RETOUR TP 3 – MATRICES, PIVOT DE GAUSS ET RSA

► **PRÉREQUIS** : *Chapitre 3 du polycopié.*

1 Matrices

On peut construire des matrices et des vecteurs de la façon suivante :

```
A = matrix([[1,2,3],[4,5,6]])
B = matrix(2,3,[1,2,1/2,-1,0,5])
v = vector([0,1,-1])
```

- Comment récupérer les coefficients de `A`, de `B`, de `v` ? Que donnent les commandes `v*A`, `A*v`, `B*v` ? Que donne `A.transpose()` ? Que donnent `A.parent()`, `B.parent()`, `v.parent()` ? Que donnent `B.nrows()` et `B.ncols()` ?

  ► **CORRECTION.–** On utilise les commandes `A[0,0]` ou `A[0][0]`. Attention que la première commande est préférable car elle permet de modifier les coefficients de la matrice `A`. Je vous renvoie au Notebook Jupyter correspondant pour plus de détails et le reste de la correction de cette question !

On peut aussi construire des matrices ou des vecteurs à coefficients dans un anneau donné.

- Essayer les commandes suivantes et préciser les objets qu'elles définissent. On appliquera en particulier la méthode `parent` à chacun d'entre eux :

  ```
  w = vector(CC,[0,5])
  C = matrix(QQ,A)
  D = matrix(GF(5),A)
  M23 = MatrixSpace(RR,2,3); E = M23([1,2,3,4,5,6])
  M4=MatrixSpace(CC,4); F = M4(1); H=M4(0)
  identity_matrix(10)
  ```

  ► **CORRECTION.–** Voir le Notebook Jupyter. Attention que l'espace dans lequel vit une matrice est important ! Si vous ne spécifiez rien, `Sage` voit la matrice `A` comme à coefficient entiers ! Si vous souhaitez alors changer le premier coefficient en $\frac{1}{2}$, vous allez déclencher un message d'erreur car `Sage` s'attend à des entiers comme entrées de la matrice !

On peut enfin construire des matrices dont les coefficients vérifient une formule donnée.

- Essayer les commandes suivantes :

  ```
  G = matrix(QQ,4, lambda i,j: i-j)
  v = vector(RR,[i^2 for i in range(3)])
  ```

- Construire la matrice de Hilbert de taille 5, c'est-à-dire la matrice carrée de taille 5 dont le coefficient en position (i, j) est donné par $\frac{1}{i+j-1}$, vue comme matrice à coefficients rationnels.
- Que donne `G[[0,2,3],1]` ? Que donne `parent` appliqué à cet objet ? Essayer des variantes, comme par exemple `G[[1,0],[0]+[2]]`.

  ► **CORRECTION.–** Voir le Notebook Jupyter.

On donne à présent un certain nombre de commandes `Sage` qu'il peut être utile de connaître et de maîtriser !

- Comment calculer la somme, le produit de deux matrices ? Le déterminant et l'inverse d'une matrice ? Tester les commandes `kernel`, `right_kernel`, `image`, `row_module`, `column_module` sur une matrice. On comparera les résultats pour la matrice `A` ci-dessus et la même matrice vue comme matrice à coefficients rationnels.

  ► **CORRECTION.–** Voir le Jupyter Notebook. Attention que la commande `A.kernel()` fournit le noyau **à gauche**, autrement dit si $A \in \mathcal{M}_{n,m}(A)$, l'ensemble des $X \in \mathcal{M}_{1,n}(A)$ tels que $XA = 0$, soit le noyau (au sens usuel) de la transposée de A ! Il faut donc **impérativement** utiliser `A.right_kernel()` pour obtenir le noyau de `A`. De même, la commande `A.image()` fournit l'image **à gauche**, autrement dit l'espace engendré par les lignes de A, soit l'image (au sens usuel) de la transposée de A ! Il faut donc **impérativement** utiliser `A.column_module()` pour obtenir l'image de `A`.
- Créer une matrice `A` et un vecteur `v` et résoudre le système linéaire `Ax=v` grâce à la commande `A.solve_right()`. Que se passe-t-il si `A` n'est pas injective ? Si l'équation n'a pas de solution ?

  ► **CORRECTION.–** Voir le Jupyter Notebook.

- Soit `A=matrix(QQ,3,[1,2,3,3,2,1,1,1,1])`. Tester :

  ```
  characteristic_polynomial, minimal_polynomial, eigenvalues,
   eigenvectors_right, eigenspaces_right, eigenmatrix_right, jordan_form
  ```

  sur `A`.

  ▶ **Correction.–** Voir le Jupyter Notebook.

- Même question avec `B=matrix(QQ,2,[0,1,2,0])`. Que se passe-t-il ? Réessayer ces commandes en voyant `B` comme une matrice à coefficients dans $K = \mathbf{Q}(\sqrt{2})$.

  ▶ **Correction.–** Voir le Jupyter Notebook.

On termine cette section avec un petit exercice d'application.

- On définit l'espace vectoriel $V = \mathbf{F}_2^4$ et tirer trois vecteurs $u, v, w \in V$ aléatoirement. Comment définir le sous-espace $W = \text{Vect}(u, v, w)$? Comment en donner une base ? Tirer un autre élément aléatoirement dans V et tester s'il appartient à W. Construire le quotient de V par W et donner sa dimension. Construire de même un deuxième sous-espace aléatoire W' à l'aide de deux vecteurs de V et donner une base de $W \cap W'$.

  ▶ **Correction.–** Voir le Jupyter Notebook.

2 Pivot de Gauß

- On se fixe un anneau A. Rappeler le coût, en termes d'opérations dans A et en fonction des dimensions des matrices, des calculs suivants : somme de deux matrices, produit[1] de deux matrices, d'une matrice et d'un vecteur colonne, élévation d'une matrice carrée à une puissance m donnée.

  ▶ **Correction.–** La somme de deux matrices carrées de taille n nécessite n^2 sommes dans A et est donc en $O(n^2)$ tandis que le produit naïf nécessite pour chacun des n^2 coefficients de calculer n sommes et produits donc on obtient du $O(n^3)$. Pour le produit d'une matrice carrée de taille n et d'un vecteur colonne, on a n coefficients à calculer qui nécessite n sommes et produits donc du $O(n^2)$. Pour élever une matrice à la puissance m, on peut effectuer $m-1$ produits en $O(n^3)$ ou utiliser l'exponentiation rapide et effectuer $O(\log_2(m))$ produits en $O(n^3)$ soit du $O(\log_2(m)n^3)$.

2.1 Le cas d'un corps

- Implémenter un algorithme naïf pour calculer le déterminant d'une matrice carrée. Le tester et le comparer à la fonction `det` de `Sage`.

