L3 Calcul Formel
Université de Lorraine

Enoncés des TP

Clément Dell'Aiera

Table des matières

Chapitre 1

TP 1 : Algorithmes d'Euclide étendu et de Garner

Quelques adresses utiles :

- le site de l'agrégation de mathématiques http://agreg.org, vous y trouverez des textes pour vous entraîner, et surtout les comptes rendus du jury. Aussi, la liste des logiciels acceptés à l'agreg : **Python, Scilab, Octave, Sage, Maxima, Xcas, R**. Tous sont libres et gratuits.
- Nous allons travailler avec Sage, que vous pouvez télécharger sur la page http://www.sagemath.org/fr/. Vous pouvez aussi travailler directement dans une page ouverte dans le navigateur.

Les références données en bibliographie sont des livres que je recommande fortement. Ils m'ont été d'une aide précieuse, et je m'en suis inspiré pour les énoncés, notamment [1],[2] et [4]. Les livres de T. Gannon [3] et J-P. Serre [5] sont toutefois utile pour la culture.

1.1 Prise en main de SAGE

1. Ouvrir la page https://cloud.sagemath.com/ dans votre navigateur, créer un compte gratuit et un nouveau fichier (*New sageworksheet*) que vous nommerez TP1 par exemple.
2. Trouver et télécharger le tutoriel SAGE : vous pouvez le faire chez vous pour vous entraîner.
3. Pour évaluer les instructions contenues dans une cellule, vous pouvez soit cliquer sur le lien *evaluate* ou *Run* en haut à gauche, soit utiliser le raccourci clavier MAJ+ENTREE. Saisir les expressions suivantes et les évaluer :
$$9+3,\ 6*5,\ 63//9,\ 17//5,\ 17\%5.$$
4. Que donne l'évaluation des instructions suivantes ?
$$cos?,\ RationalField?\ , PolynomialRing?\ .$$
5. Pour affecter une valeur à une variable, on utilise le signe $=$. Affecter la valeur $cos(\frac{\pi}{12})$ à une variable x, puis évaluer x, $print(x)$. Pour un affichage graphique, utilisez $x.show()$. Que renvoie l'évaluation de $x.n(digits = 10)$?
6. SAGE permet de faire du calcul symbolique. Pour déclarer des variables x et y, on utilise la commande $var('x, y')$. Evaluer
$$\begin{array}{l} var\ ('x, y') \\ z = cos(x)^2 + sin(y)^2 \\ print\ z.\ subs\_expr\ (x == pi/2) \\ print\ z(y = pi/2) \end{array}$$
Si SAGE indique une erreur (*deprecation*), c'est seulement que vous avez déjà utilisé un des noms de variables auparavant. Pour corriger ce problème, il suffit de réinitialiser les variables en évaluant $reset()$ au début de votre code.
7. Définir une fonction de la façon suivante :
$$\begin{array}{l} reset() \\ var\ ('x, y') \\ f(x, y)\ =\ x/sin(x) + sqrt(y) \\ f \end{array}$$
La fonction f est désormais un objet, évaluer $f.parent()$ pour savoir lequel. Evaluer
$$f.show(),\ f.limit(x = 0).show(),\ f.diff(x).show().$$

8. On peut aussi utiliser la commande def pour définir une fonction :

$$def\ FONCTION(entree) :$$
$$instructions$$
$$return(sortie)$$

Définir une fonction qui prend en entrée un nombre réel x et renvoie son carré x^2.

1.2 Arithmétique

9. Créer une fonction qui prend en entrée deux entiers positifs, et renvoie leur *pgcd*, grâce à l'algorithme d'Euclide.
10. Créer une fonction qui prend en entrée deux entiers positifs x et y, et renvoie leur *pgcd* d ainsi que deux entiers u et v qui vérifient $ux + vy = d$. Vous utiliserez par exemple l'algorithme d'Euclide étendu.
11. Créer un fonction qui prend en entrée deux listes d'entiers $[a_1, ..., a_k]$ et $[n_1, ..., n_k]$ et renvoie une solution du système de congruences

$$\begin{cases} x &= a_1 \bmod n_1 \\ & ... \\ x &= a_k \bmod n_k \end{cases}$$

12. SAGE permet de faire de l'arithmétique avec des polynômes. Construire l'anneau des polynômes à coefficients rationels à une indeterminée en évaluant $R = QQ['x']$.
13. Spécifier le nom de la variable grâce à $T = R.gen()$.
14. Créer une fonction qui effectue l'algorithme d'Euclide étendu sur deux polynômes, et la tester sur $P = T^2 + 1$ et $Q = T^3 + 2T^2 + 1$.

Chapitre 2

TP 2 : Complexité et Cryptographie

Le matériel dont est inspiré ce TP peut être trouvé dans le cours d'arithmétique de Demazure [2].

2.1 Complexité

Au premier TP, vous avez implémenté les algorithmes d'Euclide et d'Euclide étendu. Nous allons nous intéresser au coût de ces algorithmes.

1. Soit $\varphi(n)$ le nombre d'entiers premiers à n qui sont strictement inférieurs à n, i.e. l'indicatrice d'Euler. Calculer $\varphi(n)$ en fonction de n et de ses facteurs premiers.

   La suite de Fibonacci est définie par $\begin{cases} F_0 = 0, F_1 = 1 \\ F_{n+1} = F_n + F_{n-1} \end{cases}$
2. Calculer les 7 premiers termes de la suite.
3. Montrer que, si $n > 0$, alors
   $$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$
   En déduire $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$.
4. Montrer que $F_{n+m} = F_{n+1}F_m + F_nF_{m-1}$. En déduire : $pgcd(F_{n+m}, F_m) = pgcd(F_m, F_n)$, puis $pgcd(F_n, F_m) = F_{pgcd(n,m)}$.
5. Soit $\varphi = \frac{\sqrt{5}+1}{2}$. Montrer que $F_n = \frac{1}{\sqrt{5}}(\varphi^n - (-\varphi)^{-n})$, et en déduire que F_n est l'entier le plus proche de $\frac{\varphi^n}{\sqrt{5}}$.
6. Montrer la proposition suivante, que l'on doit à Lamé (1845).
   Soient x et y deux entiers tels que $0 < y < x$ et soit d leur $pgcd$. Si l'algorithme d'Euclide partant de (x, y) s'arrête au bout de n pas, on a
   $$x \geq dF_{n+2}, \quad y \geq dF_{n+1}.$$
   Est-ce optimal ?
7. Montrer que $n \leq \frac{3}{2}log(F_{n+1}) + 1$. En déduire une majoration du nombre de pas de l'algorithme d'Euclide.

