sum_{i=1}^{k} a_i \prod_{j \neq i} m_j (\prod_{j \neq i} m_j)^{-1[m_i]}$

Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- Codes détecteurs d' erreur
- **Codes correcteurs : Code linéaire, Reed Solomon**
  - Codes linéaires : codage et décodage par syndrome.
  - Codes cycliques et Reed-Solomon
  - **Dimensionnement d' un code de Reed-Solomon**
- Autres codes et applications
  - Rafales d' erreurs. Code CIRC.

Codes binaires de Reed Solomon et dimensionnement

Lien avec Reed Solomon

- Produit matrice-vecteur rapide => FFT
  - Choix pour x_i = racines de l'unité
  - Dans GF(Q) => q-1 racines de l'unité distinctes = ω^i
  - Donc limite le degré $n \leq q-1$
- Reed-Solomon: code d'interpolation de rendement maximal avec les racines de l'unité comme absicces :
  - $n = q-1$
  - $k \leq n$ arbitraire. Permet de corriger $t = (n-k)/2$ erreurs.
  - Codage: en $O(n \log n)$ [FFT]
  - Décodage en $O(n \log^2 n)$ [Euclide rapide ou Berlekamp-Massey]
- Exemple: Consultative Commitee for Space Data Standard : échange de données spatiales avec RS(255,223)

  $\Rightarrow$ 32-détecteur et 16-correcteur; rendement $\approx$ 87,5 %; bits ajoutés = 256

Codes binaires de Reed-Solomon

- V= {chiffres de m bits}: corps binaire à **q**=2^m éléments; ie V=GF(q)
  - Exemples: m=8: octets (256 éléments) m= 32 : mots de 32 bits
- **On choisit n = 2^m - 1**
- On a alors: $X^n - 1 = X^{q-1} - 1 = \prod_{a\in V,\, a\neq 0} (X - a)$.
  - Si α générateur de GF(q)* : { a $\in$ GF(q)* }={α^i / i=0..q-2}
    Donc $X^n - 1 = \prod_{a\in GF(q)^*} (X - \alpha^i)$
  - Remarque: D'après BCH, il suffit de choisir g = $\prod_{i=a..a+r-1} (X - \alpha^i)$!!!
- **Définition 4** : code cyclique de Reed-Solomon RS(n, k) :
  RS(n,k) est un code cyclique de polynôme générateur
  $g(X) = \prod_{i=a..a+r-1} (X - \alpha^i)$ de degré r
- **Théorème 5** : RS(n,k) est de distance d = n-k+1 = r+1
  - D'après théorème BCH : d ≥ r+1
  - D'après la borne de Singleton: d ≤ r+1
- Ex. Galileo : RS(255,223) sur V={octets}, 16-correcteur
- NB RS(n,k) est de distance maximale sur GF(2**m**) !!!
  … mais pas nécessairement parmi les codes binaires…

Dimensionnement d'un code binaire RS

- *Canal binaire symétrique* (BSC) avec un taux τ d'erreurs de bits. Exemple: τ=1%
  Définition: C = Capacité BSC(τ) = 1+ τ.log2 τ + (1- τ).log2(1- τ)

  On suppose m fixé: V=GF(2^m); un chiffre de V = m bits. (parfois m=8, 16, 32 etc)
  p=Prob (erreur transmission d'un chiffre de m bits) = 1-(1- τ)^m.
- Un code RS(n,k) sur V est de rendement k/n et corrige t=(n-k)/2 erreurs de chiffres.
  La probabilité ε d'une erreur non corrigée (***erreur résiduelle***) est :

$$\varepsilon \leq \sum_{i=t+1}^{n} C_n^i p^i (1-p)^{n-i} = \sum_{j=0}^{n-t-1} C_n^j p^{n-j} (1-p)^j$$

- **Théorème de Shannon** : pour tout R < C, il existe un code de rendement R et d'erreur résiduelle arbitrairement petite.
- Exemple: τ=1% => capacité = 0,91… => on cherche un code correcteur de rendement proche de 91% et d'erreur résiduelle faible.