  ▶ **Correction.–** Voir le Notebook Jupyter ! La complexité de l'algorithme utilisant la formule

  $$\det(A) = \sum_{\sigma \in \mathfrak{S}_n} \varepsilon(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

  est de l'ordre de $O(n \times n!)$ puisqu'on fait une somme de $n!$ termes dont chacun nécessite $n-1$ produits. On obtient donc du $O(n \times n!)$, ce qui est énorme ! Pour des matrices de taille 9 ou 10, les temps de calculs deviennent déjà déraisonnables ! Pour la méthode par développement selon une ligne, si l'on pose u_n le temps de calcul dans le pire cas du déterminant d'une matrice carrée de taille n, alors on a

  $$u_{n+1} = (n+1) \times u_n + 2(n+1)$$

  car pour calculer le déterminant on a besoin de faire $n+1$ produits d'un coefficient par le déterminant d'une matrice de taille n et une somme de $n+1$ termes. On montre alors facilement par récurrence que $u_n = O(n!)$. Par ailleurs, cette majoration n'est pas loin de la vérité puisque

  $$u_{n+1} \geqslant (n+1)u_n \geqslant \cdots \geqslant \frac{n!}{2} u_2 = \frac{3}{2} n!.$$

  Pour information, pour une matrice 26×26, un ordinateur capable d'effectuer 10^{12} opérations en virgule flottante par seconde (on parle *d'ordinateur téraflop*) aurait alors besoin de pas moins de 12 700 000 ans de calculs ininterrompus pour sortir le résultat ! Avec l'ordinateur le plus puissant du monde en 2020 qui atteint une puissance maximale de 415,5 pétaflops (soit $415,5 \times 10^{15}$ opérations en virgule flottante à la seconde), il faudrait encore 30 ans et pour une matrice de taille 27, on passe à 832 ans !

- On suppose ici que l'on travaille sur un **corps** k. Implémenter l'algorithme du pivot de Gauß en une fonction `pivot` qui prend en argument une matrice carrée et renvoie cette matrice sous forme échelonnée. Quelle est la complexité de cet algorithme ? Pourquoi a-t-on eu besoin de l'hypothèse que k est un corps ?

  ▶ **Correction.–** L'hypothèse que l'on a un corps est utile pour pouvoir diviser par le pivot non nul ! Le cours fournit une complexité en $O(n^3)$. Dans le cas d'une matrice de taille 26, pour un ordinateur d'un téraflop, on obtient un temps d'exécution de l'ordre de 10^{-8} seconde !

1. On pourra se renseigner sur l'algorithme de Strassen pour une amélioration de cette complexité dans le même veine que Karatsuba.

- Implémenter un algorithme de résolution d'un système linéaire de la forme $Ax = v$ avec A une matrice et v un vecteur. Préciser le coût de cet algorithme.

  ▶ **Correction.–** Voir le Jupyter Notebook! On obtient une complexité en $O(n^3)$.

- Implémenter un algorithme fournissant l'inverse d'une matrice carrée. Préciser le coût de cet algorithme. On suppose qu'on a plusieurs systèmes linéaires de la forme $MX_i = B_i$ à résoudre. À partir de combien de tels systèmes est-il plus rentable de calculer M^{-1} que de résoudre les systèmes un par un ?

  ▶ **Correction.–** Voir le Jupyter Notebook! Une première méthode consiste à rajouter k fois une colonne à la matrice M et à transformer la matrice M en I_n par des opérations du pivot de Gauß, ce qui permet d'obtenir le résultat en $O(kn^3)$. On peut en fait faire un peu mieux en rajoutant k colonnes pour obtenir une matrice de taille $n \times (n+k)$ et faire des opérations du pivot de Gauß pour transformer M en I_n et lire toutes les solutions! Cela se fait en $O(n^2(n+k))$. Si en revanche, on inverse la matrice par le pivot de Gauß, on obtient du $O(n^3)$ puis il faut faire k multiplications de M^{-1} par B_i qui coûtent chacune $O(n^2)$. On obtient donc pour cette seconde méthode une complexité de $O(n^3 + kn^2)$. Cette dernière est donc plus rentable (en négligeant les constantes) dès que

  $$n^3 + kn^2 \leqslant kn^3 \quad \text{soit} \quad \frac{n}{n-1} \leqslant k$$

  par rapport à la méthode naïve mais on retrouve la complexité de la seconde méthode ci-dessus!

- Implémenter un algorithme calculant le déterminant d'une matrice carrée, le rang d'une matrice, une base de son noyau, de son image. Préciser le coût de ces algorithmes.

  ▶ **Correction.–** Voir le Jupyter Notebook! On obtient une complexité en $O(n^3)$. Pour le rang, on met la matrice sous forme échelonnée et prend les colonnes de M correspondant à des sauts verticaux (car ici on fait des opérations sur les lignes sans échanger de colonnes, ce qui fait que la dimension de l'espace engendrée par les k premières colonnes de M est la même que celui engendré par les k premières colonnes de la matrice échelonnée). Par exemple, pour une matrice dont la forme échelonnée est

  $$\begin{pmatrix} 1 & 1 & 0 & -1 & 2 & 1 \\ 0 & 0 & 1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

  on prendra les colonnes 1, 3, 4 et 6 de M qui est de rang 4. En effet, les colonnes 1 et 2 sont liées car leur mise sous forme échelonnée est de rang 1, puis les colonnes 1, 3 et 4 sont libres mais si on rajoute la colonne 5, leur forme échelonnée est de rang 3 donc il n'est pas nécessaire de rajouter cette cinquième colonne. Enfin, on rajoute la colonne 6.

2.2 Un algorithme dans le cas de $\mathbf{Z}$

On donne à présent une méthode alternative pour calculer le déterminant d'une matrice carrée à coefficients entiers.

- Établir *l'inégalité de Hadamard* pour une matrice carrée M à coefficients complexes :

  $$|\det(M)| \leqslant \prod_{i=1}^{n} \left(\sum_{j=1}^{n} |m_{ij}|^2 \right)^{\frac{1}{2}}.$$

  ▶ **Correction.–** L'inégalité est claire si la matrice est de déterminant nul! Supposons donc M inversible. On a alors que ses colonnes $(C_1, \ldots, C_n)$ forment une base de $\mathbf{R}^n$. On peut appliquer le principe d'orthonormalisation de Gram-Schmidt à cette base pour obtenir une base orthonormée $(b_1, \ldots, b_n)$ telle que $\mathrm{Vect}(C_1, \ldots, C_k) = \mathrm{Vect}(b_1, \ldots, b_k)$ pour tout $k \in \{1, \ldots, n\}$. La matrice de passage P de la base $(b_1, \ldots, b_n)$ à $(C_1, \ldots, C_n)$ est par conséquent triangulaire supérieure. Puisque M est la matrice de $(C_1, \ldots, C_n)$ dans la base canonique, si on note Q la matrice de passage de $(e_1, \ldots, e_n)$ à la base $(b_1, \ldots, b_n)$, on a par la formule de changement de base $M = QP$. On conclut en remarquant que $|\det(Q)| = 1$ car une matrice de changement de bases entre deux bases orthonormées est orthogonale. Il vient

  $$|\det(M)| = |\det(P)| = \prod_{i=1}^{n} |p_{ii}|.$$

  Or, on a $p_{ii} = | < b_i, C_i > | \leqslant ||C_i||_2$, ce qui fournit le résultat!
  On suppose désormais que M est à coefficients entiers. Montrer qu'on peut calculer le déterminant de M de la façon suivante et implémenter cet algorithme :

  (i) Trouver des nombres premiers $p_1, \ldots, p_r$ tels que $p_1 \times p_2 \times \cdots \times p_r > 2|\det(M)|$;

(ii) Calculer $\det(M)$ modulo p_i pour tout $1 \leqslant i \leqslant r$ et en déduire $\det(M)$.
▶ **CORRECTION.–** On commence par calculer

$$2\prod_{i=1}^{n}\left(\sum_{j=1}^{n}|m_{ij}|^2\right)^{\frac{1}{2}}. \qquad (*)$$

On énumère alors les nombres premiers dans l'ordre $(p_i)_{i\in\mathbf{N}}$ (dont la liste est stockée par `Sage`) et on calcule successivement

$$\prod_{i=1}^{k} p_i$$

jusqu'à dépasser la borne $(*)$ de sorte que $p_1 \times \cdots \times p_k > 2|\det(M)|$. On calcule alors $|\det(M)|$ modulo p_i pour tout $i \in \{1, \ldots, k\}$ (ce qui permet de se ramener sur un corps et de pouvoir appliquer le pivot de Gauß comme ci-dessus en $O(kn^3)$ en évitant les dénominateurs qui ralentissent le calcul en travaillant sur $\mathbf{Q}$!). Le déterminant de M est alors l'unique entier entre $-\frac{p_1\times\cdots\times p_k}{2}$ et $\frac{p_1\times\cdots\times p_k}{2}$ vérifiant la congruence obtenue modulo p_i. On détermine alors cet entier par le théorème chinois!

