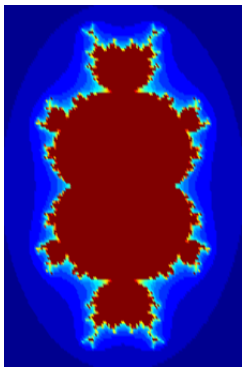
MG2 – Logiciels scientifiques – Automne 2010

Solution de l'épreuve d'examen

1 Suite complexe

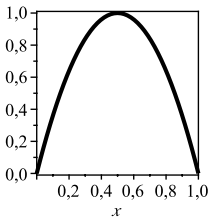
Il faut remplacer la fonction $z \mapsto z^2 + c$ par la fonction $z \mapsto z^3 + c$ dans le programme Matlab donné en cours. On obtient une variante de l'ensemble de Mandelbrot :



2 Générateur de nombres aléatoires

Ce problème peut se traiter avec Matlab ou Maple.

1. Graphe de f :



2. Pour $u_0, \ldots, u_{10}$ on obtient les valeurs suivantes : $1, 3, 9, 2, 8, 5, 9, 1, 4, 9, 1$.

3. Ce générateur est d'un intérêt limité car, en faisant une statistique sur une centaine de termes :

```
cpt=zeros(1,10);
x=0.1; cpt(2)=1;
for i=1:100
    x=4*x*(1-x); a=floor(10*x); cpt(a+1)=cpt(a+1)+1;
end
cpt
```

on s'aperçoit que la répartition des 10 chiffres est loin d'être uniforme.

Remarque.— En réalité, le calcul numérique des termes successifs de la suite (x_n) donne rapidement des résultats complètement faux. En effet, on peut montrer que la fonction f (appelée *logistic map* dans la littérature) a une dynamique chaotique, d'où la propriété de *sensibilité aux conditions initiales* qui a pour effet d'amplifier les erreurs d'approximation. Avec Maple, on peut obtenir les valeurs exactes des nombres rationnels x_i mais il n'est pas possible de pousser les calculs bien loin.

3 Modèle proie-prédateur

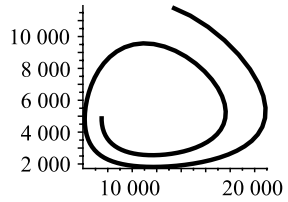
Ce problème peut se résoudre en partie avec Matlab et en partie (ou totalité) avec Maple.

1. On trouve $x_{19} = -175.5955706$: en 2019 le nombre de harengs devient négatif, ce qui montre la faiblesse du modèle.

2. On a $x_\infty = y_\infty = 0$. Pour s'en convaincre, on peut calculer un grand nombre de termes des suites (x_n) et (y_n) et constater qu'ils tendent vers 0 ou, mieux, résoudre la récurrence linéaire (fonction `rsolve` de Maple) et calculer les limites quand $n \to \infty$:

```
> eqs:=x(n+1)=27/25*x(n)-4/25*y(n),y(n+1)=7/50*x(n)+21/25*y(n);
> s:=rsolve({eqs,x(0)=7500,y(0)=5000},{x(n),y(n)});
> xn:=evalc(Re(subs(s,x(n)))); yn:=evalc(Re(subs(s,y(n))));
> map(limit,[xn,yn],n=infinity);
```

3. Le tracé de la « courbe » $n \mapsto (x_n, y_n)$ ressemble à ceci :

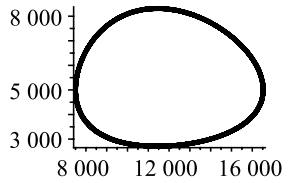


et on voit que la spirale va en s'élargissant : empiriquement, on en conclut que le système ne tend pas vers un équilibre écologique. Cependant il y a deux points fixes qui sont $(0, 0)$ et $(\frac{80000}{7}, \frac{2000000}{157})$, qu'on peut obtenir grâce à la fonction `solve` de Maple.

4. Le système différentiel est :

$$\begin{cases} x'(t) &= 4\,x(t) - 8/10000\; x(t)\,y(t) \\ y'(t) &= -8\,y(t) + 7/10000\; x(t)\,y(t) \end{cases}$$

En l'intégrant grâce à la fonction `dsolve` (option `numeric`) de Maple, ou en utilisant directement `DEplot`, on obtient la courbe intégrale suivante :



qui montre que, comme à la question 3, on ne tend pas vers un équilibre écologique.

4 Pendule forcé amorti

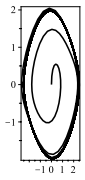
Ce problème est analogue à celui du mouvement d'un navire, vu en cours.

