—

PHYSIQUE NUMÉRIQUE

TD n° 1

Des insectes et du chaos : l'équation logistique

Cet exercice étudie un modèle simple de l'évolution d'une population d'insectes au cours du temps : lors de l'année p, n_p insectes sont susceptibles de se reproduire, chacun produisant (en moyenne) α descendants [1] l'année suivante. On aura donc :

$$n_{p+1} = \alpha n_p$$

ainsi, si α est une constante,

$$n_p = \alpha^p n_0$$

ce qui donne une croissance [2] exponentielle du nombre d'insectes : c'est en effet ce qui se passe s'il n'y a pas de prédateur et si les ressources alimentaires sont infinies.

On peut tenter d'améliorer ce modèle en introduisant de façon simple l'effet du caractère fini des ressources disponibles : si la population d'insectes augmente, les ressources totales étant finies, les ressources par insecte diminuent ce qui entraîne une diminution de leur capacité à se reproduire, ainsi, α devient une fonction décroissante de n_p. On prend ici le cas le plus simple, celui d'une décroissance linéaire :

$$\alpha = r\left(1 - \frac{n_p}{n_{max}}\right)$$

où n_{max} est le nombre d'insectes qu'il faut pour consommer toutes les ressources disponibles et interdire toute reproduction l'année suivante. La constante r est le taux de reproduction maximum. On obtient alors :

$$n_{p+1} = r\left(1 - \frac{n_p}{n_{max}}\right) n_p$$

ou bien, en remplaçant n_p par $x_p = \dfrac{n_p}{n_{max}} \in [0, 1]$:

$$x_{p+1} = r(1 - x_p)x_p$$

Cette équation est célèbre sous le nom « d'équation logistique » [3] et sa simplicité apparente est trompeuse...

1. La première étape consiste à créer un programme dit « source » qui contienne les instructions nécessaires pour calculer l'évolution de x_p en fonction de p : à chaque pas de calcul, on passe de p à $p + 1$ et, connaissant x_p, on calcule x_{p+1}.

a- Invoquer d'abord un éditeur, `emacs` ou `gedit`, au choix :

```
emacs logistic.cpp &
```

ceci ouvrira une fenêtre dans laquelle on pourra taper les lignes du programme C++ et le sauvegarder sous le nom `logistic.cpp`. L'extension `.cpp` sert à se rappeler qu'il s'agit du

1. par exemple, en pondant, avant de mourir, des œufs, dont un nombre α écloreront l'année suivante.
2. si $\alpha > 1$.
3. Voir par exemple : Edward Ott, *Chaos in dynamical systems*, Cambridge University Press (1993).

fichier-source d'un programme écrit en C++ et permet aussi au compilateur de le reconnaître comme tel.

Prévoir des variables `x` et `r` de type `float` et une variable `p` de type `int`. Initialiser `x` à la valeur 0.1, cela donnera la valeur de x_0. Donner la valeur 2 à `r`. Écrire une boucle `for` qui remplace chaque année l'ancienne valeur de `x` par une nouvelle et qui écrive dans un fichier `logistic.res` sur deux colonnes les valeurs courantes de `p` et `x`. On fera varier `p` de 1 à 100.

Compiler :
`c++ logistic.cpp -o logistic`
où `logistic` est le nom du programme exécutable (on peut l'appeler aussi, au choix `logistic.x`, `logistic.exe` ou ... ce qu'on veut). Corriger les éventuelles erreurs détectées par le compilateur, recompiler. Répéter l'opération jusqu'à disparition complète des erreurs ! Lancer l'exécution du programme :
`./logistic`
Tracer le résultat après avoir invoqué `gnuplot` :
`plot 'logistic.res' with lines`
Imprimer le résultat : dans `gnuplot`
`set term post; set out '|lpr' ; replot ; quit`
où la barre verticale `|` est obtenue à l'aide des touches *AltGr* et *6* : elle représente un « *pipe* » (un tuyau) qui permet d'utiliser le résultat d'une commande Unix comme entrée d'une autre (l'impression, en l'occurence). Imprimer aussi le programme-source : `a2ps logistic.cpp`
b- Recommencer pour plusieurs valeurs de `r` : 2.5, 3, 3.5, 3.8... On pourra prévoir que le programme demande à l'utilisateur la valeur de `r`, puis la lise sur le clavier.