2.3 Un algorithme sans division

On ne suppose plus que A est un corps et on suppose seulement qu'il s'agit d'un anneau commutatif[2]. On pose $M^{(0)} = M$, puis pour k allant de 0 à $n-2$, on effectue :

(i) Si les lignes numéro k à $n-1$ de la matrice $M^{(k)}$ sont nulles, l'algorithme est terminé ;

(ii) a) Si la ligne k de la matrice $M^{(k)}$ est nulle mais qu'il existe une ligne de numéro $> k$ non nulle, échanger la ligne k avec une ligne non nulle de numéro $> k$;
b) Si $m_{k,k}^{(k)} = 0$, échanger la colonne k avec une colonne de numéro $j > k$ tel que $m_{k,j}^{(k)} \neq 0$;

(iii) Pour j allant de k à $n-1$, remplacer dans $M^{(k)}$ la ligne L_j par $m_{k,k}^{(k)}L_j - m_{j,k}^{(k)}L_k$;

(iv) La matrice obtenue à la suite de ces opérations est appelée $M^{(k+1)}$.

- Exprimer le déterminant de $M^{(k+1)}$ en fonction de celui de $M^{(k)}$.
▶ **CORRECTION.–** À chaque étape, on obtient $\det\left(M^{(k+1)}\right) = \left(m_{k,k}^{(k)}\right)^{n-k}\det\left(M^{(k)}\right)$ puisqu'on fait l'opération $L_j \leftarrow m_{k,k}^{(k)}L_j - m_{j,k}^{(k)}L_k$ sur chaque ligne de k à $n-1$.
- En déduire un algorithme pour calculer le déterminant de M et l'implémenter.
▶ **CORRECTION.–** Il suffit d'implémenter l'algorithme ci-dessus. On obtient finalement une matrice triangulaire dont le déterminant est le produit des coefficients diagonaux. Pour retrouver le déterminant, on divise ce produit des coefficients diagonaux par

$$\prod_{k=0}^{n-2}\left(m_{k,k}^{(k)}\right)^{n-k}.$$

Voir le Jupyter Notebook.
- En déduire un algorithme pour calcule le polynôme caractéristique d'une matrice M à coefficients dans un corps k et l'implémenter.
▶ **CORRECTION.–** Il suffit d'implémenter l'algorithme ci-dessus sur $k[X]$. Voir le Jupyter Notebook.
- On suppose que les coefficients de M sont des entiers de taille majorée[3] par t. Donner une majoration de la taille des coefficients de $M^{(k)}$ en fonction de ceux de M et faire afficher ces coefficients sur plusieurs exemples. Que constatez-vous ?
▶ **CORRECTION.–** À la première étape, dans le pire des cas, on obtient des coefficients de taille[4] majorée par $2t$, puis à l'étape suivante de taille majorée par $4t$, etc... On voit donc facilement que les coefficients de $M^{(k)}$ sont de taille majorée par $2^k t$. On constate que la taille des entrées peut croître très rapidement et poser des problèmes de mémoire et ralentir les calculs.

La méthode précédente fait apparaître des coefficients qui peuvent devenir très grands. On présente donc une méthode, dite de Gauß-Bareiss, également sans division mais pour laquelle les coefficients croissent moins vite. On fixe une suite (c_k) d'éléments de A non nuls (dépendant de M) et on définit une suite $\left(M^{(k)}\right)$ de matrices comme précédemment, sauf qu'on remplace l'étape **(iii)** par

(iii)' Pour j allant de k à $n-1$, remplacer dans $M^{(k)}$ la ligne L_j par $c_k^{-1}(m_{k,k}^{(k)}L_j - m_{j,k}^{(k)}L_k)$;

2. On pourrait travailler dans le corps de fractions de A lorsque A.
3. Autrement dit, des entiers à moins de t chiffres strictement, donc inférieurs à 10^t.
4. Soit inférieurs à 10^{2t}.

Noter que le cas $c_k = 1$ correspond à l'algorithme précédent et le cas $c_k = m_{k,k}^{(k)}$ au pivot de Gauß. On cherche alors une suite (c_k) telle que $M^{(k)}$ reste à coefficients dans A mais que ces coefficients ne deviennent pas trop gros. On pose $\delta^{[0]}(1,1) = 1$ et

$$\delta^{[k]}(i,j) = \det \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,k} & m_{1,j} \\ m_{2,1} & & \cdots & & m_{2,j} \\ \vdots & \vdots & & \vdots & \vdots \\ m_{k,1} & m_{k,2} & \cdots & m_{k,k} & m_{k,j} \\ m_{i,1} & m_{i,2} & \cdots & m_{i,k} & m_{i,j} \end{pmatrix} = \det\left(D_{i,j}^{[k]}\right), \quad \text{pour} \quad i,j \geqslant k+1.$$

