

Table des matières

1 Manipulation de grands entiers	1
2 Bézout et Euclide	2
3 Nombres premiers	2
3.1 MuPAD et les nombres premiers	2
3.2 Nombres de Fermat	3
3.3 Nombres de Mersenne	3
3.4 Sur la répartition des nombres premiers	3
4 Algorithmes d'exponentiation	4
4.1 Naïf itératif et naïf récursif	4
4.2 Dichotomique itératif et dichotomique récursif	4
4.3 Algorithme d'exponentiation modulaire	4
4.4 Calcul des nombres de Fibonacci par exponentiation dans les matrices 2*2	5
5 Cryptosystème RSA	5
5.1 Introduction	5
5.2 L'ensemble des messages, codage, décodage	5
5.3 La génération de clés RSA	6
5.4 Fonctions de chiffrement et déchiffrement RSA	6
5.5 Signature avec RSA	6
Références bibliographiques	7

1 Manipulation de grands entiers

1. Les plus grands entiers manipulables par MuPAD

Que signifie le résultat retourné par la commande `time` (consulter l'aide en ligne)?

Quel est le nombre de chiffre d'un entier de la forme $10^{(10^k)}$, où $k = 1, 2, \dots$?

Faire successivement :

```
time(10^10)
time(10^(10^2))
time(10^(10^3))
...
```

En déduire une évaluation du nombre de chiffres des plus grands entiers manipulables (explicitement) par MuPAD.

2. Ecrire une fonction MuPAD qui retourne le nombre de chiffres (en base 10) d'un entier x passé en argument.

3. Donner le nombre de chiffres des nombres suivants :

$$12^{34}, (12^{34})^3, 12^{(34^3)} \quad 100!, \quad 1000!, \quad 10000!, \quad 100000!, \quad 6!!, \quad 7!!, \quad 8!!$$

Attention : NE PAS afficher ces nombres à l'écran. 8!! s'étale sur plusieurs dizaines d'écrans, et la fonction d'affichage à l'écran (appelée par défaut si on tape `>> 8!!` en omettant les `:` en fin de ligne) est TRES lente.

Remarque importante : les parenthèses dans l'exponentiation répétée sont indispensables. Comparer :

$$(12^{34})^3, (12^3)^{34}, 12^{(34^3)}, 12^{(3^{34})}$$

4. MuPAD ne permet pas de calculer 100!!, on ne peut donc pas utiliser la fonction précédente pour calculer son nombre de chiffres. Utiliser la formule de Stirling :

$$\ln(n!) = n \ln n - n + \frac{1}{2} \ln n + \ln \sqrt{2\pi} + \frac{1}{12n} + O\left(\frac{1}{n^3}\right)$$

pour écrire une fonction MuPAD qui retourne le nombre de chiffres de $n!$ d'un entier n passé en argument.

5. Calculer le nombre de chiffres du nombre de chiffres de 100!!. Evaluer le nombre d'écrans nécessaires pour afficher 100!! et le nombre de Go nécessaires pour stocker 100!!.

2 Bézout et Euclide

Ecrire une fonction MuPAD qui prend en entrée un couple $(a, b) \in \mathbb{N} \times \mathbb{N}$ et qui retourne un triplet (α, β, d) tel que

$$\alpha a + \beta b = d = \text{pgcd}(a, b).$$

Comparer avec `igcdex`. Modifier votre programme pour qu'il retourne aussi le nombre de divisions effectuées. Tester avec des nombres de Fibonacci, vérifier que le nombre de divisions est alors maximal (revoir TD1). Tester aussi la taille des coefficients de Bézout (TD1 aussi!).

3 Nombres premiers

3.1 MuPAD et les nombres premiers

Les nombres premiers sont les notes de la musique des nombres. MuPAD possède quelques commandes très utiles pour travailler avec les nombres premiers :

- `isprime(n)` pour tester si l'entier n est premier ;
- `ithprime(n)` retourne le $n^{\text{ième}}$ nombre premier ;
- `nextprime(n)` retourne le nombre premier immédiatement supérieur à n ;
- `ifactor(n)` retourne la décomposition en facteurs premiers de l'entier n .