   Pour estimer le coût de l'algorithme d'Euclide, nous allons d'abord définir le modèle dans lequel nous nous plaçons, i.e. établir le coût de chaque opération arithmétique.

   • Le coût d'une addition ou soustraction $m \pm n$ est majorée par $c_+ \max(\log|m|, \log|n|)$.
   • Le coût d'une multiplication $m \times n$ est majorée par $c_\times \log|m| \log|n|$.
   • Le coût d'une division euclidienne de m par n, où $0 < n \leq m$, est majorée par $c_\% log|m| \log|m/n|$.

   Ce modèle s'appelle le modèle à coûts bilinéaires. Montrer que dans ce modèle, le coût du pgcd de x et y avec $0 \leq x < y$ est majoré par $c_\%(\log y)^2$.

2.2 Cryptographie

On s'intéresse dans cette section à deux systèmes de cryptographie "à clefs publiques". Ces deux méthodes permettent à deux individus, que l'on appellera comme le veut l'usage Alice (A) et Bob (B), d'échanger

des messages sans qu'une tierce personne puisse en bénéficier. L'originalité des méthodes à "clefs publiques" tient dans le fait que, plutôt que d'utiliser un code secret, A et B utilisent chacun un "code" connu de tous. Nous étudierons le système de Rabin (1979), ainsi que RSA (1977), du nom de ses auteurs Rivest, Shamir et Adleman. L'originalité du protocole de Rabin est qu'il est "prouvé" : casser le code est démontré être équivalent à savoir factoriser la clé publique, qui est un grand entier, en facteurs premiers. Ce n'est pas le cas de RSA, où l'on sait seulement que si l'on sait factoriser de grands entiers en temps polynômial, alors on peut casser RSA.

Le principe de ces méthodes repose sur le choix d'une fonction bijective publique pour chaque personne, dont l'inverse est difficile à calculer. Si l'ensemble des messages est $\mathfrak{M}$, A choisit une bijection $\phi_A : \mathfrak{M} \to \mathfrak{M}$ et B fait de même. Ces bijections sont publiques : tout le monde peut les trouver dans un "annuaire", et faciles à calculer. Par contre, l'inverse ϕ_A^{-1} est difficile à calculer (par exemple nécessite une temps de calcul exponentiel), sauf pour A. Si A veut envoyer un message $m \in \mathfrak{M}$ à B, A cherche ϕ_B dans l'annuaire, et envoie $s = \phi_B \circ \phi_A^{-1}(m)$ à B. Pour le décoder, B cherche ϕ_A dans l'annuaire et calcule $\phi_A \circ \phi_B^{-1}(s) = m$. On voit ici le double avantage de cette méthode : une personne tierce doit en théorie calculer ϕ_B^{-1} pour décoder "illégalement" le message, ce qui supposé impraticable, et B est sûr que A lui a envoyé le message. En effet, celui-ci est codé avec ϕ_A^{-1} qui est le secret de A ! Les protocoles de Rabin et RSA diffèrent quant à leur choix dans les fonctions de codages.

Nous nous contenterons de vouloir transmettre des messages binaires, c'est-à-dire des suites de 0 et de 1. On les convertira en base 10, et l'ensemble des messages sera $\mathfrak{M} = \mathbb{Z}/N\mathbb{Z}$, où N est un entier assez grand pour que tous vos messages soient plus petits.

1. Si tous les messages que l'on veut envoyer contiennent moins de p bits, quels N pouvez-vous choisir ?
2. Créer une fonction qui, à une suite de 0 et de 1, retourne sa valeur en base 10, i.e. $\overline{a_k...a_0} \mapsto \sum a_j 2^j$.

2.2.1 RSA

Voici le déroulement du protocole RSA.

•Génération des clés
A choisit 2 grands entiers premiers p et q, et calcule $N_A = pq$. A choisit ensuite un entier de taille moyenne d_A premier avec $\varphi(N) = (p-1)(q-1)$. La clé publique de A est (N_A, d_A), sa clé privée (p, q).

•Fonction de codage
La fonction de codage correspond à $\phi_A(m) = m^d \ mod \ N$. L'inverse de ϕ_A est donnée par $\phi_A^{-1}(s) = s^u$ où u est un inverse de d modulo $\varphi(N)$.

1. Montrer que ces fonctions sont bien inverses l'une de l'autre.
2. Y-a-t-il une contrainte sur le message m ? Est-ce grave ? Indication : Calculer la proportion d'entiers non premiers à N si p et q sont très grands, par exemple $\geq 10^{50}$.
3. Montrer que si l'on ne connait seulement que la clé publique (N_A, d_A), alors calculer l'inverse de ϕ_A équivaut à la connaissance de $\varphi(N)$. Conclure.
4. Implémenter une fonction *code* qui, étant donnée une clé publique (N_A, d_A) et un message m, retourne $\phi_A(m)$.
5. Implémenter une fonction *decode* qui, étant donnée une clé privée (p, q, d_A) et un message codé s, retourne $\phi_A^{-1}(m)$.
6. Implémenter une fonction *RSA* qui, étant donnée une clé privée (p_B, q_B, d_B), une clé publique (N_A, d_A) et un message m, retourne le message secret qu'envoie B à A.