- Soit $N = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ une matrice par blocs avec A, D carrées et A inversible. Montrer que $\det(N) = \det(A)\det(D - CA^{-1}B)$.
  ▶ **Correction.–** On vérifie que
  $$N = \begin{pmatrix} A & 0 \\ C & I \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{pmatrix}$$
  et on utilise le fait que[5]
  $$\det \begin{pmatrix} A & 0 \\ C & I \end{pmatrix} = \det(A) \quad \text{et} \quad \det \begin{pmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{pmatrix} = \det(D - CA^{-1}B).$$
  Le résultat en découle alors immédiatement !
- En déduire que si $i, j > k+1$, on a
  $$\delta^{[k+1]}(i,j)\delta^{[k+1]}(k+1,k+1) = \delta^{[k]}(k+1,k+1)\delta^{[k]}(i,j) - \delta^{[k]}(i,k+1)\delta^{[k]}(k+1,j).$$
  On pose alors $c_k = \delta^{[k]}(k+1,k+1)$ et on note $M^{(k)}$ la matrice obtenue par l'algorithme ci-dessus avec l'étape **(iii)'**.
  ▶ **Correction.–** On suppose $c_k \neq 0$ pour tout k. En utilisant la formule précédente pour $\delta^{[k]}(i,j)$ avec A le mineur en haut à gauche de taille k, il vient
  $$\delta^{[k]}(i,j) = \delta^{[k-1]}(k,k)\left(m_{i,j} - {}^t a \left(D_{k,k}^{[k-1]}\right)^{-1} b\right)$$
  avec ${}^t a = (m_{i,1}, \ldots, m_{i,k})$ et ${}^t b = (m_{1,j}, \ldots, m_{k,j})$. Par ailleurs, toujours en utilisant la formule précédente avec la matrice
  $$\begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,k} & m_{1,k+1} & m_{1,j} \\ m_{2,1} & & \cdots & & m_{2,k+1} & m_{2,j} \\ \vdots & \vdots & & \vdots & \vdots & \\ m_{k,1} & m_{k,2} & \cdots & m_{k,k} & m_{k,k+1} & m_{k,j} \\ m_{k+1,1} & m_{k+1,2} & \cdots & m_{k+1,k} & m_{k+1,k+1} & m_{k+1,j} \\ m_{i,1} & m_{i,2} & \cdots & m_{i,k} & m_{i,k+1} & m_{i,j} \end{pmatrix},$$
  il vient
  $$\delta^{[k+1]}(i,j) = \delta^{[k-1]}(k,k) \det \left(\begin{pmatrix} m_{k+1,k+1} & m_{k+1,j} \\ m_{i,k+1} & m_{i,j} \end{pmatrix} - \begin{pmatrix} m_{k,1} & m_{k,2} & \cdots & m_{k,k} \\ m_{i,1} & m_{i,2} & \cdots & m_{i,k} \end{pmatrix} \left(D_{k,k}^{[k-1]}\right)^{-1} \begin{pmatrix} m_{1,k+1} & m_{1,j} \\ m_{2,k+1} & m_{2,j} \\ \vdots & \vdots \\ m_{k,k+1} & m_{k,j} \end{pmatrix} \right).$$
  Or, on constate que les coefficients de
  $$\begin{pmatrix} m_{k+1,k+1} & m_{k+1,j} \\ m_{i,k+1} & m_{i,j} \end{pmatrix} - \begin{pmatrix} m_{k,1} & m_{k,2} & \cdots & m_{k,k} \\ m_{i,1} & m_{i,2} & \cdots & m_{i,k} \end{pmatrix} \left(D_{k,k}^{[k-1]}\right)^{-1} \begin{pmatrix} m_{1,k+1} & m_{1,j} \\ m_{2,k+1} & m_{2,j} \\ \vdots & \vdots \\ m_{k,k+1} & m_{k,j} \end{pmatrix}$$
  sont de la forme $m_{i,j} - {}^t a \left(D_{k,k}^{[k-1]}\right)^{-1} b$ avec ${}^t a = (m_{i,1}, \ldots, m_{i,k})$ et ${}^t b = (m_{1,j}, \ldots, m_{k,j})$ et $(i,j) \in \{(k+1,k+1), (k+1,j), (i,k+1), (i,j)\}$. On en déduit que
  $$\delta^{[k-1]}(k,k) \left(\begin{pmatrix} m_{k+1,k+1} & m_{k+1,j} \\ m_{i,k+1} & m_{i,j} \end{pmatrix} - \begin{pmatrix} m_{k,1} & m_{k,2} & \cdots & m_{k,k} \\ m_{i,1} & m_{i,2} & \cdots & m_{i,k} \end{pmatrix} \left(D_{k,k}^{[k-1]}\right)^{-1} \begin{pmatrix} m_{1,k+1} & m_{1,j} \\ m_{2,k+1} & m_{2,j} \\ \vdots & \vdots \\ m_{k,k+1} & m_{k,j} \end{pmatrix} \right) = \begin{pmatrix} \delta^{[k]}(k+1,k+1) & \delta^{[k]}(k+1,j) \\ \delta^{[k]}(i,k+1) & \delta^{[k]}(i,j) \end{pmatrix}.$$

5. Pour le démontrer, on peut développer successivement par rapport aux dernières colonnes dans le premier cas et par rapport aux premières colonnes dans le second cas.

En prenant les déterminants, il vient

$$\left(\delta^{[k-1]}(k,k)\right)^2 \det\left(\begin{pmatrix} m_{k+1,k+1} & m_{k+1,j} \\ m_{i,k+1} & m_{i,j} \end{pmatrix} - \begin{pmatrix} m_{k,1} & m_{k,2} & \cdots & m_{k,k} \\ m_{i,1} & m_{i,2} & \cdots & m_{i,k} \end{pmatrix} \left(D_{k,k}^{[k-1]}\right)^{-1} \begin{pmatrix} m_{1,k+1} & m_{1,j} \\ m_{2,k+1} & m_{2,j} \\ \vdots & \vdots \\ m_{k,k+1} & m_{k,j} \end{pmatrix}\right)$$
$$= \delta^{[k]}(k+1,k+1)\delta^{[k]}(i,j) - \delta^{[k]}(k+1,j)\delta^{[k]}(i,k+1)$$

ce qui fournit (enfin !) le résultat souhaité !

- Montrer que l'algorithme est bien défini et que $M^{(k)}$ est bien à coefficients dans A. On pourra vérifier que pour $i,j > k$, on a $M^{(k)}[i,j] = \delta^{[k]}(i,j)$.
  ▶ **CORRECTION.–** On a à l'étape k que $M^{(k+1)}[i,j] = M^{(k)}[i,j]$ si i ou $j \leqslant k$. Si $i,j > k$, on a

$$M^{(k+1)}[i,j] = \frac{M^{(k)}[k+1,k+1]M^{(k)}[i,j] - M^{(k)}[i,k+1]M^{(k)}[k+1,j]}{c_k}$$

  et on conclut par récurrence et d'après la question précédente que pour $i,j > k$, on a $M^{(k)}[i,j] = \delta^{[k]}(i,j)$. On a donc bien que

$$M^{(k+1)}[i,j] = \delta^{[k]}(i,j) \in A$$

  et on a le résultat. Noter que si l'un des c_k s'annule, alors l'étape **(ii)** de l'algorithme cherche un autre pivot non nul dans les lignes en-dessous de k. Si on en trouve une, on va remplacer notre coefficient $m_{k,k}^{(k-1)}$ qui était nul par un coefficient non nul et on va continuer l'algorithme. Sinon, c'est que l'algorithme est terminé ! On a alors à chaque étape

$$\det(M^{(k+1)}) = \frac{c_k^{n-k}}{c_{k-1}^{n-k}} \det(M^{(k)}),$$

  mais surtout $c_{n-1} = \det(M)$, ce qui permet d'implémenter un nouvel algorithme de calcul sans division où on lit le déterminant comme le coefficient (n,n) de la matrice $M^{(n-1)}$! Voir le Jupyter Notebook.
- Montrer que pour calculer c_k, on n'a en fait pas besoin de calculer $\delta^{[k]}(k+1,k+1)$. Implémenter cet algorithme pour calculer le déterminant et le polynôme caractéristique d'une matrice.
  ▶ **CORRECTION.–** On a en effet que $c_0 = 1$, $c_1 = m_{1,1}$ (le premier pivot) puis

$$c_k = \frac{c_{k-1}m_{k+1,k+1}^{(k-1)} - m_{k+1,k}^{(k-1)}m_{k,k+1}^{(k-1)}}{c_{k-2}}$$

  que l'on peut obtenir par division euclidienne dans $\mathbf{Z}$ ou $k[X]$ par exemple puisqu'on a établi que la division était exacte !
- On suppose ici que $A = \mathbf{Z}$ et que tous les coefficients de M sont de taille majorée par t. Montrer que tous les coefficients de $M^{(k)}$ ont une taille majorée par $n\left(\frac{1}{2}\log_{10}(n) + t\right) + 1$. Quel est le coût de l'algorithme précédent ?
  ▶ **CORRECTION.–** On utilise le fait que les coefficients de $M^{(k)}$ sont des mineurs extraits de M de sorte que l'inégalité de Hadamard établie ci-dessus fournit