1. Expliquer la commande suivante :

```
a:=select([$1..100],isprime):
```

où le dollar représente la "sequence generation" (en anglais, consulter l'aide de `_seqgen`).

2. Quel est le nombre de nombres premiers inférieurs ou égaux à 100 ? 1000 ? 10000 ? 100000 ? (utiliser `nops`).

3.2 Nombres de Fermat

Les nombres de Fermat sont de la forme

$$F_n = 2^{(2^n)} + 1$$

Fermat¹ avait conjecturé que tous les F_n sont premiers. Montrer qu'il avait tort.

3.3 Nombres de Mersenne

Les nombres de Mersenne sont de la forme

$$M_p = 2^p - 1 \quad p \text{ nombre premier}$$

Marin Mersenne² affirma (en 1644, apprécier la performance!) que M_p est premier pour $p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127$ et 257 , et composé pour les autres valeurs de p inférieures à 257 .

Euler écrit en 1732 que M_{41} et M_{47} sont premiers, mais il se trompe . . .

L'objectif est de tester l'affirmation de Mersenne.

1. Former la liste des nombres premiers inférieurs ou égaux à 257 . Combien y en a-t-il? Former la liste des nombres de Mersenne correspondants.
2. Parmi la liste précédente, donner les nombres de Mersenne qui sont des nombres premiers. Rectifier l'affirmation de Mersenne.
3. Décomposer M_{41} et M_{47} en facteurs premiers.

3.4 Sur la répartition des nombres premiers

On sait, semble-t-il depuis Euclide (environ -300), que l'ensemble des nombres premiers est infini. On ne connaît toujours pas de formule permettant de générer les nombres premiers, ni même seulement des nombres premiers arbitrairement grands. Leur répartition parmi les entiers reste mystérieuse³ mais on dispose quand même du **théorème des nombres premiers (Hadamard et De La Vallée Poussin, 1896)** :

$$\lim_{n \rightarrow +\infty} \pi(n) \frac{\ln n}{n} = 1$$

où $\pi(n)$ désigne le nombre de nombres premiers $\leq n$.

Autrement dit, la densité des nombres premiers $\frac{\pi(n)}{n}$ tend à se comporter comme $\frac{1}{\ln n}$: c'est le

1. **Pierre de FERMAT** 1601-1665. On lui doit de nombreux résultats en arithmétique et en probabilité, ainsi que quelques fameuses affirmations qui ont fait couler beaucoup d'encre. Si Euler montra que $F_5 = 4294967297 = 641 * 6700417$ réfutant ainsi la conjecture de Fermat annoncée dans le texte, il fallut attendre 1994 pour que Andrew Wiles établisse la preuve du "grand théorème de Fermat", à savoir : Pour tout entier $n \geq 3$, l'équation $x^n + y^n = z^n$ n'admet aucune solution (x, y, z) dans $(\mathbb{N}^*)^3$.

2. **Marin MERSENNE** 1588-1648. Moine de l'ordre des Minimes, grand esprit de son temps, a traduit de nombreux ouvrages de mathématiciens grecs.

3. On sait que pour tout entier $n \geq 1$, il existe un nombre premier p tel que $n < p \leq 2n$ (propriété appelée postulat de Bertrand, démontrée pour la première fois par Chebyshev en 1850). On sait aussi qu'il existe des trous arbitrairement grands entre deux nombres premiers consécutifs : pour tout entier $n > 1$, aucun de ces $n - 1$ entiers consécutifs n'est premier : $n! + 2, n! + 3, \dots, n! + n$. La recherche actuelle sur les nombres premiers est très active et laisse entrevoir des propriétés merveilleuses, liées à l'hypothèse de Riemann (les zéros non triviaux de la fonction zéta de Riemann

$$\zeta(s) = \sum_{n \geq 1} \frac{1}{n^s} = \prod_{p \text{ premiers}} \frac{1}{1 - p^{-s}}$$

sont tous sur la droite $Re(s) = \frac{1}{2}$).

phénomène de raréfaction des nombres premiers.

