

TP5 : Puissance rapide, Code cryptographique RSA

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

– Comment mesurer des temps de calculs et en faire des graphiques

Dans les exercices 1 et 2 vous devez comparer la vitesse de plusieurs algorithmes en faisant des séries de mesure de temps d'exécution et des graphiques de ces mesures.

Un exemple : je veux mesurer la vitesse de l'opérateur $^$ de Maple sur les entiers.

Je vais calculer 11^n en faisant varier n .

```
[ > t:=time():11^10:time()-t;
                                0
```

C'est visiblement trop petit, le temps n'est pas mesurable. On augmente la puissance :

```
[ > t:=time():11^100000:time()-t;
                                1.813
```

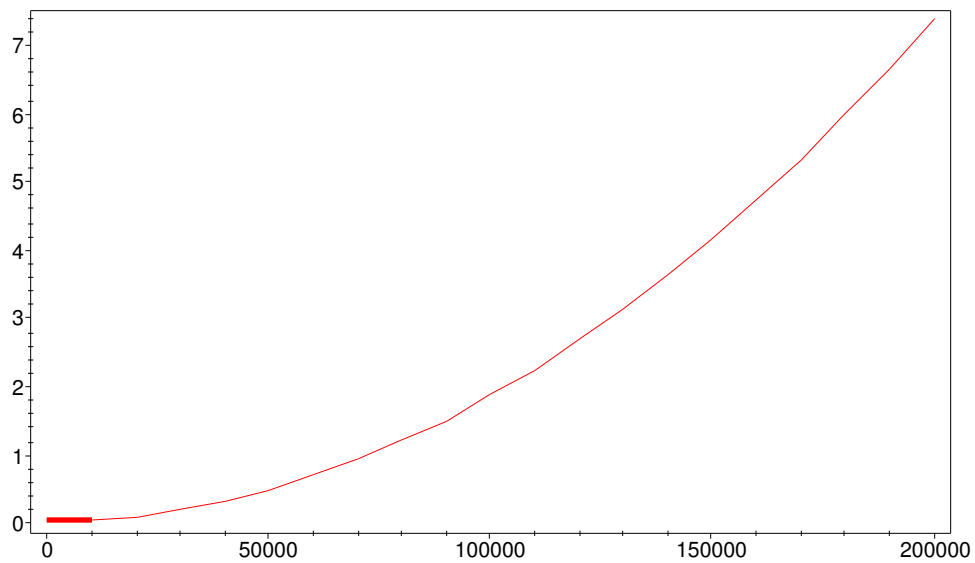
On commence à voir quelque chose (ceci peut dépendre de la machine).

Je vais maintenant faire varier n de 0 à 200000 par pas de 10000 et stocker des couples n , le temps de calcul de 11^n :

```
[ > s:=NULL:
    for n from 0 by 10000 to 200000 do
        t0:=time():11^n:t1:=time()-t0:s:=s,[n,t1]:
    od:
    s;
[0, 0], [10000, 0.015], [20000, 0.064], [30000, 0.171], [40000, 0.297], [50000, 0.454],
[60000, 0.671], [70000, 0.891], [80000, 1.172], [90000, 1.469], [100000, 1.843],
[110000, 2.188], [120000, 2.656], [130000, 3.094], [140000, 3.594], [150000, 4.109],
[160000, 4.688], [170000, 5.281], [180000, 5.937], [190000, 6.610], [200000, 7.359]
```

Je peux maintenant tracer cette suite de points :