$$\left|m_{i,j}^{(k)}\right|^2 \leqslant \prod_{i=1}^{k+1}\left(\sum_{j=1}^{k+1} 10^{2t}\right) = \left(k10^{2t}\right)^k,$$

  ce qui fournit le résultat. En ce qui concerne la complexité, pour passer de $M^{(k-1)}$ à $M^{(k)}$, dans le cas du pivot de Gauß classique, on utilise pour chacune des $n-k$ lignes en dessous de la ligne k, pour calculer chacun des nouveaux $n-k+1$ coefficients de ces lignes, une multiplication par une fraction et une soustraction contre deux multiplications, une soustraction et une division euclidienne. Il s'ensuit, puisque

$$\sum_{k=1}^{n}(n-k+1)^2 = \frac{n(n-1)(2n-1)}{6} \underset{n\to+\infty}{\sim} \frac{n^3}{3},$$

  dans les deux cas une complexité en $O(n^3)$ opérations arithmétiques avec une constante légèrement moins bonne dans le cas de Gauß-Bareiss.

2.4 Compléments sur le polynôme caractéristique

- Justifier que la connaissance de $\det(M - xI_n)$ pour $x \in \{0,\ldots,n-1\}$ permet de déterminer χ_M. Implémenter cet algorithme et en préciser le coût.

▶ **Correction.–** Puisque le degré du polynôme caractéristique est n, si on construit le polynôme d'interpolation de Lagrange [6] associés aux valeurs obtenues, on obtient deux polynômes de degré n ayant $n+1$ racines en commun donc ces deux polynômes sont égaux! Voir le Jupyter Notebook! Pour la complexité, il suffit de calculer n déterminants en utilisant le pivot de Gauß, ce qui fournit une complexité en $O(n \times n^3) = O(n^4)$. Puis pour le polynôme interpolateur de Lagrange, on peut effectuer le calcul en $O(n^2)$ et je vous renvoie pour cela au partiel de 2019. Ainsi, on obtient finalement un $O(n^4)$. Noter qu'en comparaison, le pivot de Gauß ou de Gauß-Bareiss sans division nécessitent $O(n^3)$ opérations dans $k[X]$, certaines de ces opérations correspondant à des multiplications de polynômes de taille bornée par n, ce qui le rend moins efficace en pratique que cette méthode!

- Rappeler les relations entre les sommes de Newton et les fonctions symétriques élémentaires d'une famille de n éléments de A. Expliquer comment calculer χ_M à partir des traces des M^i pour $i \in \{1, \dots, n\}$. Implémenter cet algorithme [7] et en préciser le coût.
▶ **Correction.–** On rappelle que si on se donne une famille $\{x_1, \dots, x_n\}$ à n éléments de A, alors les sommes de Newton associées sont pour tout $k \geqslant 1$ les quantités

$$N_k(\mathbf{x}) = \sum_{i=1}^{n} x_i^k$$

tanids que les polynômes symétriques élémentaires sont définis pour tout $k \in \{1, \dots, n\}$ par

$$\sigma_k(\mathbf{x}) = \sum_{1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n} x_{i_1} x_{i_2} \cdots x_{i_k}.$$

ON rappelle que si l'on pose $P = \prod_{i=1}^{n} (X - x_i) \in A[X]$, alors

$$P = X^n + \sum_{i=1}^{n} (-1)^i \sigma_i X^{n-i}.$$

On a alors pour $1 \leqslant i \leqslant n$ et $p \leqslant n$,

$$x_i^n - \sigma_1(\mathbf{x}) x_i^{n-1} + \sigma_2(\mathbf{x}) x_i^{n-2} - \cdots + (-1)^n \sigma_n(\mathbf{x}) = 0.$$

Multipliant le tout par x_i^{p-n}, il vient

$$x_i^p - \sigma_1(\mathbf{x}) x_i^{p-1} + \sigma_2(\mathbf{x}) x_i^{p-2} - \cdots + (-1)^n \sigma_n(\mathbf{x}) x^{n-p} = 0,$$

et après sommation

$$N_p(\mathbf{x}) - \sigma_1(\mathbf{x}) N_{p-1}(\mathbf{x}) + \sigma_2(\mathbf{x}) N_{p-2}(\mathbf{x}) - \cdots + (-1)^n \sigma_n(\mathbf{x}) N_{p-n}(\mathbf{x}) = 0. \qquad (*)$$

Lorsque $1 \leqslant p \leqslant n-1$, on constate que

$$N_p(\mathbf{x}) = \sigma_1(\mathbf{x}) N_{p-1}(\mathbf{x}) - \sum_{\substack{1 \leqslant i_1, i_2 \leqslant n \\ i_1 \neq i_2}} x_{i_1} x_{i_2}^{p-1}.$$

Puis, de même

$$\sigma_2(\mathbf{x}) N_{p-2}(\mathbf{x}) = \sum_{\substack{1 \leqslant i_1, i_2 \leqslant n \\ i_1 \neq i_2}} x_{i_1} x_{i_2}^{p-1} + \sum_{\substack{1 \leqslant i_1 < i_2 \leqslant n \\ 1 \leqslant i_3 \leqslant n \\ i_3 \neq i_1, i_2}} x_{i_1} x_{i_2} x_{i_3}^{p-2}.$$

On constate alors que l'on va avoir si l'on pose

$$S_k = \sum_{\substack{1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n \\ 1 \leqslant i_{k+1} \leqslant n \\ i_{k+1} \neq i_1, i_2, \dots, i_k}} x_{i_1} x_{i_2} \cdots x_{i_k} x_{i_{k+1}}^{p-k},$$

que $S_0 = N_p(\mathbf{x})$, que $S_{p-1} = p\sigma_p(\mathbf{x})$ et que

$$\sigma_k(\mathbf{x}) N_{p-k}(\mathbf{x}) = S_{k-1} + S_k.$$

6. On rappelle que si on connaît $\chi_M(0), \dots, \chi_M(n-1)$, alors le polynôme interpolateur de Lagrange correspondant est donné par

$$\sum_{i=0}^{n} \chi_M(i) \prod_{\substack{k=0 \\ k \neq i}}^{n} \left(\frac{x-k}{i-k} \right).$$

7. Dit de Le Verrier.

Il suffit alors de multiplier la première de ces équations par -1, la seconde par $(-1)^2$, jusqu'à la $(p-1)$-ème par $(-1)^{p-1}$ pour obtenir que

$$\sum_{k=1}^{p-1}(-1)^k\sigma_k(\mathbf{x})N_{p-k}(\mathbf{x}) = -S_0 + (-1)^{p-1}S_{p-1} = -N_p(\mathbf{x}) + (-1)^{p-1}\sigma_p(\mathbf{x}).$$

Ainsi, pour tout $1 \leqslant p \leqslant n-1$,

$$\mathbb{N}_p(\mathbf{x}) - \sigma_1(\mathbf{x})N_{p-1}(\mathbf{x}) + \sigma_2(\mathbf{x})N_{p-2}(\mathbf{x}) - \cdots + (-1)^{p-1}\sigma_{p-1}(\mathbf{x})S_1(\mathbf{x}) + (-1)^p p\sigma_p(\mathbf{x}) = 0 \qquad (**)$$

Cela permet, si l'on connaît les n premières sommes de Newton de calculer les fonctions symétriques élémentaires récursivement. Je vous renvoie au tome d'algèbre 1 du Francinou–Gianella–Nicolas pour une autre preuve un peu plus élégante mais plus astucieuse ! Il suffit alors de constater (par exemple en trigonalisant sur une clôture algébrique) que les sommes de Newton des valeurs propres d'une matrice M sont données par les $\mathrm{Tr}(M^i)$ pour $i \in \{1, \ldots, n\}$. On calcule alors ces puissances successives (noter qu'ici une exponentiation rapide n'aurait pas beaucoup de sens puisqu'on a besoin de **toutes** les puissances) à l'aide de n multiplications qui coûtent [8] chacune $O(n^3)$. On obtient ainsi du $O(n^4)$ à nouveau. Il faut alors à partir de ces sommes de Newton reconstruire les coefficients du polynôme caractéristique, ce qui se fait en $O(n^2)$ opérations [9] et donc on a une complexité totale de $O(n^4)$ à nouveau.