L'objectif est d'observer le comportement de suite $u_n = \pi(n) \frac{\ln n}{n}$, pour des n les plus grands possibles.

1. A l'aide de la commande `time`, évaluer les limites de la formule `nops(select([1..n], isprime))` qui fournit $\pi(n)$. Pour cela, prendre successivement $n = 10^2, 10^3, \dots$ et s'arrêter dès que `time` retourne des temps de l'ordre de la minute.
2. Utiliser la formule précédente pour tracer sur un graphe u_n en fonction de n . On utilisera pour cela `plot::Point`, `plot::Pointlist`, `plot`, avec environ 100 valeurs de n réparties de façon logarithmique entre 1 et la limite obtenue précédemment.
3. Utiliser `nextprime` (pour éviter des appels inutiles à `isprime`) pour calculer u_n pour des valeurs de n au-delà de la limite du 1. Jusqu'à quelle valeur de n (ordre de grandeur) est-il raisonnable d'aller ?

4 Algorithmes d'exponentiation

Il s'agit de programmer quelques algorithmes d'exponentiation et de comparer leur complexité empirique. Dans chaque cas, on comparera aussi avec la commande MuPAD usuelle `x^n`.

4.1 Naïf itératif et naïf récursif

Ecrire une fonction MuPAD qui calcule x^n par multiplications successives par x . Cette fonction prendra comme arguments d'entrée x et n , et retournera x^n .

Ecrire un programme itératif, un autre récursif. Comparer leur complexité.

4.2 Dichotomique itératif et dichotomique récursif

On remarque d'abord que

$$x^n = \begin{cases} (x^2)^{\frac{n}{2}} & \text{si } n \text{ pair} \\ x(x^2)^{\frac{n-1}{2}} & \text{si } n \text{ impair} \end{cases}$$

ce qui suggère la règle de réécriture

$$(x, n, y) \longrightarrow \begin{cases} (x^2, \frac{n}{2}, y) & \text{si } n \text{ pair} \\ (x^2, \frac{n-1}{2}, y \cdot x) & \text{si } n \text{ impair} \end{cases}$$

qu'on initialise à $(x, n, 1)$ et qu'on applique tant que $n > 0$.

Ecrire un programme itératif, un autre récursif. Comparer leur complexité.

4.3 Algorithme d'exponentiation modulaire

Adapter l'algorithme précédent pour calculer $x^n \pmod{N}$. On rappelle que calculer modulo N revient à calculer avec des restes de divisions par N et que cela pourra se faire par la commande `mod`.

Ecrire une fonction MuPAD qui calcule $x^n \pmod{N}$. Elle devra prendre comme argument d'entrée l'entier N (en sus de x et n évidemment).

Comparer avec `powermod`.

4.4 Calcul des nombres de Fibonacci par exponentiation dans les matrices 2*2

On a vu en TD que

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Ecrire une fonction MuPAD qui calcule F_n par cette méthode. Pour ce faire, utiliser les facilités de MuPAD concernant les matrices et en particulier l'exponentiation.

Evaluer sa complexité empirique. MuPAD utilise-t-il un algorithme dichotomique pour calculer A^n quand A est une matrice? Comparer aussi avec `numlib::fibonacci`.

Par curiosité, taper aussi les commandes suivantes :

```
expose(_power)
expose(powermod)
```

5 Cryptosystème RSA

5.1 Introduction

Rivest, Shamir et Adelman ont proposé dans [2] une méthode de cryptographie à clé publique dont on rappelle les grandes lignes :

- (i) choix de deux (grands) nombres premiers p et q , qu'on gardera secrets ;
- (ii) pour clé publique, former $N = pq$ et choisir e premier avec $\phi(N) = (p-1)(q-1)$. Le couple (N, e) sera rendu public et permettra le cryptage des messages ;
- (iii) pour clé privée, on calcule un entier d tel que $ed = 1 \pmod{\phi(N)}$ par l'algorithme d'Euclide étendu.