2.2.2 Cryptosystème de Rabin

Alice et Bob veulent échanger des messages cryptés. Ils utilisent pour cela le crytposystème à clé publique de Rabin. Voici les 3 étapes de ce protocole.

•Génération des clés
A choisit 2 grands entiers premiers p et q, et calcule $N_A = pq$. La clé publique de A est N_A, sa clé privée (p, q).

•Chiffrement
B souhaite envoyer un message crypté à A. Il récupère sa clé publique N_A, représente le message comme

un entier m entre 0 et $N_A - 1$, et envoie m^2 modulo N_A à A.

•Déchiffrement
A reçoit c, et souhaite calculer une racine carrée. A calcule donc les racines carrées $\pm r_p$ de c modulo p, et $\pm r_q$ de c modulo q. Le théorème chinois donne alors les racines de c modulo N_A : m_1, $N_A - m_1$, m_2, $N_A - m_2$. A priori, A ne peut décider lequel de ces 4 messages est celui que B veut transmettre. Toutefois, ce problème peut être corrigé par redondance : B peut copier les 6 derniers bits de son message à la fin d'icelui, et A choisit le message qui présente une redondance sur ses 6 derniers bits.

Pour calculer une racine carrée de c modulo p ou q, nous utiliserons l'algorithme de Tonelli-Shanks.

Tonelli-Shanks
Entrées : p nombre premier impair, a carré modulo p.
Sortie : une racine carrée de a modulo p.

1. Trouver b entre 2 et $p-1$ qui ne soit pas carré modulo p.
2. Calculer $s \in \mathbb{N}$ et t impair tels que $p - 1 = 2^s t$.
3. $z \leftarrow b^t$
4. $m \leftarrow 0$
5. Pour j de 0 à $s-1$ faire :
   si $(a^t z^m)^{2^{s-1-j}} = -1 \ (mod\ p)$ alors $m \leftarrow m + 2^j$ fin si.
   fin pour
6. Retourner $a^{\frac{t+1}{2}} z^{\frac{m}{2}}$.

1. Implémenter l'algorithme de Tonelli-Shanks.
2. Implémenter des fonctions qui génère des clés, chiffre et déchiffre un message dans le protocole de Rabin.

2.3 Conclusion provisoire

Au terme de ce TP, l'élève attentif se posera les questions suivantes : comment fabriquer de très grands nombres premiers ? Comment choisir d ? Quelles sont les méthodes pour factoriser un entier ? Comment choisir p et q de façon à ce que RSA résiste aux méthodes de factorisation ?

Pour la première question, nous verrons qu'il existe des tests de primalité qui décident si un entier est premier ou non. Les autres questions sont plus difficiles. **Coin de la culture :** Depuis les années 80, les physiciens savent manipuler et observer des objets quantiques tels que les photons, les atomes, les ions, etc. Ce progrès technique a permis l'essor de l'informatique quantique, et la création d'algorithmes basés sur la manipulation de bits quantiques. Peter Shor a proposé un algorithme (l'algorithme de Shor..) qui permet de factoriser les grands entiers en temps polynomial, et permet donc de casser RSA. En pratique, la construction d'ordinateurs quantiques est très ardue (dûe à des problèmes liés à ce que les physiciens appellent la décohérence) et ces ordinateurs ne peuvent manipuler qu'un nombre très restreints de bits. A titre d'exemple, une équipe d'IBM a réussi à factoriser en 2001 le nombre 15 en 3×5 grâce à un ordinateur quantique manipulant 7 qubits. Pour plus de détails, vous pouvez consulter le livre (disponible à la Bibliothèque Universitaire) de Michel Le Bellac, *Introduction à l'information quantique.*

Chapitre 3

TP 3 : Tests de primalité

Le matériel dont est inspiré ce TP peut être trouvé dans le cours d'arithmétique de Demazure [2].

3.1 Calcul rapide de puissance

1. Implémenter une fonction qui à deux entiers a et m retourne a^m.
2. Voici un algorithme dit rapide pour élever un entier à un certaine puissance. Si on écrit m en base binaire, soit $m = \overline{m_k...m_1m_0}^{(2)} = \sum_{j=0}^{k} m_j 2^j$, on peut se servir récursivement de l'identité

$$a^m = a^{m_0}\left(a^{m_1}(a^{m_2}...)^2\right)^2$$

pour diminuer les coûts de calculs. Implémenter une telle fonction qui utilise moins de $2E[\log m]$ multiplications, avec E la partie entière.

3.2 Tests de primalité

3.2.1 Premier algorithme naif

Le premier test de primalité qui vient à l'esprit est de parcourir à l'aide d'une boucle tous les entiers de 2 à $n-1$ et de vérifier si l'un deux divise n. Un instant de réflexion permet de comprendre que l'on peut se limiter aux nombres inférieurs à $\sqrt{n}$. Pourquoi ?
Implémenter une fonction qui effectue ce test.

3.2.2 Test de Fermat

Nous allons utiliser le petit théorème de Fermat, que voici.

Théorème 1. Soient p un nombre premier et a un entier. Alors $a^{p-1} \equiv 1 \pmod{p}$ pour a premier à p et $a^p \equiv a \pmod{p}$ pour tout entier a.

Si l'on parvient à trouver un entier $a \in \{1, 2, ..., n-1\}$ tel que $a^{n-1} \neq 1 \mod n$ alors n n n'est pas premier. Un tel a est appelé un témoin de Fermat.

1. Implémenter une fonction $Fermat(a, n)$ qui prend deux entiers a et n en entrée, et qui retourne $True$ si a est une témoin de Fermat pour n, $False$ sinon.
2. Implémenter une fonction qui prend en entrée deux entiers n et M, qui effectue au plus M fois le test $Fermat(a, n)$ sur un nombre a tiré au hasard entre 2 et $n-1$, et qui s'arrête dès que $Fermat(a, n)$ renvoie $True$. Cette fonction doit retourner une chaîne de caractère : "n n'est pas premier" si elle a trouvé un témoin de Fermat, et "n est probablement premier" sinon.