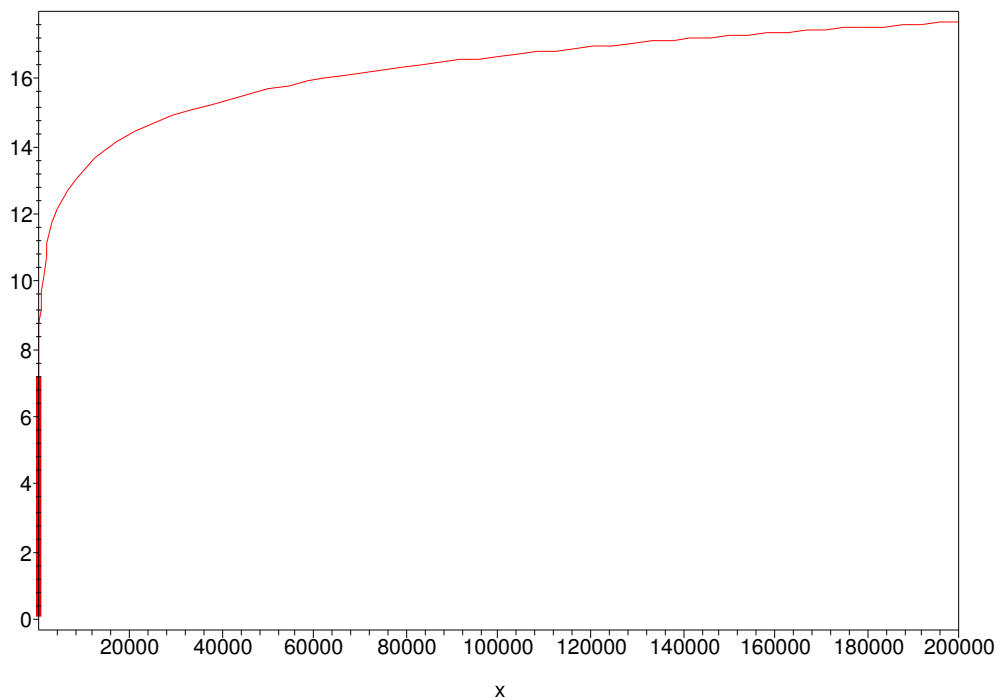
```
[ > plot([s]);
```



Et voilà un joli graphique du temps de calcul de 11^n en fonction de n .

Au fait le prof a dit (vite, vite, à la fin du cours) que le nombre d'opérations pour calculer a^n était en $\log_2(n)$.

```
> plot(log[2](x),x=1..200000);
```



Est-ce que ça ressemble ?

Où est l'arnaque ?

La réponse dans la suite de ce palpitant cours plein de suspens...

– Puissance rapide

On a besoin de calculer des puissances $x^n \bmod m$ avec x, n, m entiers de grande taille. On va comparer la vitesse de l'algorithme naïf et celle de l'algorithme rapide de calcul de puissances expliqué en cours.

Exercice 1 : Ecrire une fonction qui rende x^n en utilisant l'algorithme naïf par multiplications successives et une fonction qui rende x^n en utilisant l'algorithme rapide.

Chercher des entiers tels que les deux algorithmes aient des temps de calcul significativement différents.

En faisant varier n de manière exponentielle comparer graphiquement la vitesse de l'algorithme naïf et celle de l'algorithme rapide.

Que se passe-t-il si n est très grand (de l'ordre de 10^{50}) ?

Exercice 2 : Ecrire une fonction qui rende $x^n \bmod m$ en utilisant l'algorithme rapide.

Recommencer vos tests précédents en prenant m à 10 chiffres. Que constatez-vous ?

Essayer avec des puissances plus grandes et comparer graphiquement avec la puissance modulaire de Maple &^.

– Code cryptographique RSA

Vous allez fabriquer un code RSA :

- à l'aide de **isprime** construire deux nombres premiers distincts à 20 chiffres et n leur produit
- calculez $\phi(n)$, un nombre c compris entre 1 et $\phi(n) - 1$ premier avec $\phi(n)$
- vous pouvez maintenant diffuser votre méthode de cryptage : si quelqu'un souhaite vous envoyer un message m (de moins de 20 chiffres) crypté vous lui donnez c et n et il vous envoie $m_c = m^c \bmod n$.
- pour décoder le message crypté il faut calculer l'inverse de c modulo $\phi(n)$. Décodez le message crypté m_c .
- quelqu'un qui n'a en possession que c et n (la méthode de cryptage) doit calculer $\phi(n)$ pour pouvoir décrypter. Constatez que ça semble très long !