3 RSA

Tout message est représenté [10] par votre ordinateur comme une suite de *bits* (donc de 0 et de 1). Tout message correspond donc à un entier. Si on se donne une borne sur le nombre de bits, cela borne l'entier correspondant. Si on choisit alors dans RSA un entier $n = pq$ tel que tous les messages soit inférieurs ou égaux à n, alors connaître un message m modulo n ou connaître m revient au même.

3.1 RSA

- Écrire un programme qui prend en entrée la clé publique (n, e) d'un cryptosystème RSA et un message en clair m, puis renvoie le message chiffré $m^e \mod n$.
  ▶ **Correction.–** Voir le Notebook !
- Écrire un programme de déchiffrement, connaissant les facteurs premiers de n.
  ▶ **Correction.–** Voir le Notebook ! Si l'on connaît les facteurs premiers de n, on peut calculer $\varphi(n)$ et donc l'inverse de e modulo $\varphi(n)$ par l'algorithme d'Euclide étendu ! Il suffit alors de calculer c^d modulo n (où c est le message chiffré) pour retrouver le message initial m.
- On peut utiliser le théorème chinois pour accélérer le calcul de $c^d \pmod n$ (où c est le message chiffré, d l'exposant de déchiffrement), en calculant $c^{d(\mathrm{mod}\ p-1)} \pmod p$ et $c^{d(\mathrm{mod}\ q-1)} \pmod q$. Justifier cette approche et l'implémenter.
  ▶ **Correction.–** Voir le Notebook ! Si on connaît $c^{d(\mathrm{mod}\ p-1)} \pmod p$, puisque $c^{p-1} \equiv 1$ si $c \not\equiv 0 \pmod p$ et $c^{p-1} \equiv 0 \pmod p$ sinon, on connaît $c^d \pmod p$ et de même on connaît $c^d \pmod q$. Le théorème chinois garantit alors que l'on connaît $c^d \pmod{pq}$. Cette méthode permet de faire des calculs de déchiffrement plus rapidement modulo p et q sont tous les deux de grands nombres premiers et sont par conséquent de taille bien plus petite que n. Noter que le déchiffrement classique (via exponentiation rapide) a un coût en $O(\log(d)\log(n)^2)$.
- Comparer les temps d'exécution en faisant varier e (pour n fixé).
  ▶ **Correction.–** Voir le Notebook !

8. En fait $O(n^{2.18})$ par Strassen.

9. Noter qu'en utilisant une stratégie *baby step-giant step*, c'est-à-dire en écrivant pour $q = E(\sqrt{n})$, $k = aq + r$ la division euclidienne de k par q, on a alors $A^k = (A^q)^a A^r$ et on a seulement besoin d'effectuer $O(\sqrt{n})$ multiplications de matrices (on calcule tous les A^r pour $0 \leqslant r < q$, A^q par exponentiation rapide et ensuite $a \leqslant \sqrt{n}$). Cela permet déjà d'obtenir une complexité meilleure de l'ordre de $O\left(n^{\frac{1}{2}+3}\right)$. Et on peut même faire mieux en utilisant le fait que l'on ne cherche que les coefficients diagonaux, et calculer les traces des puissances successives en $O(n^2)$

10. Voir par exemple la table ASCII suivante

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	`	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	-	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

3.2 Quelques attaques

- Supposons que $n = pq$ avec p et q "proches". Montrer que l'on peut écrire $n = r^2 - s^2$ avec r de l'ordre de grandeur de $\sqrt{n}$ (et donc s petit). Expliquer comment déterminer facilement si un entier est le carré d'un entier et en déduire un programme qui prend en entrée n et renvoie (p, q) en temps rapide si p et q sont proches.
  ▶ **CORRECTION.–** Il suffit de voir que dans ce cas $n = (r - s)(r + s) = pq$ et donc si $p < q$, on a

  $$r = \frac{p+q}{2} \quad s = \frac{q-p}{2}.$$

  Ces deux quantités ayant bien un sens puisqu'on choisit toujours p et q grands donc impairs tous les deux. Puisque p et q sont de taille similaire, on a bien $n = pq \approx p^2 \approx q^2$ soit $p \approx q \approx \sqrt{n}$ donc $r \approx \sqrt{n}$ et s très petit. On remarque que $r^2 \geqslant n$. On peut donc initialiser r à la partie entière de $\sqrt{n}$ plus 1. Puis on a que $r + s = p < n$ donc on peut calculer pour tous les s tels que $r + s < n$, $r^2 - s^2$ et vérifier si on retombe sur n ou non. Si ce n'est pas le cas, on incrémente r et si on a trouvé une solution on s'arrête ! On a alors que $r < r + s < n$ et donc on sait qu'il faut incrémenter r au plus jusqu'à n. On obtient alors (en cas de succès)

  $$p = r + s \quad \text{et} \quad q = r - s.$$

- Écrire un programme permettant de retrouver un message en clair m en prenant en entrée ses chiffrés pour deux clés publiques (n, e_1) et (n, e_2) où e_1, e_2 sont premiers entre eux.
  ▶ **CORRECTION.–** Ici, il suffit de voir que les deux chiffrés sont $m^{e_1} \pmod n$ et $m^{e_2} \pmod n$ et que par Bézout, on a deux entiers u et v tels que $e_1 u + e_2 v = 1$ et donc

  $$m = (m^{e_1})^u (m^{e_2})^v \pmod n.$$

- Bob a choisi la clé publique $(n, 3)$. Alice envoie à Bob le chiffré c du message m et le chiffré c' du message $m + r$. Exprimer $c' - c + 2r^3$ et $c' + 2c - r^3$ en fonction de m et r. Oscar intercepte c et c'. On suppose qu'il connaît r et que $c' - c + 2r^3$ est inversible modulo n. Comment peut-il obtenir m en temps polynomial ? Retrouver m lorsque $n = 2173$, $r = 1$, $c = 793$ et $c' = 2083$.
  ▶ **CORRECTION.–** Il est fréquent de vouloir choisir e petit pour éviter trop de calculs lors du chiffrement. Ici, on va voir que $e = 3$ présente des risques ! je vous renvoie par exemple aux attaques de Håstad et de Coppersmith. On prend souvent en pratique $e = 2^{16} + 1$ (le quatrième nombre premier de Fermat) qui est (sauf exception rare) premier avec $\varphi(n)$. On a alors que $\log(e)$ est une constante et le chiffrement a alors un coût en $O(\log(n)^2)$.
  Notons que $c \equiv m^3 \pmod n$ et $c' \equiv (m + r)^3 \pmod n$. On a donc

  $$\begin{cases} c \equiv m^3 & \pmod n \\ c' \equiv m^3 + 3m^2 r + 3mr^2 + r^3 & \pmod n \end{cases}$$

  soit

  $$\begin{cases} c' - c + 2r^3 \equiv 3r(m^2 + mr + r^2) & \pmod n \\ c' + 2c - r^3 \equiv 3m(m^2 + mr + r^2) & \pmod n. \end{cases}$$

  Si l'on suppose que $c' - c + 2r^3$ est inversible modulo n, alors en notant ℓ l'inverse de $c' - c + 2r^3$ modulo n, il vient que $m \equiv r(c' + 2c - r^3)\ell \pmod n$. On obtient $m = 1678$!