3.2.3 Deux tests probabilistes

Importer le package *random* grâce à la commande *import random*. Que retourne l'évaluation de $random.randint(a, b)$, où a et b sont deux entiers ?

Test de Miller

Théorème 2. Soit $p > 2$ un nombre premier, et s et t, t impair, tels que $p - 1 = 2^s t$. Soit a un entier non divisible par p. Alors , ou bien $a^t \equiv 1 \pmod{p}$, ou bien il existe un entier j tel que $0 \leq j < s$ et $a^{2^j t} \equiv -1 \pmod{p}$.

De ce théorème, on déduit que si n est un entier impair, alors l'existence d'un entier a, $1 < a < n$, tel que

$$a^t \neq 1 \pmod{n} \quad \text{et} \quad a^{2^j t} \neq -1 \pmod{n}$$

pour $j = 0, .., s - 1$ assure que n est composé. Un tel a est appelé témoin de Miller pour n.(La méthode de ce numéro a été proposé par Gary Miller)

1. Implémenter une fonction $Miller(a, n)$ qui prend deux entiers a et n en entrée, et qui retourne $True$ si a est une témoin de Miller pour n, $False$ sinon.
2. Implémenter une fonction qui prend en entrée deux entiers n et M, qui effectue au plus M fois le test $Miller(a, n)$ sur un nombre a tiré au hasard entre 2 et $n-1$, et qui s'arrête dès que $Miller(a, n)$ renvoie $True$. Cette fonction doit retourner une chaîne de caractère : "n n'est pas premier" si elle a trouvé un témoin de Miller, et "n est probablement premier" sinon.

Test de Solovay-Strassen

Soit n et m deux entiers. On dit que n est un résidu quadratique modulo m s'il existe un entier a tel que $n \equiv a^2 \pmod{m}$, i.e. si n est un carré dans l'anneau $\mathbb{Z}/m\mathbb{Z}$. Si p est premier, le symbole de Legendre $\left(\frac{a}{p}\right)$ est un nombre défini comme valant 0 si p divise n, 1 si p ne divise pas n et n est un résidu quadratique modulo p, -1 sinon. Un formule dûe à Euler permet de calculer le symbole de Legendre grâce à l'algorithme des puissances rapides du premier numéro.

Proposition 1 (Euler)**.** Soit p premier impair. L'anneau $\mathbb{Z}/p\mathbb{Z}$ possède p éléments qui sont des carrés : 0 et $\frac{p-1}{2}$ éléments de $(\mathbb{Z}/p\mathbb{Z})^\times$. De plus $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$.

Si $n \geq 3$ est impair, on rappelle que $\left(\frac{a}{n}\right)$ est le symbole de Jacobi, défini comme suit. On décompose n en facteurs premiers $n = p_1^{\alpha_1} ... p_k^{\alpha_k}$, alors

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} ... \left(\frac{a}{p_k}\right)^{\alpha_k}$$

Théorème 3 (Solovay-Strassen)**.** Soit $n > 2$ un entier impair tel que $\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}$ pour tout entier a premier à n. Alors n est premier.

On peut en déduire que si n est premier, $\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}$ pour tout entier a premier à n, et si $n > 2$ est composé, l'ensemble des a premiers à n tels que $0 < a < n$ et $\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}$ a au plus $\frac{\varphi(n)}{2}$ éléments.

1. Implémenter une fonction qui calcule le symbole de Jacobi. (Difficile, même avec le théorème de réciprocité quadratique) Si vous n'y arrivez pas, vous pouvez utiliser la fonction $kronecker(a, n)$ que Sage a déjà en mémoire, et qui calcule le symbole de Jacobi.
2. Comme dans les numéros précédents, implémenter une fonction qui teste si un nombre est un témoin de Solovay-Strassen pour n, et ensuite une autre fonction qui effectue ce test aléatoirement au plus M fois.

3.2.4 Exercices

Indicatrice d'Euler

On dit qu'une fonction $f : \mathbb{N} \to \mathbb{N}$ est multiplicative si, lorsque $pgcd(n, m) = 1$, alors $f(nm) = f(n)f(m)$. Attention ce n'est pas une définition standard : généralement multiplicatif signifie respecter la mutliplication, et ce de façon inconditionnelle. Un fonction multiplicative est déterminée par ses valeurs sur les puissances de nombres premiers. On note $\varphi(n)$ l'indicatrice d'Euler, i.e. le nombre d'entiers $< n$ et premiers avec n.

1. Montrer que si f est multiplicative, alors
$$g(n) = \sum_{d|n} f(d)$$
l'est aussi.
2. Montrer que
$$n = \sum_{d|n} \varphi(d).$$

1. Démontrer le petit théorème de Fermat.
2. Coin de la culture : chercher le théorème de réciprocité quadratique si vous ne le connaissez pas.

Chapitre 4

TP 4 : Codes correcteurs

4.1 Codes linéaires

Nos deux protagonistes préférés, Alice (A) et Bob (B) ont réussi les TP 2 et 3. Ils savent donc trouver de très grands entiers premiers (TP 3) et s'en servir pour coder et décoder leurs messages (TP 2). Ils prennent toutefois conscience que leur canal de transmission est bruité ! De façon aléatoire, un voire plusieurs bits de la transmission peuvent être altérés. Pour remédier à ce problème, ils décident d'utiliser la théorie des codes correcteurs.

A et B utilisent toujours un alphabet $\mathcal{A}$ à q éléments. Nous supposerons que $\mathcal{A} = \mathbb{F}_q$ est le corps à q éléments, qui s'évalue en Sage grâce à $A = GF(q)$. On rappelle que q est alors forcément une puissance d'un nombre premier p, i.e. $q = p^f$. Un mot sera un élément x de $\mathcal{A}^n$, qui est un espace vectoriel de dimension n. On muni l'ensemble des mots de la distance de Hamming

$$d(x, y) = Card\{j \in [1, n] : x_j \neq y_j\}$$

et on définit le poids d'un élément comme

$$w(x) = Card\{j \in [1, n] : x_j \neq 0\}.$$

Définition 1.