Le choix de m et t définit le code RS … et impose R et ε

- Pour τ=**5.10^{-4} = 0,0005 :** rendement < capacité = 0,993

m	t	r	n	R	ε
8	2	4	255	0,984	0,083314104
8	4	8	255	0,969	0,003850848
8	8	16	255	0,937	1,16402E-06
8	12	24	255	0,906	6,0687E-11
8	16	32	255	0,875	9,02514E-16
8	32	64	255	0,749	1,01325E-38
16	500	1000	65535	0,985	0,830965135
16	600	1200	65535	0,982	0,00038756
16	700	1400	65535	0,979	4,80635E-14
16	800	1600	65535	0,976	4,27197E-30
16	1000	2000	65535	0,969	5,75845E-78

=> ***Code(255, 223) sur GF(256)***
de rendement 87% et 16-correcteur
avec erreur résiduelle = 10^{-15}

NB Chiffres de 8 bits:
bloc de 1784 bits codés par 2040 bits

- Pour τ=**1% = 0,01 :** rendement < capacité = 0,91

m	t	r	n	R	ε
8	8	16	255	0,9373	0,998133479
8	16	32	255	0,8745	0,769179312
8	32	64	255	0,749	0,002590694
8	48	96	255	0,6235	3,08393E-09
8	64	128	255	0,498	6,64789E-18
8	80	160	255	0,3725	7,67393E-29
16	9000	18000	65535	0,7253	1
16	10000	20000	65535	0,6948	0,001808289
16	10500	21000	65535	0,6796	4,22996E-17
16	11000	22000	65535	0,6643	7,73789E-43

Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- Codes détecteurs d'erreur
- Codes correcteurs : Code linéaire, Reed Solomon
- **Autres codes et applications**
  - **Rafales d'erreurs – Code déroulé et entrelacement**.
  - Code CIRC.

Table d'entrelacement : principe simple

- **Entrelacement avec profondeur p** (exemple p=3)
  - Message à coder: a b c d e f g …
  - Après codage (n=5): $a_1a_2a_3a_4a_5b_1b_2b_3b_4b_5c_1c_2c_3c_4c_5d_1d_2d_3d_4d_5e_1e_2e_3e_4e_5f_1f_2$…
  - Table entrelacement à profondeur 3 : (table p x n remplie à la volée)

    $a_1\ a_2\ a_3\ a_4\ a_5$
    $b_1\ b_2\ b_3\ b_4\ b_5$
    $c_1\ c_2\ c_3\ c_4\ c_5$
  - Après entrelacement (écriture de la table par colonne):
    $a_1b_1c_1a_2b_2c_2a_3b_3c_3a_4b_4c_4a_5b_5c_5d_1e_1f_1d_2e_2f_2d_3e_3f_3d_4e_4$…

- Comme 2 symboles d'un même mot de code sont distants de p positions:
  Propriété: si le code initial corrige des rafales de longueur l, l'entrelacement à profondeur p corrige des rafales de longueur L:
  $$L = p.l$$

- **Exemple**: Un code de Hamming entrelacé à profondeur 10 corrige des rafales de longueur 10.

Table d'entrelacement à retard ρ

- **Entrelacement dans une table à n lignes et nr colonnes**
  - Exemple n= 5, ρ = 2 . Message à coder: a b c d e f g …
  - Après codage (n=5): $a_1a_2a_3a_4a_5b_1b_2b_3b_4b_5c_1c_2c_3c_4c_5d_1d_2d_3d_4d_5e_1e_2e_3e_4e_5f_1f_2$…
  - Table n x (ρ.n) remplie à la volée (ici n=5, ρ=2)

a_1	b_1	c_1	d_1	e_1	f_1	g_1	h_1	i_1	j_1
_	_	a_2	b_2	c_2	d_2	e_2	f_2	g_2	h_2
_	_	_	_	a_3	b_3	c_3	d_3	e_3	f_3
_	_	_	_	_	_	a_4	b_4	c_4	d_4
_	_	_	_	_	_	_	_	a_5	b_5

Initialisation [ajout de r.n.(n-1)/2 symboles vides]

m_1	n_1	o_1	p_1	q_1	r_1	s_1	t_1	u_1	v_1
k_2	l_2	m_2	n_2	o_2	p_2	q_2	r_2	s_2	t_2
i_3	j_3	k_3	l_3	m_3	n_3	o_3	p_3	q_3	r_3
g_4	h_4	i_4	j_4	k_4	l_4	m_4	n_4	p_4	p_4
e_5	f_5	g_5	h_5	i_5	j_5	k_5	l_5	m_5	n_5