1. L'erreur sur $\theta(t)$ est équivalente à $\frac{1}{2688}a^3t^8$. Pour le voir avec Maple, on résout les deux équations différentielles sous forme de séries (`dsolve` avec l'option `series`) et on fait la différence des parties principales :

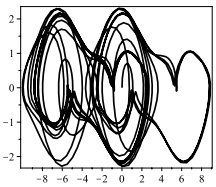
```
> deq:=diff(theta(t),t$2)+k*diff(theta(t),t)+sin(theta(t))=a*cos(omega*t);
> deq2:=subs(sin(theta(t))=theta(t),deq);
> s1:=dsolve({deq,theta(0)=0,D(theta)(0)=0},theta(t),series,order=9); sol1:=subs(s1,theta(t));
> s2:=dsolve({deq2,theta(0)=0,D(theta)(0)=0},theta(t),series,order=9); sol2:=subs(s2,theta(t));
> series(sol1-sol2,t=0,9);
```

2. On obtient les trajectoires de phases $t \mapsto (\theta(t), \dot{\theta}(t))$ en procédant comme pour le navire. Les courbes suivantes sont obtenues en faisant varier t de 0 à 400.

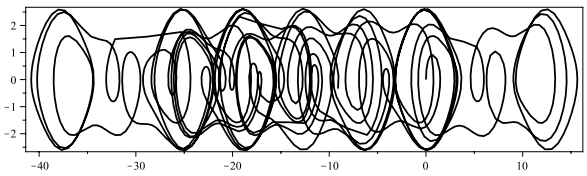
– pour $a = 0.9$, il y a un cycle limite :



– pour $a = 1.15$, il y a probablement un cycle limite ou alors un régime quasi-périodique :



– pour $a = 1.5$, le mouvement est chaotique :

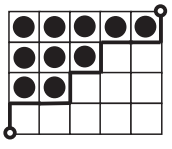


5 Analyse d'un jeu

1. Pour le jeu de Chomp à 6 lignes et 5 colonnes, il y a 462 positions possibles. On pourrait les compter en programmant 5 boucles imbriquées (ici en Matlab) :

```
cpt=0;
for i1=0:6
    for i2=0:i1
        for i3=0:i2
            for i4=0:i3
                for i5=0:i4
                    cpt=cpt+1;
                end
            end
        end
    end
end
cpt
```

mais il est plus astucieux de remarquer que, dans le jeu de Chomp à n lignes et p colonnes, se donner une position revient à se donner un chemin montant du coin inférieur gauche au coin supérieur droit (voir schéma ci-dessous pour $n = 4$ et $p = 5$), donc choisir p segments horizontaux (ou n verticaux) parmi $n + p$. Le nombre de positions est donc $\binom{n+p}{p} = \binom{n+p}{n}$. Pour $n = 6$ et $p = 5$, on trouve $\binom{11}{5} = 462$.



2. Un théorème de Von Neumann vu en cours permet d'affirmer qu'il y a une stratégie gagnante pour l'un des deux joueurs.

3. L'analyse du jeu se fait comme dans l'exemple du jeu de Nim à plusieurs tas, vu en cours. On commence par écrire une procédure qui énumère toutes les filles d'une position donnée. Pour cela on remarque qu'une fille de $(i_1, i_2, \ldots, i_k)$ est obtenue :
– soit en jouant en première colonne,
– soit en mettant i_1 devant une fille de $(i_2, \ldots, i_k)$.

On écrit donc une fonction récursive `f(n,L)` qui, étant donnée une position $L = (i_1, \ldots, i_k)$ et un entier $n \leqslant i_1$, renvoie la liste des positions obtenues en jouant en colonne 1, lignes $p \leqslant n$ à partir de L, puis une fonction récursive `filles(L)` utilisant `f`, qui donne la liste de toutes les filles d'une position L.

Voici le code en Maple :

```
> f:=proc(n,L)
     if n=1 then [[]]
     else [map(u->min(n-1,u),L),op(f(n-1,L))]
     end if
  end proc:
> filles:=proc(L)
     local t,L1,L2,L3;
     if L=[] then []
     else
        t:=L[1];
        L1:=L[2..nops(L)];
        L2:=filles(L1);
        L3:=map(u->[t,op(u)],L2);
        [op(L3),op(f(t,L))]
     end if;
  end proc:
```

il reste à programmer les fonctions `perdante` et `gagnante` comme dans l'exemple du cours :

```
> perdante:=L->not gagnante(L):
> gagnante:=proc(L)
     option remember;
     if L=[] then true
     else convert(map(perdante,filles(L)),`or`)
     end if
  end proc:
```

Alors l'instruction :

```
> L:=[4,4,3,3,1]: gagnante(L);
```

renvoie *true* et l'instruction :

```
> select(perdante,filles(L));
```

donne les coups gagnants à partir de L, à savoir $(4, 4, 3, 1, 1)$ et $(2, 2, 2, 2, 1)$.