On suppose que les messages sont codés en des listes d'éléments de $\mathbb{Z}/N\mathbb{Z}$. La fonction de cryptage (ou de chiffrement) est alors

$$\begin{aligned} RSA_{(N,e)} : \mathbb{Z}/N\mathbb{Z} &\longrightarrow \mathbb{Z}/N\mathbb{Z} \\ x &\longmapsto x^e \end{aligned}$$

tandis que le déchiffrement se fera par $RSA_{(N,d)}$.

Ainsi chacun peut chiffrer (puisque (N, e) est publique) mais seul celui qui connaît d peut déchiffrer. En effet, dans l'état actuel des connaissances, le calcul de d (à partir de N et e) nécessite celui de $\phi(N)$, donc de factoriser N , ce qu'on ne sait pas faire en temps raisonnable. D'où l'efficacité de la méthode RSA (jusqu'à présent?).

L'objectif du TP est de construire un système de cryptographie RSA en utilisant les facilités offertes par MuPAD.

5.2 L'ensemble des messages, codage, décodage

Comme les fonctions de chiffrement-déchiffrement RSA travaillent sur $\mathbb{Z}/N\mathbb{Z}$, l'ensemble des messages admissibles est constitué des suites d'entiers compris entre 0 et $N-1$ (pour avoir le représentant canonique des classes d'entiers modulo N). Bien sûr, ces suites d'entiers devront coder pour des messages initiaux écrits en français.

On va utiliser le code ASCII de MuPAD : consulter l'aide de `numlib::fromAscii` et `numlib::toAscii`. Le message à coder sera découpé en paquets de Nc caractères, puis chaque caractère sera remplacé par son code ASCII. Enfin ces codes Nc seront "concaténés" pour former un entier qui devra être $< N$. Ainsi une chaîne de caractères sera transformée en une liste d'entiers tous compris entre 0 et $N - 1$, et cette liste constitue le message à crypter.

Ecrire une fonction `numerise` qui réalise cette transformation. Elle admettra comme paramètres d'entrée :

- une chaîne de caractères $m1$;
- un entier Nc .

En sortie, elle produira une liste d'entiers $m1c$.

Ecrire une fonction `alphabetise` qui réalise l'opération inverse. Attention au traitement du dernier entier qui peut être issu d'un morceau de message de moins de Nc caractères.

5.3 La génération de clés RSA

C'est un point crucial du dispositif RSA : il faut choisir des couples de nombres premiers (p, q) tels qu'il soit impossible de calculer p et q à partir de la connaissance de leur produit $n = pq$.

Evidemment, il faut choisir p et q assez grands pour que leur produit n ne puisse pas être factorisé par la fonction `ifactor` de MuPAD. Evaluer les limites de `ifactor`.

[Indication : attention, ce n'est pas si facile. En particulier, il faut s'affranchir des nombres à petits facteurs qui se factorisent rapidement, même s'ils ont plus de 100 chiffres décimaux.]

A l'aide d'une recherche Wiki, Google, ...

Quelles sont les recommandations de la société RSA Data-Security ? Pourquoi les nombres de Mersenne premiers doivent être évités ? Pourquoi l'exposant de chiffrement e ne doit pas être choisi trop petit ?

Ecrire une fonction `cleRSA` qui délivre un ensemble de paramètres RSA (N, e, d) qui tiennent compte des recommandations précédentes. Elle prendra en entrée un entier m qui sera un minimum du nombre de chiffres du N délivré. Produire quelques clés avec N de 30 chiffres, et indiquer le temps mis par `ifactor` pour casser ces clés. Votre fonction `cleRSA` peut-elle générer des clés RSA de 2048 bits (réputées incassables) ?

5.4 Fonctions de chiffrement et déchiffrement RSA

Elles sont de même nature (une exponentiation modulaire) et s'écrivent en une seule ligne dans MuPAD. Elles transforment une liste d'entiers compris entre 0 et $N - 1$ en une liste de même nature.