En régime de croisière

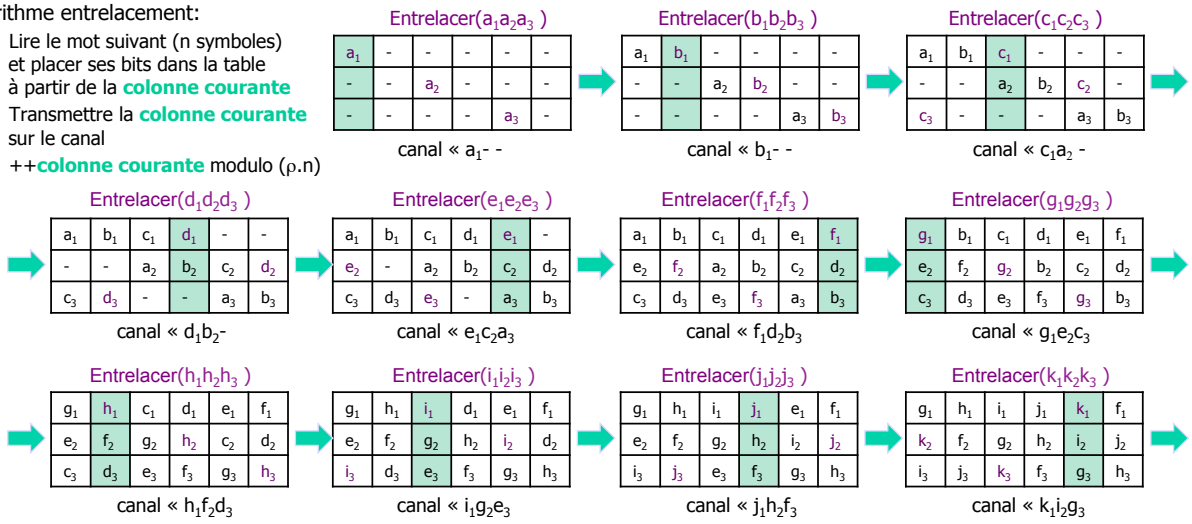
  - Après entrelacement (écriture de la table par colonne):
    a_1- - - - b_1- - - - c_1a_2- - - d_1b_2- - - $e_1c_2a_3$- - $f_1d_2b_3$- - $g_1e_2c_3a_4$- $h_1f_2d_3b_4$- $i_1g_2e_3c_4a_5j_1h_2f_3d_4b_5$…

- Comme 2 symboles d'un même mot de code sont distants de ρ.n+1 positions:
  Propriété: si le code initial corrige des rafales de longueur l, l'entrelacement à retard ρ corrige des rafales de longueur L:
  $$L = (\rho.n+1).l$$

- **Exemple**: Un code de Hamming binaire (15,11) entrelacé à retard 4 corrige des rafales de longueur 61 bits. (avec une table de taille 15 x (15.4) = 900 bits)