- Un code $\mathcal{C}$ est un sous ensemble non vide de $\mathcal{A}^n$ qui possède au moins 2 éléments distincts.
- Un code linéaire est un code $\mathcal{C}$ qui est un sous-espace vectoriel de $\mathcal{A}^n$.
- Un code est binaire si $\mathcal{A} = \mathbb{F}_2$, ternaire si $\mathcal{A} = \mathbb{F}_3$.
- La distance d'un code $\mathcal{C}$ est définie comme

$$d(\mathcal{C}) = \min_{x \neq y \in \mathcal{C}} d(x, y).$$

- On définit $t(\mathcal{C})$ comme le nombre maximal d'erreurs qui sont corrigés par le code.

Voici quelques questions d'ordre théorique (i.e. à résoudre avec un crayon).

1. Montrer que $t(\mathcal{C}) = E[\frac{d(\mathcal{C})-1}{2}]$, et $d(\mathcal{C}) = 2t(\mathcal{C}) + 1$ ou $2t(\mathcal{C}) + 2$.
2. Si moins de $d(\mathcal{C}) - 1$ erreurs sont commises, on peut détecter la présence d'erreur(s). Pourquoi ?
3. Montrer que si t erreurs ont été commises lors de la transmission d'un message m, telles que $2t + 1 \leq d(\mathcal{C})$, alors il existe un unique $x \in \mathcal{C}$ tel que $d(x, m) \leq d(\mathcal{C})$.

A un code linéaire $\mathcal{C}$, on associe

- Une matrice vérificatrice $H \in \mathfrak{M}_{n-k,n}(\mathcal{A})$ de rang $n - k$ dont les lignes forment une base des formes linéaires s'annulant sur $\mathcal{C}$.
- Une matrice génératrice $G \in \mathfrak{M}_{k,n}(\mathcal{A})$ de rang k dont les lignes forment une base de $\mathcal{C}$.

On a donc $GH^T = 0$ et $HG^T = 0$. De plus $\mathcal{C}$ est le noyau de H.

Si A veut envoyer le message m, il code le message $x = mG$, et l'envoie à B. On suppose qu'au plus une erreur a été commise lors de la transmission, i.e. au plus un bit de x a été changé. Comment B peut-il tester la présence d'erreur et, le cas échéant, retrouver m ? On note $e = y - x$ l'erreur commise.

B calcule Hx.
Si $Hx = 0$, $x \in \mathcal{C}$ et $m = x$.

Sinon, $Hx \neq 0$ et, comme une seule erreur a été commise, il existe un unique j tel que He_j soit proportionnel à Hx, i.e. $\exists! j, \exists a \in \mathcal{A}, Hx = aHe_j = H(ae_j)$. Alors $m = x - ae_j$.

Code de Hamming $H(2,7)$: Le code $H(2,7)$ est le sous-espace vectoriel de $(\mathbb{F}_2)^7$ engendré par

$$e_0 = \begin{pmatrix}1\\1\\0\\1\\0\\0\\0\end{pmatrix}, e_1 = \begin{pmatrix}0\\1\\1\\0\\1\\0\\0\end{pmatrix}, e_2 = \begin{pmatrix}0\\0\\1\\1\\0\\1\\0\end{pmatrix}, e_3 = \begin{pmatrix}0\\0\\0\\1\\1\\0\\1\end{pmatrix}$$

1. Implémenter une fonction **Test** qui prend en entrée un message bruité ainsi qu'une matrice vérificatrice associée à un code $\mathcal{C}$; et retourne $True$ si le message appartient au code défini par H, $False$ sinon.
2. Implémenter une fonction **Code** qui prend en entrée un message à envoyer ainsi qu'une matrice génératrice associée à un code $\mathcal{C}$; et retourne le message codé.
3. Implémenter une fonction **Bruit** qui prend en entrée un message non bruité et le bruite. On prendra pour cela un bit du message au hasard que l'on inverse ($0 \mapsto 1$ et $1 \mapsto 0$).
4. Implémenter une fonction **Decode** qui, étant donné un code et un message reçu, vérifie s'il y a une erreur, et le cas échéant, corrige l'erreur. La fonction doit retourner le message corrigé. En cas d'erreur, en plus du message corrigé, la fonction afficher un message du type "Erreur detectée sur le bit numéro j" avec j la position de l'erreur.
5. Combien le code de Hamming $H(2,7)$ contient-t-il de mots ? Calculer sa distance, sa capacité de détection et de correction.
6. Tester vos fonctions sur le code de Hamming $H(2,7)$: choisir un message à envoyer, le coder, lui ajouter un bruit, puis le décoder.

4.2 Codes cycliques

On se donne une application appelée **shift**, définie par

$$S : \left\{ \begin{array}{lcl} A^n & \to & A^n \\ (a_0, ..., a_{n-1}) & \mapsto & (a_{n-1}, a_0, a_1, ..., a_{n-2}) \end{array} \right.$$

Définition 2. Un code cyclique de longueur n est un code linéaire $\mathcal{C} \subset A^n$ stable par S.

Comme $\mathcal{A} = \mathbb{F}_q$, on a un isomorphisme de $\mathbb{F}_q$-espace vectoriel

$$\Psi : \left\{ \begin{array}{lcl} \mathcal{A}^n & \to & \mathbb{F}_q[X]/(X^n - 1) \\ (a_0, ..., a_{n-1}) & \mapsto & a_0 + a_1 X + ... + a_{n-1} X^{n-1} \bmod (X^n - 1) \end{array} \right.$$

qui permet d'identifier A^n à l'anneau $R_n = \mathbb{F}_q[X]/(X^n - 1)$ (et donc d'avoir une structure d'anneau sur les mots). L'application Ψ envoie S sur la multiplication par X

$$\Psi \circ S(a) = X\Psi(a) \quad , \forall a \in A^n,$$

et un code cyclique de longueur n $\mathcal{C}$ n'est rien d'autre qu'un sous espace-vectoriel de R_n stable par multiplication par X, i.e. un idéal de R_n. De tels idéaux sont en bijection avec les polynômes unitaires divisant $X^n - 1$ dans $\mathbb{F}_q[X]$. On s'intéresse donc à la décomposition de $X^n - 1$ en polynômes irréductibles, ce qui se fait grâce aux polynômes cyclotomiques.