Se passe-t-il quelque événement intéressant ?
c- Au lieu de tracer les courbes x_p une à une en tapant les commandes à la main, on peut automatiser le processus en une animation pendant laquelle r pourra varier automatiquement entre 2 et 4.

Pour cela, il faut utiliser la commande `reread` (*relire*) de `gnuplot` : les commandes nécessaires sont incluses dans le fichier `logistic.plot`. Le copier à l'aide de la commande Unix :
`cp /u/depondt/pub/logistic.plot .` (ne pas oublier le point à la fin.)
La commande `cp` signifie *copy*, ce qui suit est : le fichier de départ à copier depuis le répertoire où il se trouve, suivi d'un point `.` qui désigne votre répertoire de travail. Ce fichier contient des commandes pour `gnuplot` :

```
r = r + 0.01
system "echo ".sprintf("%f",r)." > logistic.inp"
! ./logistic
plot [][0:1]'logistic.res' with lp ti "r = ".sprintf("%f",r)
pause 0.1
if ( r < 3.99 ) reread
```

La première ligne incrémente une variable `r`, ce qui suppose qu'elle ait d'abord été initialisée ; la deuxième écrit cette valeur dans un fichier `logistic.inp` ; la troisième lance l'exécution du programme `logistic` ; la quatrième trace la courbe contenue dans le fichier `logistic.res` en écrivant la valeur de `r` en haut à droite ; la cinquième attend 0.1 secondes ; la sixième renvoie à la première ligne. Si, dans `gnuplot`, on tape par exemple :
`r=2`
`load "logistic.plot"`
`gnuplot` fera une animation en faisant varier `r` de 2 à 4 par pas de 0.01.

Il reste toutefois à adapter le programme `logistic.cpp` : il suffit de lui faire lire la valeur de `r` dans le fichier `logistic.inp`.

Faire la modification, compiler, et lancer l'animation. On pourra faire plusieurs essais en changeant le point de départ, le pas d'incrémentation de `r`, la durée de la pause et la valeur maximum de `p`. Faire le lien avec ce qui a déjà été observé. Tenter de repérer la « fenêtre d'ordre ».

d- On veut maintenant étudier la question systématiquement en fonction de `r` : pour cela, on enchâsse la boucle initiale dans une autre boucle qui fait varier `r` de 2 à 4. On écrira un fichier sur 3 colonnes : `r`, `p`, `x`.

Les calculs seront faits pour `p` variant de 1 à 2000, mais on n'écrira que les 100 derniers pas pour `r`< 3 et les 1000 derniers sinon. On fera varier `r` par pas de 0.1. Tracer les résultats de ces calculs à l'aide de `gnuplot` avec la commande

```
plot 'logistique.res' using 1:3 with dots
```

Cette figure s'appelle « diagramme de bifurcation ». Une fois que tout marche bien, la refaire en faisant varier `r` avec un pas plus fin comme 0.005.

Comment faire le lien entre cette figure et les calculs précédents ? Que signifie dans ce cas l'expression « doublement de période » ? Qu'entend-on par « chaos déterministe » ? Donner une valeur de r approchée au-dessus de laquelle le système devient chaotique.

Imprimer cette figure ainsi que le programme modifié.

e- Refaire la même figure pour `r` variant de 3.74 à 3.745 par pas de 0.00001. Commentaire ?

2. L'une de caractéristiques essentielles du chaos déterministe mise en évidence par Poincaré dans les premières années du 20ème siècle est la sensibilité aux conditions initiales.

a- À l'aide du diagramme de bifurcation, on choisira une valeur de `r` propice. Modifier encore le programme pour faire simultanément deux trajectoires `x1` et `x2` dont les points de départ sont très proches, par exemple : `x1 = 0.01` et `x2 = 0.00999999`. On écrira les résultats sur 3 colonnes `p`, `x1`, `x2`. Dans `gnuplot` tracer les deux courbes à l'aide de la commande :

```
plot 'logistique.res' w l, '' u 1:3 w l
```

Commenter.

b- Tracer ensuite l'écart entre les deux trajectoires en échelle semi-logarithmique :

```
set logscale y
plot 'logistique.res' u 1:(abs($3-$2)) w l
```

On pourra faire plusieurs essais.

c- On écrit généralement cet écart comme $e^{\lambda p}$ où λ est « l'exposant de Lyapounov » qui caractérise la vitesse à laquelle deux systèmes quasiment identiques divergent. Essayer de déterminer l'exposant de Lyapounov en traçant (toujours en semi-log) :

```
plot 'logistique.res' u 1:(abs($3-$2)), a*exp(l*x)
```

et en donnant des valeurs judicieusement choisies à `a` et `l`.

d- Que vous inspire la sensibilité aux conditions initiales quant à la prédictibilité d'un tel système ?