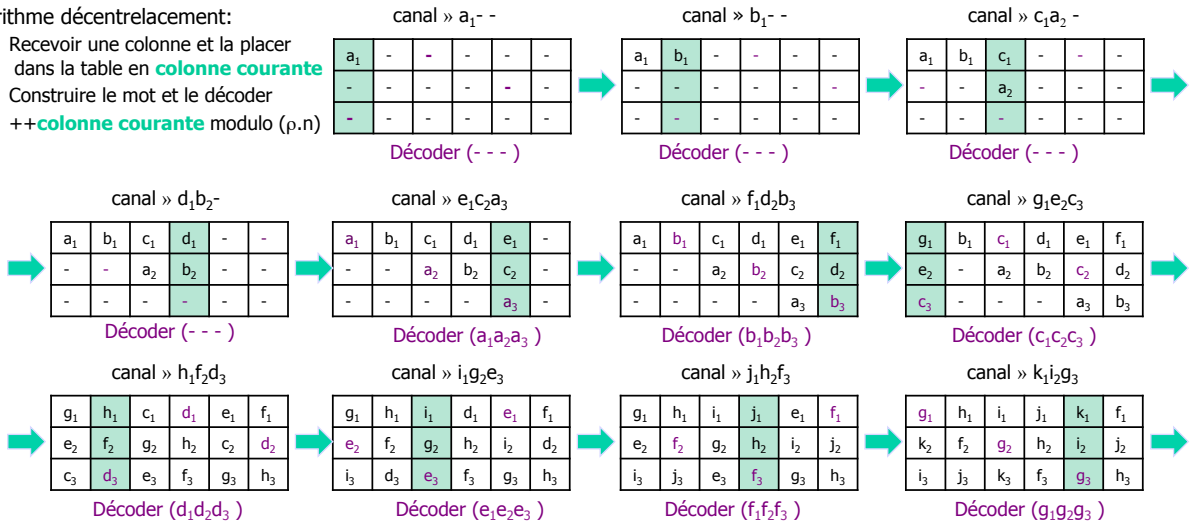
Table d'entrelacement

- Algorithme entrelacement:
  - Lire le mot suivant (n symboles) et placer ses bits dans la table à partir de la **colonne courante**
  - Transmettre la **colonne courante** sur le canal
  - ++**colonne courante** modulo (ρ.n)



Désentrelacement

- Algorithme décentrelacement:
  - Recevoir une colonne et la placer dans la table en **colonne courante**
  - Construire le mot et le décoder
  - ++**colonne courante** modulo (ρ.n)



Effacements

- Effacements et correction
- Code raccourci
- Application: entrelacement croisé
  -Exemple code CIRC

Correction des effacements d'un code linéaire

- **Théorème**: un code linéaire (n,k) sur V systématique de distance d permet de corriger (d-1) effacements
  - Preuve constructive:

Codage

$$[x_1, \ldots, x_k] \cdot \left[\mathbf{G}\ (k,n) \right] = [c_1, \ldots, c_i, \ldots, c_j, \ldots, c_n]$$

(sous forme systématique: toute sous-matrice kxk est inversible)

Canal avec effacements

$$= [c_1, \ldots, ?_i, \ldots, ?_j, \ldots, c_n]$$

$$[x_1, \ldots, x_k] = \left[\mathbf{G^{-1}}\ (k,k) \right] \cdot [c_1, \ldots, ?_i, \ldots, ?_j, \ldots, c_n]$$

(avec k colonnes non effacées)

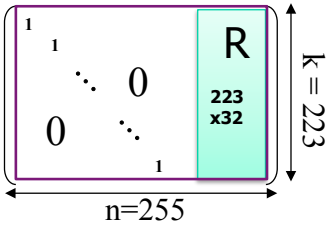
Décodage avec correction

- **Correction** : résolution d'un système linéaire

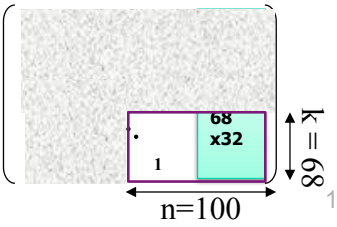
Code raccourci – Exemple Reed-Solomon

- Def 4 : code **raccourci** : soit s ∈ V et 1 ≤i ≤n fixés:
  C' = raccourci(C,i,s) ={$c_1 \ldots c_{i-1}\ c_{i+1} \ldots c_n$ tq $c_1 \ldots c_{i-1}$ s $c_{i+1} \ldots c_n \in C$}
  Remarque: on a d' ≥ d

- Exemple de construction de code raccourci
  - Matrice génératrice sous forme canonique
  - mots de code qui ont ρ '0' aux ρ première positions
    - Suppression des ρ premières lignes et colonnes
    - Code (n-ρ, k-ρ, d)



- Application: code (100,68) dans GF(256) raccourci de Reed-Solomon
  - Matrice initiale RS(255, 223) sous forme canonique
  - Suppression des 155 premières lignes et colonnes



Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- Codes détecteurs d' erreur
- Codes correcteurs : Code linéaire, Reed Solomon
- **Autres codes et applications**
  - Rafales d' erreurs – Code déroulé et entrelacement.
  - **Code CIRC.**