On note $\mu_n = \{w \in \mathbb{C}, w^n = 1\}$ les racines complexes $n^{\text{ième}}$ de l'unité, et $\mu_n^* = \{e^{2i\pi d/n} : d \text{ premier avec } n\}$ les racines primitives $n^{\text{ième}}$ de l'unité.

Définition 3. Le $n^{\text{ième}}$ polynôme cyclotomique est défini par

$$\Phi_n = \prod_{w \in \mu_n^*} (X - w) \in \mathbb{C}[X]$$

Ces polynômes sont en fait à coefficients entiers, et donnent la décomposition de $X^n - 1$ en facteurs irréductibles dans $\mathbb{Z}[X]$:

Proposition 2. Les polynômes cyclotomiques sont à coefficients entiers $\Phi_n \in \mathbb{Z}[X]$ et sont irréductibles dans $\mathbb{Z}[X]$, deg $\Phi_n = \varphi(n)$ (indicatrice d'Euler) et

$$X^n - 1 = \prod_{d|n} \Phi_d(X)$$

La proposition suivante donne une méthode pour calculer effectivement les polynômes cyclotomiques.

Proposition 3. Soit p premier et m non multiple de p. Si $n = p_1^{\alpha_1}...p_k^{\alpha_k}$ est non nul, on note $n' = p_1...p_k$ sa partie sans facteur carré. Alors

- $\Phi_p(X) = 1 + X + ... + X^{p-1}$
- $\Phi_{mp}(X) = \frac{\Phi_m(X^m)}{\Phi_m(X)}$
- $\Phi_n(X) = \Phi_{n'}(X^{\frac{n}{n'}})$

On en déduit l'algorithme suivant.

Polynôme cyclotomique
Entrée : $n \in \mathbb{N}^*$
Sortie : Φ_n

1. Déterminer la décomposition en facteur premier de $n = p_1^{\alpha_1}...p_k^{\alpha_k}$
2. $m = p_1 p_2 ... p_k$
3. $P_0 = X - 1$
4. Pour $j = 1, .., k$ faire $P_j \leftarrow P_{j-1}(X^{p_j})//P_{j-1}(X)$
5. Retourner $P_k(X^{n/m})$

On peut réduire les polynômes cyclotomiques modulo q et les voir comme des polynômes à coefficients modulo q, $\Phi_{n,q} \in \mathbb{F}_q[X]$, où ils ne sont pas forcément irréductibles : la réductibilité de $\Phi_{n,q}$ dépend de la nullité de n modulo q.

Proposition 4.

- Si $n = p^s m$, p ne divisant pas m, alors

$$\Phi_{n,q}(X) = \Phi_{m,q}(X)^{p^s - p^{s-1}}.$$

- Si n et q sont premiers entre eux ($s = 0$), alors $\Phi_{n,q}$ est un produit de $\frac{\varphi(n)}{r}$ polynômes unitaires irréductibles distincts, où r est l'ordre de $q \pmod n$ dans $(\mathbb{Z}/n\mathbb{Z})^*$.

Si on se place dans le cas des codes binaires de longueur impaire $n = 2^s - 1$, on obtient que $\Phi_{n,2}$ se décompose en produit de $\frac{\varphi(n)}{s}$ polynômes irréductibles sur $\mathbb{F}_2$ de degré s. On définit le polynôme suivant :

$$T_d(X) = \sum_{j=0}^{d-1} X^{2^j}.$$

On va chercher un facteur irréductible non trivial de $\Phi_{n,2}$ grâce à l'algorithme de Cantor-Zassenhauss, qui permet de trouver un facteur d'un polynôme dont tous les facteurs irréductibles ont même degré.

Cantor-Zassenhauss
Entrée : $P \in \mathbb{F}_2[X]$ unitaire sans facteur carré ayant tous des facteurs irréductibles de même degré d, $deg\ P = n > 0$.
Sortie : Un facteur non trivial de P, ou $False$.

1. Choisir $Q \in \mathbb{F}_2[X]$ au hasard de degré $< n$.
2. $g = pgcd(P, Q)$
3. Si $g = 1$, alors retourner g
4. $a = T_d(Q) \pmod P$
5. $g \leftarrow pgcd(a - 1, P)$
6. Si $g \neq 1$ et $g \neq P$, retourner g, sinon retouner $False$

1. Implémenter une fonction qui, étant donné un entier n, retourne le $n^{\text{ième}}$ polynôme cyclotomique Φ_n, grâce à l'algorithme **Polynôme Cyclotomique**.

2. Implémenter l'algorithme de Cantor-Zassenhauss.
3. Implémenter une fonction qui, étant donné un entier impair n, retourne la liste des facteurs irrédutibles du $n^{\text{ième}}$ polynôme cyclotomique $\Phi_{n,2}$.

Attention, la première fonction travaille dans $\mathbb{Z}[X]$, la troisième dans $\mathbb{F}_2[X]$. Cette dernière fonction vous donne donc la liste de tous les codes cycliques binaires de longueur impaire n.