On présente à présent l'attaque de Wiener. Soit $n = pq$ avec p et q premiers tels que $p \in]q, 2q[$.

- Montrer que si, pour une clé RSA publique (n, e), on a $d < \frac{1}{3}\sqrt[4]{n}$, alors un attaquant peut calculer d en temps polynomial à partir de n et e. On pourra utiliser pour cela le théorème suivant :
  Théorème 1.– *Soient $\frac{a}{b}$ et $\frac{c}{d}$ deux fractions sous forme irréductible telles que $\left|\frac{a}{b} - \frac{c}{d}\right| \leqslant \frac{1}{2d^2}$. Alors $\frac{c}{d}$ est une des réduites du développement de $\frac{a}{b}$ en fractions continues.*
  ▶ **CORRECTION.–** Ici, on montre que l'on ne peut pas choisir d trop petit (ce qui permettrait d'accélérer le déchiffrement) sous peine de rendre possible une attaque ! Je vous renvoie au Bonus du TP 2 pour des compléments sur les développements en fractions continues. Je rappelle simplement que pour un nombre rationnel q, si l'on note $\frac{h_1}{k_i}$ les réduites du développement en fractions continues, on a a alors $\left|q - \frac{h_i}{k_i}\right| < \frac{1}{k_i^2}$. Le théorème [11] admis est donc une réciproque partielle de ce résultat. Supposons que $d < \frac{1}{3}\sqrt[4]{n}$. On sait, par définition, que $de = 1 + k\varphi(n)$ pour un certain entier k. On a alors

  $$\left|\frac{e}{n} - \frac{k}{d}\right| = \left|\frac{1 + k\varphi(n) - kn}{dn}\right|.$$

  Maintenant, $n = pq$ donc $\varphi(n) = n - (p + q) + 1$ donc

  $$\left|\frac{e}{n} - \frac{k}{d}\right| = \frac{k(p + q - 1) - 1}{dn} \leqslant \frac{k(p + q - 1)}{dn}.$$

11. Voir par exemple l'Appendix 4 du *Number Theory, Quantum Computation and Quantum Information* de Nielsen et Chuang.

Mais, $p+q \leqslant 3q < 3\sqrt{n}$ car $n = pq > q^2$. Par ailleurs, $e \leqslant \varphi(n)$ donc $k\varphi(n) \leqslant d\varphi(n)$ et par conséquent $k \leqslant d$. Il s'ensuit que

$$\left|\frac{e}{n} - \frac{k}{d}\right| \leqslant \frac{3}{\sqrt{n}}.$$

Enfin, l'hypothèse que $d < \frac{1}{3}\sqrt[4]{n}$ implique que $d^2 \leqslant \frac{1}{9}\sqrt{n}$ et finalement

$$\left|\frac{e}{n} - \frac{k}{d}\right| \leqslant \frac{1}{3d^2}.$$

Le théorème admis garantit alors que $\frac{k}{d}$ est une réduite du développement en fractions continues de $\frac{e}{n}$. Comme n et e sont publiques, il suffit pour un attaquant de calculer les réduites successives du développement en fractions continues. Pour une réduite donnée $\frac{k_i}{d_i}$, on calcule alors u tel que $d_i e = 1 + k_i u$ et u est alors le candidat $\varphi(n)$. On regarde alors si le système

$$\begin{cases} u = n - (p+q) + 1 \\ n = pq \end{cases}$$

admet des solutions entières (en résolvant une équation du second degré). Si oui on a factorisé n et donc l'attaque est réussi ! Ce qui précède garantit que si $d < \frac{1}{3}\sqrt[4]{n}$, cela fonctionnera nécessairement avec une des réduites ! ON peut alors effectuer cela avec une complexité $O(\log(n)^3)$ puisqu'on a $\log(n)$ réduites que l'on peut calculer par l'algorithme d'Euclide (*cf.* TP 2).

- Pour éviter cette attaque, on donne (n, e') comme clé RSA publique où $e' = e + t\varphi(n)$ avec t grand. Montrer que si $e' > n\sqrt{n}$, alors même si $d < \frac{1}{3}\sqrt[4]{n}$, l'attaque ci-dessus ne peut pas être effectuée.
  ▶ **Correction.–** Dans ce cas, on a si $de' = 1 + k'\varphi(n)$,

$$\left|\frac{e'}{n} - \frac{k'}{d}\right| = \frac{k'(p+q-1)-1}{dn}.$$

Or, $p > q$ et on peut supposer que $p > q + 2$ car sinon $p \approx q$ et on est vulnérable à une attaque comme vu ci-dessus. On a donc

$$\left|\frac{e'}{n} - \frac{k'}{d}\right| \geqslant \frac{k'(p+q-2)}{dn} \geqslant \frac{q}{\sqrt{n}}$$

car $2k\varphi(n) \geqslant 1 + k\varphi(n) = de' > dn\sqrt{n}$. On utilise alors que $\varphi(n) \leqslant n$ et que $p > q+2$. Enfin, $n = pq \leqslant 2q^2$ donc $q \geqslant \frac{\sqrt{n}}{\sqrt{2}}$. Ainsi,

$$\left|\frac{e'}{n} - \frac{k'}{d}\right| \geqslant \frac{1}{\sqrt{2}} > \frac{1}{d^2}$$

dès que $d > 4$ (ce qu'il est raisonnable de supposer sinon il est facile d'attaquer). On ne peut donc pas avoir que $\frac{k'}{d}$ est une réduite de $\frac{e'}{d}$ d'après le résultat rappelé en question précédente !
On constate donc que prendre d petit (et améliorer le déchiffrement) se fait alors au détriment du chiffrement (pusiqu'il faut alors prendre e grand).

- En déduire un programme qui prend en entrée (n, e) et renvoie d en temps rapide si $d < \frac{1}{3}\sqrt[4]{n}$.
  ▶ **Correction.–** Voir le Notebook !

3.3 Factorisation

- Écrire un programme qui prend en entrée n et k, puis renvoie `Echec` si $k \neq \varphi(n)$ ou si n n'est pas le produit de deux nombres premiers, et renvoie les deux facteurs premiers de n sinon.
  ▶ **Correction.–** On calcule facilement (en faisant n fois un algorithme d'Euclide pour tester si $\ell \leqslant n$ est premier à n ou non), $\varphi(n)$ si n est connu. On peut alors vérifier si $k = \varphi(n)$. Si on a trouvé $\varphi(n)$, on obtient p et q à partir de $\varphi(n)$ et de n comme dans le cours. Une fois nos deux nombres premiers déterminés, on vérifie si leur produit vaut bien n. Si c'est le cas, on a factorisé n et on a gagné et si ce n'est pas le cas, c'est que n n'était pas le produit de deux nombres premiers ! Voir le Notebook !