Application

Code CIRC

Cross Interleaved Reed-Solomon code

Code de Reed-Solomon entrelacé croisé

CDROM / DVDROM (1/3)

- Codes par bloc cycliques
- Exemple : CD Audio :
  - Données = octets
  - K=trame = 24 octets codés sur 32 octets sur le CD
  - CIRC = Code de Reed-Solomon entrelacé croisé
    - Code (32, 24) = entrelacement de 2 codes cycliques
  - Base:
    - code de Reed-Solomon (255, 251, 5)
    - Code C1(28,24, 5)
    - Code C2(32,28,5)
  - Codes en détail
  - Description de l' entrelacement

Le code C_1

Il s'agit d'un code (28,24,5) sur le corps à 256 éléments.

Si x est un mot de 24 octets le mot de code de C_1 qui lui correspond est le mot de 28 octets égal à $(x, x\,R_1{}^t)$, où la matrice de dimension (4,24) R_1 est définie par

$$R_1 = \begin{bmatrix} a^{6} & a^{192} & a^{142} & a^{159} & a^{99} & a^{88} & a^{104} & a^{144} & a^{55} & a^{180} & a^{174} & a^{101} & a^{111} & a^{118} & a^{169} & a^{107} & a^{132} & a^{25} & a^{167} & a^{239} & a^{168} & a^{188} & a^{11} \\ a^{45} & a^{108} & a^{248} & a^{131} & a^{64} & a^{221} & a^{100} & a^{235} & a^{147} & a^{45} & a^{198} & a^{21} & a^{228} & a^{186} & a^{231} & a^{56} & a^{68} & a^{81} & a^{46} & a^{32} & a^{60} & a^{225} & a^{13} \\ a^{50} & a^{52} & a^{59} & a^{132} & a^{186} & a^{81} & a^{128} & a^{126} & a^{133} & a^{32} & a^{213} & a^{195} & a^{43} & a^{198} & a^{194} & a^{13} & a^{167} & a^{167} & a^{252} & a^{61} & a^{3} & a^{12} & a^{6} \\ a^{42} & a^{136} & a^{153} & a^{93} & a^{82} & a^{98} & a^{138} & a^{49} & a^{174} & a^{168} & a^{95} & a^{105} & a^{112} & a^{163} & a^{101} & a^{126} & a^{19} & a^{161} & a^{233} & a^{162} & a^{182} & a^{105} & a^{3} \end{bmatrix}$$

Le code C_2

Il s'agit d'un code (32,28,5) sur le corps à 256 éléments.

Si x est un mot de 28 octets le mot de code de C_2 qui lui correspond est le mot de 32 octets égal à $(x, x\,R_2{}^{tr})$, où la matrice (4,28)R_2 est :

$$R_2 = (\,R_1 \mid R'\,) \text{ où } R' = \begin{bmatrix} a^{232} & a^{98} & a^{54} & a^{174} \\ a^{167} & a^{211} & a^{180} & a^{143} \\ a^{24} & a^{41} & a^{188} & a^{164} \\ a^{92} & a^{48} & a^{168} & a^{67} \end{bmatrix}$$