Chapitre 5

TP 5 : Pivot de Gauss-Jordan

On travaillera pour ce TP dans l'anneau des entiers $A = \mathbb{Z}$, mais l'algorithme présenté fonctionne correctement dans tout anneau euclidien A. La base canonique de $\mathfrak{M}_{n,m}(A)$ est notée $E_{ij} = (\delta_{l=i,l'=j})_{1\leq l\leq n, 1\leq j\leq m}$. Soit $\mathcal{P}$ un système complet d'éléments irréductibles de A, pour les entiers on peut prendre $\mathcal{P} = \mathbb{P}$ l'ensemble des nombres premiers. Tout élément $n \in A$ s'écrit

$$n = u \prod_{p\in\mathcal{P}} p^{v_p(n)}$$

où u est une unité de A. On définit le poids d'un élément n comme

$$\delta(n) = \sum_{p\in\mathcal{P}} v_p(n) \in \mathbb{N}.$$

- Si $M = (m_{ij}) \in \mathfrak{M}_{n,m}(A)$, $M^{(k)}$ désigne la sous matrice de taille $(n-k+1)\times(m-k+1)$ obtenue en ne gardant que le "coin en bas à droite" :

$$M^{(k)} = (m_{ij})_{k\leq i\leq n, k\leq j\leq m}.$$

- Si $x \in A$, $i \neq j$, et $l > 0$, on appelle matrices de transvection les matrices

$$T^l_{ij}(x) = I_l + xE_{ij} \in GL(l, A)$$

Lorsqu'une matrice M est fixée, on note L_i sa $i^{\text{ème}}$ ligne et C_j sa $j^{\text{ème}}$ colonne. L'algorithme du pivot de Gauss ramène un système linéaire quelconque à une forme que l'on appelle échelonnée au moyen d'opérations élémentaires sur les lignes et les colonnes. On se servira tout au long du TP des faits suivants :

- l'opération $L_i \leftarrow L_i + xL_j$ est donnée par l'opération matricielle

$$\left\{\begin{array}{lcl} \mathfrak{M}_{n,m}(A) & \to & \mathfrak{M}_{n,m}(A) \\ M & \mapsto & T^n_{ij}(x)M \end{array}\right.$$

- l'opération $C_j \leftarrow C_j + xC_i$ est donnée par l'opération matricielle

$$\left\{\begin{array}{lcl} \mathfrak{M}_{n,m}(A) & \to & \mathfrak{M}_{n,m}(A) \\ M & \mapsto & MT^m_{ij}(x) \end{array}\right.$$

- l'opération $L_i \leftrightarrow L_j$ est donnée par l'opération matricielle

$$\left\{\begin{array}{lcl} \mathfrak{M}_{n,m}(A) & \to & \mathfrak{M}_{n,m}(A) \\ M & \mapsto & T^n_{ij}(1)T^n_{ji}(-1)T^n_{ij}(1)M \end{array}\right.$$

- l'opération $C_i \leftrightarrow C_j$ est donnée par l'opération matricielle

$$\left\{\begin{array}{lcl} \mathfrak{M}_{n,m}(A) & \to & \mathfrak{M}_{n,m}(A) \\ M & \mapsto & MT^m_{ij}(-1)T^m_{ji}(1)T^m_{ij}(-1) \end{array}\right.$$

Si $M = (m_{ij}) \in \mathfrak{M}_{n,m}(A)$, on définit $p_M(i) = \inf\{k : m_{ik} \neq 0\}$. On dit que M est sous **forme échelonnée** si la suite $p_M(i)$ est strictement croissante,

$$M = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & 0 & * \end{pmatrix} \quad \text{par exemple.}$$

Le but du pivot de Gauss est, étant donnée une matrice $M \in \mathfrak{M}_{n,m}(A)$, de trouver une suite de transvections telle que, en faisant successivement les multiplications à gauche par ces transvections, la matrice obtenue soit échelonnée. La matrice échelonnée est équivalente à la matrice de départ au sens suivant.

$$M \sim N \text{ si } \exists P \in GL(n, A)/M = PN.$$

L'algorithme est basé sur le lemme suivant :

Lemme 1. Soit $x \in A^n$ un vecteur, et d un pgcd des composantes de x, alors il existe $L \in GL(n, A)$ telle que

$$Lx = \begin{pmatrix} d \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Par récurrence, on en déduit que pour toute matrice $M \in \mathfrak{M}_{n,m}(A)$, il existe une matrice inversible $L \in GL(n, A)$ telle que LM soit échelonnée.

Pivot de Gauss
Entrée : $M \in \mathfrak{M}_{n,m}(A)$ non nulle.
Sortie : $D \in \mathfrak{M}_{n,m}(A)$ matrice échelonnée équivalente à M.
$L := I_n$
$M_0 = M$
Pour j allant de 1 à $k = \min(m, n)$:

1. trouver $P \in GL(n - j + 1, A)$ telle que

$$PC_j = \begin{pmatrix} d_j \\ \vdots \\ 0 \end{pmatrix}$$

où C_j est la $j^{\text{ème}}$ colonne de $M_j^{(j)}$, et d_j le pgcd de ses composantes.

2. $M_{j+1} = P^{(j)} M_j$ où $P^{(j)} = \begin{pmatrix} I_{j-1} & 0 \\ 0 & P \end{pmatrix}$

Retourner M_k

1. Implémenter une fonction T qui prend entrée deux indices i et j, un entier l, et un élément $x \in A$, et renvoie la matrice de transvection $T_{ij}^l(x)$.
2. Implémenter une fonction *echange* qui prend en entrée une matrice M et quatres indices i_0, i_1, j_0 et j_1, et renvoie la matrice M ayant subie les permutations $L_{i_0} \leftrightarrow L_{i_1}$ et $C_{j_0} \leftrightarrow C_{j_1}$.
3. Implémenter une fonction *Test* qui prend en entrée un vecteur x de $\mathbb{Z}^k$ et renvoie $True$ si une seule composante de x est non nulle, $False$ sinon.
4. Implémenter une fonction *DE* qui, étant donné un vecteur x de $\mathbb{Z}^k$, renvoie une matrice inversible $P \in GL(j, \mathbb{Z})$ telle que

$$Px = \begin{pmatrix} pgcd(x_1, ...x_k) \\ \vdots \\ 0 \end{pmatrix}.$$

Pour cela, on trouve la composante i_0 de valeur absolue minimale de x, $|x_{i_0}| = \min\{|x_j|\}$ et on fait la division euclidienne de toutes les autres composantes par x_{i_0}. On recommence jusqu'à ce qu'il n'y ait au plus qu'une composante non nulle. Une remarque importante, la division euclidienne de x_j par x_{i_0} se fait grâce à l'opération $L_j \leftarrow L_j - qL_{i_0}$ où $q = x_j//x_{i_0}$, donc grâce à la multiplication à gauche par une transvection bien choisie, i.e. $T_{ji_0}^n(-q)$. Une fois qu'il ne reste qu'une composante non nulle, utiliser *echange* pour mettre ce coefficient en première position. Vous pouver utiliser des fonctions auxiliaires.