[Indication : utiliser `map` et `powermod`.]

5.5 Signature avec RSA

On va utiliser deux protocoles de signature. Pour les présenter, considérons deux individus Alice et Bob, de paramètres respectifs (N_A, e_A, d_A) et (N_B, e_B, d_B) , qui désirent s'échanger des messages. On suppose Nc fixé et que $3Nc$ (= le nombre de chiffres nécessaire pour coder un bout de message) vérifie $3Nc < N_A$ et $3Nc < N_B$. Alice envoie un message à Bob. Ce message

ne doit pas être lu par un tiers, Bob doit savoir le lire et il doit être sûr qu’Alice est bien à l’origine du message.

1. **Protocole 1** : message+signature

- (i) Alice numérise son texte (m_1) et sa signature (s_1). Elle obtient (m_{1c}, s_{1c}) .
- (ii) Alice chiffre m_{1c} avec la clé publique de Bob (N_B, e_B) et s_{1c} avec sa clé privée (N_A, d_A). Elle obtient (m_{2c}, s_{2c}) qu’elle transmet à Bob.
- (iii) Bob déchiffre m_{2c} avec sa clé privée (N_B, d_B) (il est le seul à pouvoir le faire) et déchiffre s_{2c} avec la clé publique d’Alice (N_A, e_A) (il est sûr que le message vient d’Alice car elle seule a pu ainsi chiffrer sa signature). Il retrouve ainsi (m_{1c}, s_{1c}) .
- (iv) Bob alphabétise m_{1c} et s_{1c} pour retrouver le texte d’Alice m_1 et sa signature s_1 .

2. **Protocole 2** : message signé

Alice applique un double cryptage à son message, l’un avec la clé publique de Bob (seul Bob pourra déchiffrer le message), l’autre avec sa clé privée (seule Alice peut le faire). Mais attention : pour que Bob puisse retrouver le message original, ces deux opérations doivent se faire dans le bon ordre.

Si $N_A > N_B$:

- (i) Alice numérise son texte m_1 , elle obtient m_{1c} .
- (ii) Alice crypte m_{1c} avec la clé publique de Bob (N_B, e_B), elle obtient m_{2c} , qu’elle chiffre ensuite avec sa clé privée (N_A, d_A) pour obtenir m_{3c} qu’elle transmet à Bob.
- (iii) Bob applique d’abord la clé publique d’Alice pour récupérer m_{2c} , auquel il applique sa clé privée pour récupérer m_{1c} .
- (iv) Enfin, Bob alphabétise m_{1c} et retrouve le texte d’Alice m_1 .

Si $N_A < N_B$:

- (i) Alice numérise son texte m_1 , elle obtient m_{1c} .
- (ii) Alice crypte m_{1c} d’abord avec sa clé privée (N_A, d_A) et obtient m_{2c} . Elle chiffre ensuite m_{2c} avec la clé publique de Bob (N_B, e_B) et obtient m_{3c} qu’elle transmet à Bob.
- (iii) Bob applique d’abord sa clé privée pour récupérer m_{2c} , puis la clé publique d’Alice pour récupérer m_{1c} .
- (iv) Enfin, Bob alphabétise m_{1c} et retrouve le texte d’Alice m_1 .

Ecrire des fonctions `protocole1` et `protocole2` qui réalisent la signature (ou la lecture selon les paramètres RSA fournis en entrée) en mettant en oeuvre les deux protocoles ci-dessus.

Expliquer pourquoi il faut considérer deux cas dans le protocole 2. Pourquoi n’a-t-on pas considéré le cas $N_A = N_B$?

Références

- [1] JEAN-PAUL DELAHAYE, *Merveilleux nombres premiers*, Belin, Pour la science, 2000.
- [2] R. RIVEST, A. SHAMIR, L. ADLEMAN, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Comm. ACM **21**, p. 120-126, 1978.