Supposons que $n = pq$. On note e, premier avec $\varphi(n)$, l'exposant public d'un système RSA de module n.

- Montrer comment, à partie de e, d et n, on peut construire un multiple B de $\varphi(n)$.
  ▶ **Correction.–** On sait que $ed \equiv 1 \pmod{\varphi(n)}$ donc $B = ed - 1$ est un multiple de $\varphi(n)$.
- On note $m = \text{ppcm}(p-1, q-1)$. Montrer que pour tout $a \in (\mathbf{Z}/n\mathbf{Z})^\times$, on a $a^m = 1$. Montrer que $a^{m/2}$ peut prendre quatre valeurs et que deux de ces valeurs permettent de factoriser n.

▶ **Correction.–** On a pour tout $a \in (\mathbf{Z}/n\mathbf{Z})^\times$, que $a^m \equiv 1 \pmod p$ et $a^m \equiv 1 \pmod q$ car $p-1, q-1 \mid m$. On en déduit par le théorème chinois que $a^m \equiv 1 \pmod n$. On a que m est pair (car p et q sont impairs) et donc

$$\begin{cases} \left(a^{\frac{m}{2}}\right)^2 \equiv 1 \pmod p \\ \left(a^{\frac{m}{2}}\right)^2 \equiv 1 \pmod q . \end{cases}$$

Or, sur les corps $\mathbf{Z}/p\mathbf{Z}$ et $\mathbf{Z}/q\mathbf{Z}$, l'équation $X^2 - 1$ n'a que deux solutions ± 1. On a donc que

$$\begin{cases} \left(a^{\frac{m}{2}}\right)^2 \equiv \varepsilon_p \pmod p \\ \left(a^{\frac{m}{2}}\right)^2 \equiv \varepsilon_q \pmod q \end{cases} \quad \text{pour} \quad \varepsilon_p, \varepsilon_q \in \{\pm 1\}.$$

Par le théorème chinois, cela fournit 4 valeurs modulo n pour $aa^{\frac{m}{2}}$. Il est clair que si $\varepsilon_p = \varepsilon_q = 1$, alors $a = 1$ est l'unique solution modulo n fournie par le théorème chinois et que si $\varepsilon_p = \varepsilon_q = -1$, alors $a = 1$ est l'unique solution modulo n fournie par le théorème chinois. Supposons alors (le dernier cas étant analogue par symétrie) que $\varepsilon_p = 1$ et que $\varepsilon_q = -1$. Notons x la solution modulo n fournie par le théorème chinois (le cas symétrique correspondant alors à $-x$). On constate alors pgcd$(n, x-1)$ fournit un facteur non trivial et permet de factoriser n!

- On pose H l'ensemble des éléments de $(\mathbf{Z}/n\mathbf{Z})^\times$ tels que $a^{m/2} = \pm 1 \mod n$. Montrer qu'il s'agit d'un sous-groupe de $(\mathbf{Z}/n\mathbf{Z})^\times$.
  ▶ **Correction.–** Le fait que ce soit un sous-groupe est clair.
- Montrer qu'il existe $b \in (\mathbf{Z}/n\mathbf{Z})^\times$ tel que b soit d'ordre $p-1$ modulo p et d'ordre $\frac{q-1}{2}$ modulo q.
  ▶ **Correction.–** Soit ω_p un générateur du groupe des inversibles modulo p (donc d'ordre $p-1$ modulo p) et ω_q un générateur du groupe des inversibles modulo q (donc d'ordre $q-1$ modulo q). Alors, ω_q^2 est d'ordre $\frac{q-1}{2}$ et l'unique solution modulo n au système $b \equiv \omega_p \pmod p$ et $b \equiv \omega_q^2 \pmod q$ convient alors.
- On pose $p - 1 = 2^{\nu_p} p'$ et $q - 1 = 2^{\nu_q} q'$ avec p', q' impairs et on suppose, sans perte de généralité, que $\nu_p \geqslant \nu_q$. Exprimer $\frac{m}{2}$ en fonction de ν_p et du ppcm de p' et q'. En déduire que b n'appartient pas à H. Si l'on prend x au hasard dans $(\mathbf{Z}/n\mathbf{Z})^\times$, montrer que la probabilité que x n'appartienne pas à H est supérieure ou égale à $\frac{1}{2}$.
  ▶ **Correction.–** On a immédiatement que $m = 2^{\nu_p}\text{ppcm}(p', q')$ donc $\frac{m}{2} = 2^{\nu_p - 1}\text{ppcm}(p', q')$. On a que $b^{m/2} \equiv \omega_p^{m/2} \pmod p$. Or, $p - 1 \nmid \frac{m}{2}$, donc $b^{m/2} \equiv -1 \pmod p$. En revanche, on a que $b^{m/2} \equiv \omega_q^m \equiv 1 \pmod q$ car $q - 1 \mid m$. On est donc bien dans le cas de figure où $b^{m/2} \neq \pm 1 \pmod n$. On a donc que H est un sous-groupe dont le complémentaire est non trivial. On en déduit que $\#H < \#G$. On a alors que si x est choisi au hasard, la probabilité que x appartienne à H est donnée par $\frac{\#H}{\#G} \leqslant \frac{1}{2}$ car $\#H$ est un diviseur de $\#G$ et donc $\#G/\#H$ est un entier supérieur ou égal à 2! On en déduit la probabilité du complémentaire qui est $1 - \frac{1}{[G:H]} \geqslant \frac{1}{2}$.
- Montrer que m divise B et en déduire qu'il existe un entier naturel k tel que pour tout $x \in (\mathbf{Z}/n\mathbf{Z})^\times$, on a $x^{\frac{m}{2}} \equiv x^{B/2^{k+1}} \mod n$.
  ▶ **Correction.–** On a clairement que $\varphi(n) = (p-1)(q-1)$ est divisible par m donc comme B est un multiple de $\varphi(n)$, le résultat en découle. On a par conséquent que $B = \ell m$. On peut poser $\ell = 2^k \ell'$ avec ℓ' impair de sorte que $\frac{B}{2^{k+1}} = \ell' \frac{m}{2}$. On a alors que modulo p, $x^{m/2} \equiv \varepsilon \pmod p$ avec $\varepsilon \in \{\pm 1\}$ et comme ℓ' est impair, $x^{\ell' \frac{m}{2}} \equiv \varepsilon \pmod p$ et de même modulo q. Ainsi, $x^{m/2}$ et $x^{B/2^{k+1}}$ sont tous les deux l'unique solution modulo n du même système de congruences modulo p et q fournie par le théorème chinois. On en déduit donc bien que $x^{\frac{m}{2}} \equiv x^{B/2^{k+1}} \mod n$.
- En déduire et implémenter un algorithme polynomial qui factorise n étant donnés n, e et d et donner sa probabilité de succès.
  ▶ **Correction.–** On tire x au hasard dans $(\mathbf{Z}/n\mathbf{Z})^\times$. On calcule alors successivement et tant que l'on peut faire la division de B par 2^{k+1}, le pgcd de n avec $x^{B/2^{k+1}} \mod n$. Avec probabilité $1 - \frac{1}{[G:H]}$, on va finir par obtenir un entier qui permette de factoriser n. Si on échoue, on peut recommencer avec un autre tirage aléatoire. Voir le Notebook!