5. Implémenter une fonction *Pivot* qui prend en entrée une matrice M de taille $n \times m$, et renvoie une matrice inversible $P \in GL(n, \mathbb{Z})$ telle que PM soit échelonnée.
6. Implémenter une fonction *poids* qui calcule le poids d'un entier.
7. Implémenter une fonction *MinimalWeight* qui prend en entrée une matrice M et un entier k, et renvoie la position de l'élément de poids minimal de la première colonne de la sous-matrice $M^{(k)}$.
8. Implémenter une fonction qui effectue le pivot de Gauss, cette fois-ci en remplaçant "valeur absolue" par "poids" dans le choix du pivot. (Dans le premier cas, le pivot était choisi de façon à minimiser la valeur absolue.)

Voici quelques applications de cet algorithme. Elles sont toutes traitées dans le chapitre VIII du livre de Berhuy, *Modules : théorie et pratique...et un peu d'arithmétique*, que je vous conseille vivement.

- **Forme normale de Smith**
  En appliquant le lemme sur la première ligne, puis la première colonne, et en affinant un peu, on peut obtenir le théorème suivant. Si $M \in \mathfrak{M}_{n,m}(\mathbb{Z})$, alors il existe des matrices inversibles $L \in GL(n,\mathbb{Z}), R \in GL(m,\mathbb{Z})$ telles que LMR soit diagonales à coefficients positifs, de coefficients diagonaux b_j tels que $b_j|b_{j+1}$. La matrice diagonale obtenue est unique, on l'appelle la forme de Smith de M.
- **Résolution de systèmes d'équations linéaires dipophantiennes.**
  Soit A un anneau principal et a_1, a_2, ..., a_k des éléments de A non tous nuls et $b \in A$. Soit d un généateur de $(a_1, ..., a_k)$. L'équation

  $$a_1x_1 + ... + a_kx_k = b$$

  admet une solution dans A^k ssi $d|b$. Alors toute solution s'écrit de manière unique

  $$\alpha C_1 + x_2C_2 + x_kC_k, x_j \in A,$$

  avec $\alpha \in A$ tel que $b = \alpha d$ et les C_j sont les colonnes d'un matrice inversible $C \in GL(k,A)$ telle que

  $$(a_1 \ ... \ a_k)C = (d \ 0 \ ... \ 0).$$

  Par exemple, vous pouvez résoudre $3x + 4y + 7z = b$ dans $\mathbb{Z}$, un exemple corrigé dans le livre de Berhuy, chp VII, I.22, p 248.
- **Théorème de structure des groupes abéliens de type fini.**
  Soit G un groupe abélien admettant un nombre fini de générateurs. Alors il existe deux entiers positifs p et q, et des entiers non nuls $d_1|d_2|...|d_q$, $d_j \geq 2$, tels que

  $$G \simeq \mathbb{Z}^p \times \mathbb{Z}/d_1\mathbb{Z} \times ... \times \mathbb{Z}/d_q\mathbb{Z}.$$

  Ces entiers p et d_j sont uniques et caractérisent la classe d'isomorphisme de G.
- **Décomposition de Frobenius.**
  Soit V un k-espace vectoriel et $f \in \mathcal{L}(V)$ un endomorphisme. Alors il existe des polynômes unitaires $P_1|P_2|...|P_r$ de $k[X]$ et une base B de V tels que

  $$Mat(f,B) = \begin{pmatrix} C_{P_1} & & \\ & ... & \\ & & C_{P_r} \end{pmatrix}$$

  où C_P est la matrice compagnon associée à P, i.e. si $P = X^n + p_{n-1}X^{n-1} + ... + p_0 \in k[X]$,

  $$C_P = \begin{pmatrix} 0 & & & -p_0 \\ 1 & 0 & & -p_1 \\ & 1 & ... & ... \\ & & 0 & -p_{n-2} \\ & & 1 & -p_{n-1} \end{pmatrix}.$$

  Ces polynômes sont appelés les invariants de similitudes de f et la matrice diagonale par bloc ci-dessus est la forme de Frobenius de f ? De plus, 2 endomorphismes sont semblables ssi ils ont la même forme de Frobenius.
- **Théorème de structure des modules de type fini sur un anneau principal.**
  Tous ces théorèmes sont des corollaires du théorème de structure des modules de type fini sur un anneau principal, qui est traité dans le livre de Berhuy. Pour les TP précédents, j'ai principalement utilisé les livres *Cours d'Arithmétique* de Michel Demazure et *Arithmétique* de Marc Hindry.

Bibliographie

[1] Grégory Berhuy. *Modules : théorie, pratique,... Et un peu d'arithmétique !* 2012.

[2] Michel Demazure. *Cours d'algèbre : primalité, divisibilité, codes*, volume 1. Cassini Paris, 1997.

[3] Terry Gannon. *Moonshine beyond the Monster : The bridge connecting algebra, modular forms and physics*. Cambridge University Press, 2006.

[4] Marc Hindry. *arithmétique*. 2008.

[5] Jean-Pierre Serre. Cours d'arithmétique. 1970.