

## TP7 : Factorisation d'entiers par la méthode de Pollard

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

### – Méthode de Pollard

Exercice 1 : Ecrire une fonction correspondant à l'algorithme  $\rho$  de Pollard donné en cours pour trouver des facteurs de l'entier  $m$  : on part d'un  $x_0$  au hasard puis on construit la suite

$$x_{i+1} = (x_i^2 + 1) \bmod m \text{ et on calcule les } \text{pgcd}(x_{2i} - x_i, m) .$$

Si on trouve un pgcd différent de 1 on retourne les deux facteurs de  $m$  obtenus ainsi que l'indice  $i$

sinon on s'arrête lorsque  $2 m^{\left(\frac{1}{4}\right)} < i$  et on rend FAIL.

- Tester cette fonction avec un entier qui a deux grands facteurs premiers :  $m = p_1 p_2$  en ajustant la taille de  $p_1$  et  $p_2$  jusqu'à obtenir un temps de calcul notable tout en restant supportable.
- Faire plusieurs exécutions pour le même entier en partant de  $x_0$  différents.

### – Solution

On utilise la possibilité de construire une variable indicée de taille quelconque.

```
> facteur:=proc(m)
  local x,i,m1;
  x[0]:=rand() mod m;
  for i from 1 while i^4<=16*m do
    x[2*i-1]:=(x[2*i-2]*x[2*i-2]+1) mod
m;x[2*i]:=(x[2*i-1]*x[2*i-1]+1) mod m;
    m1:=igcd(x[2*i]-x[i],m);
    if m1<>1 then RETURN(m1,iquo(m,m1),i) fi
  od;
  FAIL
end;
```

Premier essai (on rappelle que les temps sont dépendants de la machine) :

```
[ > tire:=rand(10^6..10^7):
[ > p1:=nextprime(tire());p2:=nextprime(tire());m:=p1*p2;
      p1 := 1621597
      p2 := 9657619
      m := 15660765997543
[ > t:=time():facteur(m);time()-t;
```

1621597, 9657619, 934

0.047

Le temps est un peu petit, on compare le nombre d'itérations réelles 934 et la borne

$\left(\frac{1}{4}\right)$

$2^m$

```
[ > evalf(2*m^(1/4));
```

3978.627186

On a obtenu un diviseur nettement avant la borne.

On aimerait des temps dix fois plus grands: on va multiplier  $m$  par  $10^4$  donc  $p_1$  et  $p_2$  par

$10^2$  :

```
[ > tire:=rand(10^8..10^9):
```

```
[ > p1:=nextprime(tire());p2:=nextprime(tire());m:=p1*p2;
```

p1 := 753829717

p2 := 662222053

m := 499202662804149001

```
[ > evalf(2*m^(1/4));
```

53161.74390

```
[ > t:=time():facteur(m);time()-t;
```

753829717, 662222053, 31904

3.608

Le temps de calcul devient significatif.

On recommence ce qui va changer  $x_0$  :

```
[ > t:=time():facteur(m);time()-t;
```

753829717, 662222053, 31904

3.531

```
[ > t:=time():facteur(m);time()-t;
```

753829717, 662222053, 5514

0.422

On constate que le nombre de calculs donc le temps dépend fortement du  $x_0$  .

Exercice 2 : Tester la fonction **facteurs** du TP6 (algorithme naïf) avec les entiers  $m$  de l'exercice 1.

Trouver des entiers  $m = p_1 p_2$  pour lesquels le temps de la fonction **facteurs** soit raisonnable.

Quelle stratégie envisagez-vous pour factoriser un entier quelconque ?

### **Solution**

On teste d'abord avec le deuxième entier 499202662804149001 :

```
[ > facteurs(499202662804149001);
```

```
[ Warning, computation interrupted
```

On abandonne le calcul alors que ce nombre est de l'ordre de  $.5 \cdot 10^{18}$  et que la fonction

**facteurs** avait décomposé instantanément  $100!$  de l'ordre de  $10^{158}$  .

C'est vraiment la taille des facteurs qui compte.

On essaie avec le premier entier

```
> t:=time():facteurs(15660765997543);time()-t;
      [1621597, 1], [9657619, 1]
      3.703
```

Le temps est raisonnable bien que 80 fois plus grand que celui de la méthode de Pollard.

On peut envisager d'utiliser la fonction **facteurs** tant que les candidats diviseurs sont plus petits que  $10^5$  par exemple et utiliser la méthode de Pollard sur ce qui reste.

## S'il vous reste quelques minutes

Exercice 3 : Adapter la fonction de l'exercice 1 pour qu'elle traite un entier  $m$  produit de plus de deux grands facteurs premiers :

- lorsqu'on décompose  $m$  on n'est pas sûr que les facteurs soient premiers, on rappelle donc la fonction sur chaque facteur,
- on met un test de primalité en début de fonction pour ne pas travailler sur un facteur (très probablement) premier.

### Solution

```
> facteurplus:=proc(m)
  local x,i,m1;
  if isprime(m) then RETURN(m) fi;
  x[0]:=rand() mod m;
  for i from 1 while i^4<=16*m do
    x[2*i-1]:=(x[2*i-2]*x[2*i-2]+1) mod
  m;x[2*i]:=(x[2*i-1]*x[2*i-1]+1) mod m;
    m1:=igcd(x[2*i]-x[i],m);
    if m1<>1 then
      RETURN(facteurplus(m1),facteurplus(iquo(m,m1))) fi
    od;
  FAIL
end:
```

On teste :

```
> tire:=rand(10^6..10^7):
p1:=nextprime(tire());p2:=nextprime(tire());
p3:=nextprime(tire());p4:=nextprime(tire());
m:=p1*p2*p3*p4;

      p1 := 2589809
      p2 := 6561857
      p3 := 1727377
      p4 := 5367601
      m := 157565762451972291544191601

> facteurplus(m);
```

## — Et pour une poignée de minutes de plus

Ecrire un algorithme complet de factorisation de  $m$  en utilisant l'algorithme naïf pour extraire les facteurs premiers inférieurs à  $10^5$  (par exemple) et l'algorithme de Pollard sur ce qu'il reste de  $m$ .