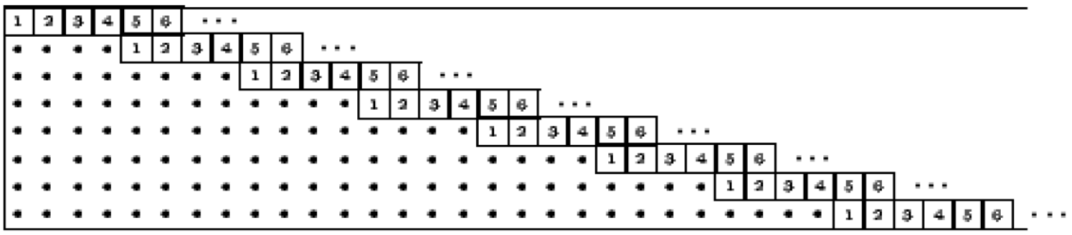


Figure 1: Table d'entrelacement à retard 4 de profondeur 8

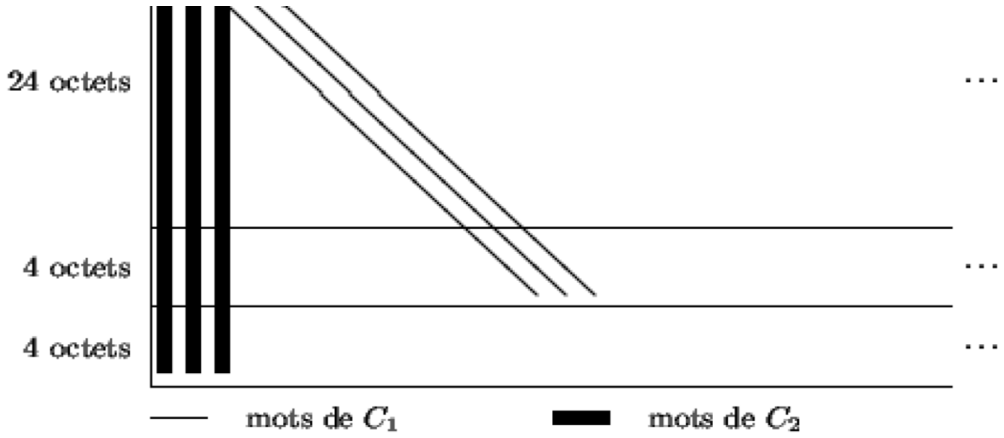


Figure 2: Schéma de codage

Code CIRC (32,24)

- Codage
  - C1(28, 24) : ajout de 4 octets de parité puis entrelacement avec retard 4
  - C2(32,28) : ajout de 4 octets de parité

- Décodage CIRC :
  - Lecture bloc de 32 octets
  - Décodage par C2 : récupération de 28 octets
    - corrige 0 ou 1 erreur.
    - Si 2 erreurs (ou plus), les 28 octets sont effacés
  - Désentrelacement des mots de 28 octets
  - Décodage par C1(28,24) : corrige jusqu'à 4 effacements.

Code CIRC / CD Audio

- Fréquence echantillonage = 44.1 kHz
  - 44100 * 2 (stéréo) * 16 = 1 411 200 bits pour 1 seconde audio
  - Une rayure de 1 mm modifie environ 3300 bits sur la piste
- Bitstream source structuré en trames de 6 échantillons stéréo :
  - Bloc : 1 trame = 6*16*2=192 bits = 24 octets source
- Codage
  - C1(28, 24) : ajout de 4 octets de parité puis entrelacement avec retard 4
  - C2(32,28) : ajout de 4 octets de parité : corrige rafale de 3840 bits
  - 1 octet supplémentaire C&D « Control&Display » => 33 octets
    - (informations pour affichage durée, etc)
- Ecriture sur le disque: codage EFM (Eight-toFourteen Modulation)
  - 1 trame de 192 bits = 33 * (14 + 3 [bits liason]) +27 [bits synchro]=588 bits
  - Au final: 588*44100/6 = 4 321 800 bits « gravés » pour 1 second audio
  - NB un 2ème entrelacement corrige une rafale de 12288 bits par interpolation

TP Codeur / Décodeur CIRC

Plan du cours

- Introduction : Notion de code
- Définition, distance, effacements et erreurs
- Code de Hamming
- Codes détecteurs d' erreur
- Codes correcteurs : Code linéaire, Reed Solomon
- **Autres codes et applications**
  - Rafales d' erreurs – Code déroulé et entrelacement.
  - Code CIRC.
  - **Quelques exemples de codes utlisés dans des matériel.**

Satellites : ex. Voyager

- Photos de Saturne et Jupiter (1977)
  - Données peu critiques (images 800x800 - 8 bits)
  - Données critiques: GSE (General Science&Engineering)
    - Mesures
    - Contrôle
- Données GSE codées par Golay(24,12) sur F_2 (6-correcteur)
- Autres données: code convolutif

GSM

- Signal parole: par tranche de 20 ms
- Codec: Numérisation : 260 bits
  = 50 très critiques + 132 critiques + 78 complémentaires
- Codage de canal : 456 bits
  - » 50 bits : CRC (X3+x+1)
  - » 182+3 : codage convolutif : x2 : 378 bits
  - » +78= 4 bits controles :
- Entrelacement (diagonal : sur plusieurs trames de 456 bits)
- Chiffrement / modulation