3. De manière classique, si l'on dépose dans une boîte cubique des objets avec un nombre uniforme d'objets par unité de volume, le nombre d'objets que l'on peut placer dans la boîte croît comme ℓ^3, la taille de la boîte au cube. S'il s'agit d'un carré, à la place du cube, alors ce même nombre croît comme ℓ^2, et si c'est un segment de droite, on obtient ℓ^1. Voici donc un moyen de définir la dimensionalité d d'un espace $d = 3$ pour un volume, 2 pour une surface, 1

pour une ligne : on calcule le nombre n d'objets que contient l'espace :

$$n = a\ell^d$$

où a est une constante de proportionalité et d est la pente de la courbe donnant $\ln n$ en fonction de $\ln \ell$. Il existe cependant une autre façon de procéder, plus adaptée aux besoins de cet exercice. Imaginons, par exemple, une ligne [4] (de dimension 1) qui tracée une surface (de dimension 2) : si l'on divise la surface en petits carrés de côté ℓ, le nombre de carrés qu'il faudra pour recouvrir entièrement la ligne sera égal à la longueur L de la ligne divisée par ℓ, donc à ℓ^d, puisqu'ici $d = 1$:

$$n_\ell = \frac{L}{\ell^d} \;\Rightarrow\; d = \frac{\ln L}{\ln \ell} - \frac{\ln n_\ell}{\ln \ell}$$

et, quand $\ell \to 0$. On peut la généraliser par :

$$d = \lim_{\ell \to 0} \frac{\ln n_\ell}{\ln 1/\ell}$$

où n_ℓ est le nombre de boîtes de côté ℓ qu'il faut pour recouvrir l'espace étudié : cette dimension est (logiquement) appelée, en anglais, « *the box-counting dimension* ».

a- On se place dans l'espace de dimension 1 constitué par le segment $[0, 1]$ qui contient les valeurs prises par x_p. On cherche la dimension du sous-espace qui contient ces valeurs, en excluant les zones vides, s'il y en a.

On doit alors écrire un programme, appelé par exemple `boxdim.f90`, capable de lire le fichier `logistic.res` produit par le premier programme du TP (que l'on modifiera pour qu'il fasse varier p de 1 à 1000), afin de lire les valeurs de x_p : ceci doit être fait dans une première boucle interne. Au cours de cette lecture on testera si `int(x/l)` est égal à `k` : `l` représentant ℓ ci-dessus, la taille des boites considérées et `k` un entier qui repère une boite. Si la réponse est oui, on incrémentera `n_box` le nombre de boites pleines et on sortira de la boucle de lecture (par un `exit`).

La valeur de `k` est fournie par une autre boucle qui enchâsse la première qui fait varier `k` de 0 à `m`. À chaque sortie de la boucle de lecture, penser à revenir au début du fichier par un `rewind` car on doit relire le fichier plusieurs fois.

Enfin, une troisième boucle englobera les deux autres : on fera varier un entier `q` de 1 à 10, ce qui donnera `m=2**q` et `l=1.0/m`. Pour chaque nouvelle valeur de `q`, on calculera la dimension d et on l'écrira dans un fichier en même temps que `q`. Ne pas oublier de réinitialiser ce qui doit l'être...

b- Tracer d en fonction de `q` pour plusieurs valeurs de r. Commenter.

—

Note : si vous en avez assez de taper `with lines` ou `w l` pour chaque graphe tracé avec `gnuplot`, vous pouvez créer dans votre répertoire racine (pour être sûr(e) d'y être, taper `cd ~` ou `cd $HOME`) un fichier `.gnuplot` (ne pas oublier le point !) qui contienne la ligne
`set style data lines`
et `with lines` deviendra l'option par défaut.

4. pas forcément droite.