

TP4 : Calculs dans $\mathbb{Z}/n\mathbb{Z}$

N'oubliez pas d'exécuter (valider avec la touche Entrée) les commandes Maple (texte en rouge) avant de les utiliser.

— Les commandes Maple qui peuvent vous être utiles

Beaucoup des très nombreuses fonctions de Maple ne sont pas disponibles au démarrage : elles sont rangées dans des bibliothèques par thème (algèbre linéaire, équations différentielles, arithmétique, graphiques, etc...) et il faut les charger pour les utiliser.

La bibliothèque d'arithmétique s'appelle **numtheory** (théorie des nombres). On commence donc par

```
[ > with(numtheory);
```

Parcourez rapidement les exemples ci-dessous, vous reviendrez faire votre marché au besoin en résolvant les exercices.

```
[ > ifactor(60);
```

La fonction usuelle de décomposition d'un entier en facteurs premiers, le résultat est joli mais peu utilisable dans un programme.

```
[ > ifactors(60);
```

Beaucoup moins lisible mais beaucoup plus utile : le premier élément 1 ou -1 est le signe, le deuxième une liste de couples $[p, k]$ représentant le facteur p^k .

Pour n'avoir que cette liste

```
[ > ifactors(60)[2];
```

La fonction **phi** de Maple vous permettra de vérifier vos programmes des exercices 1 et 2.

```
[ > phi(60);
```

Les fonctions **divisors** et **factorset** donnent l'ensemble des diviseurs et l'ensemble des facteurs premiers d'un entier.

```
[ > divisors(60);
```

```
[ > factorset(60);
```

La fonction **order** permet de calculer l'ordre d'un élément modulo n et donc de vérifier vos programmes des exercices 3 et 4.

```
[ > order(49, 60); 49*49;
```

La fonction **primroot** donne le plus petit élément primitif de $\mathbb{Z}/n\mathbb{Z}$ s'il existe.

```
[ > primroot(11); primroot(60);
```

— Facultatif : pour celles et ceux qui aiment soulever le capot

La plupart des fonctions de Maple sont écrites en Maple, pour les curieux il est possible de faire afficher leur code.

La commande magique qui permet de voir le code que Maple a dans le ventre :

```
[ > interface(verboseproc=2);
```

```
[
```

```
[ > print(numtheory[phi]);
```

La procédure commence par un certain nombre de tests sur le type de l'argument.

Dans les versions plus récentes une partie de ces tests sont dans le typage du paramètre.

Certaines procédures sont de longues listes de tests sur le type des arguments suivis de l'appel aux procédures qui font le boulot et ce sont celles-là qu'il faut aller voir, il peut y avoir un peu de jeu de piste.

J'ai choisi phi parce qu'elle est simple et on peut voir l'algorithme :

- si $n < 10$ le résultat est pris dans une liste
- si n est premier on rend $n - 1$
- sinon on appelle factorset qui rend l'ensemble $\{p_1, \dots, p_k\}$ des facteurs premiers de n sans

leurs exposants et on utilise la formule
$$\frac{n \left(\prod_{i=1}^k (p_i - 1) \right)}{\prod_{i=1}^k p_i} .$$

Dans les versions plus récentes c'est plus lisible car on utilise des variables à la place des horribles " et "" (expression précédente et préprécédente).

Exercice (facile) : démontrer que la formule utilisée donne le même résultat que la formule usuelle.

— Nombre d'inversibles modulo n

On va écrire nos propres versions de la fonction phi.

Exercice 1 : Ecrire une fonction `NbInv1(n)` qui donne le nombre d'inversibles modulo n en comptant le nombre d'entiers compris entre 1 et n-1 qui sont premiers avec n.

Exercice 2 : Ecrire une fonction `NbInv2(n)` qui donne le nombre d'inversibles modulo n en utilisant la décomposition de n en facteurs premiers par la fonction `ifactors` et la formule usuelle.

Vous pouvez utiliser la boucle for qui parcourt une liste : `for truc in liste do ... od`.

Comparer la vitesse d'exécution des deux fonctions NbInv sur de grands entiers tirés au hasard.

On trouvera dans la partie facultative une autre formule qui utilise `factorset`.

— Ordre d'un élément de $\mathbb{Z}/n\mathbb{Z}$

Exercice 3 : Ecrire une fonction `Ordre1(a,n)` qui rend l'ordre de a modulo n si a est premier avec n , FAIL sinon. On calculera l'ordre en utilisant sa définition : le premier $0 < k$ tel que $a^k \bmod n = 1$.

On pourra utiliser `RETURN` pour s'arrêter dès qu'on a trouvé l'ordre.

Exercice 4 : On va essayer d'être plus malin.

Ecrire une fonction `Ordre2(a,n)` qui rend l'ordre de a modulo n si a est premier avec n , FAIL sinon. On calculera l'ordre en utilisant : le premier k diviseur de $\phi(n)$ tel que $a^k \bmod n = 1$.

Attention la fonction `divisors` retourne l'ensemble des diviseurs donc pas forcément dans

l'ordre croissant.

```
[ > divisors(1515);
```

Pour garantir l'ordre des diviseurs il faut faire

```
[ > sort(convert(divisors(1515),list));
```

Comparer l'efficacité des deux fonctions Ordre sur des n grands avec des a dont l'ordre modulo n est grand.

— Générateurs de $(\mathbf{Z}/n\mathbf{Z})^*$

Exercice 5 : Si l'ordre d'un élément de $(\mathbf{Z}/n\mathbf{Z})^*$ est égal au nombre d'éléments $\phi(n)$ de $(\mathbf{Z}/n\mathbf{Z})^*$, cet élément engendre le groupe $(\mathbf{Z}/n\mathbf{Z})^*$ et le groupe est cyclique.

Ecrire une fonction qui rende un générateur du groupe $(\mathbf{Z}/n\mathbf{Z})^*$ s'il existe, FAIL sinon. Vous pouvez utiliser les fonctions **phi** et **order** de Maple.

Grace à cette fonction construire une suite de couples $[n, \text{un générateur du groupe des inversibles modulo } n \text{ sinon FAIL}]$ pour n allant de 2 à 100.

Pouvez-vous conjecturer pour quels n le groupe $(\mathbf{Z}/n\mathbf{Z})^*$ a un générateur ? Si nécessaire agrandissez l'intervalle.

Exercice 6 : Ecrire une fonction qui rende l'ensemble, éventuellement vide, des générateurs du groupe $(\mathbf{Z}/n\mathbf{Z})^*$.

Vous pouvez utiliser la méthode simple ou mieux la remarque du cours qui dit que quand on a un générateur on sait construire les autres.

Vous pouvez aussi constater que quand il y a un générateur il y en a $\phi(\phi(n))$.