



# scilab

## VIII - Plus d'exercices

Lycée

Auteur : Raymond Moché

**Mots-clefs** : boucles « tant que » et « pour », commandes « taille », « floor », « find », « reste », « liste\_preiers », « A(\$) », instruction conditionnelle.

### Références :

✓ Livret de présentation de « scilab pour les lycées » (2010) et Mise à jour et compléments (mars 2011) :

<http://www.scilab.org/education/lycee/docs>

✓ Aide scilab 5.3.2

[http://help.scilab.org/docs/5.3.2/fr\\_FR/index.html](http://help.scilab.org/docs/5.3.2/fr_FR/index.html)

### Liste des exercices :

**Énoncé n° 1** : [\*\*\*\*] Crible d'Ératosthène.

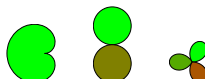
**Énoncé n° 2** : [\*]  $n$  est-il premier ? (suite de l'exercice 1)

**Énoncé n° 3** : [\*\*] Nombre premier suivant (suite de l'exercice 1).

**Énoncé n° 4** : [\*\*] Factorisation en nombres premiers (suite de l'exercice 1).

**Énoncé n° 5** : [\*\*\*] Peut-on faire l'appoint ?

**Énoncé n° 6** : [\*\*\*] Records.



---

**Énoncé 1** [\*\*\*\*] : Crible d'Ératosthène.

On demande une fonction « scilab » qui, à tout nombre entier  $n \geq 2$  associe la liste des nombres premiers qui lui sont inférieurs ou égaux, rangée dans l'ordre croissant.

**Mots-clefs** : boucles « tant que », commandes « taille », « floor », « find », « reste », « liste\_preiers », « A(\$) ».

« A(\$) » est le dernier terme du vecteur  $A$ .

Le crible d'Ératosthène est supposé connu. L'algorithme ci-dessous : « Erato.sci » *le reproduit exactement*. On écrit la suite  $A$  des nombres entiers successifs de 2 à  $n$ . 2 est le premier nombre premier  $\leq n$ . On le conserve en effaçant tous ses multiples dans  $A$  (commande «  $A(I) = []$  »). La suite ainsi obtenue continue à s'appeler  $A$ . Si  $n > 2$ , ses deux premiers éléments (2 et 3, évidemment) sont des nombres premiers  $\leq n$ , rangés dans l'ordre croissant. On recommence les effacements si nécessaire jusqu'à ce que le dernier nombre premier obtenu soit le dernier terme de  $A$ .

Listing 1 – Erato.sci

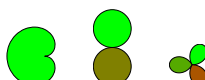
```
function A=Erato(n)
A=2:n; // n entier >=2.
k=1;
while A(k) <= sqrt(A($))
    Restes=A(k+1:taille(A))-A(k)*floor(A(k+1:taille(A))/A(k))
    I=find(Restes==0);
    if taille(I)>0
        A(k+I)=[];
    end;
    k=k+1;
end;
endfunction
```

L'algorithme « Erato.sci » est téléchargeable. Une commande « liste\_preiers(n) » de « scilab » a exactement la même fonction (mêmes entrées et sorties). Nous ne savons pas si elle reproduit le crible d'Ératosthène.

Par exemple, après avoir chargé l'algorithme « Erato.sci » dans la console, la commande

```
-->A=Erato(100000)
```

permet de trouver qu'il y a 9592 nombres premiers  $\leq 100\,000$ , le dernier étant 99 991. L'instruction conditionnelle permet d'éviter le piège «  $a+[ ]=a$  ». L'utilisation « floor » pour le calcul de restes permet de se passer de la commande « reste » qui n'accepte pas « reste(A,a) » où  $A$  est une matrice d'entiers et  $a$  un entier.



---

**Énoncé 2** [\*] :  $n$  est-il premier ? (suite de l'exercice n° 1)

Étant donné un entier  $n \geq 2$  et en utilisant la fonction « Erato.sci » de l'exercice 1 ou la commande « liste\_premiers », fabriquer un algorithme qui répond si  $n$  est ou non un nombre premier.

---

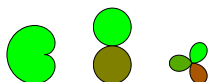
**Mots-clefs** : commande « liste\_premiers ».

---

C'est assez évident :

Listing 2 –  $n$  est-il premier ?

```
n=input('n=');
A=liste_premiers(n);
if n==A($) then
    afficher(+string(n)+' est un nombre premier. ');
else
    afficher(+string(n)+' n ' est pas un nombre premier. ');
end
```



---

**Énoncé 3** [\*] : Nombre premier suivant (suite de l'exercice n° 1).

Fabriquer un algorithme qui associe à tout entier  $n \geq 2$  le plus petit nombre premier  $P_n$  strictement supérieur à  $n$ .

---

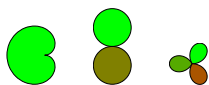
**Mots-clefs** : « liste\_premiers ».

---

On peut s'appuyer sur la fonction « Erato.sci » de l'exercice 1 (qu'il faut charger dans « scilab » avant d'exécuter l'algorithme ci-dessous) ou, de manière équivalente, sur la commande « liste\_premiers ». Il suffit de repérer quand la liste des nombres premiers  $\leq m$ , pour  $m = n, \dots$ , augmente d'une unité.

Listing 3 – Nombre premier suivant

```
n=input('n=');  
m=n+1;  
while taille(liste_premiers(n))==taille(liste_premiers(m))  
    m=m+1;  
end;  
Pn=m;  
afficher(Pn);
```



---

**Énoncé 4** [**\*\***] : Factorisation en nombres premiers (suite de l'exercice n° 1).

Fabriquer un algorithme qui à tout entier  $n \geq 2$  associe la suite de ses facteurs premiers et la suite de leurs puissances respectives.

---

**Mots-clefs** : commandes « liste\_preemiers », « find », extraction d'une sous-matrice, effacement de composantes dans un vecteur.

---

L'algorithme ci-dessous est très simple : on passe en revue la liste des nombres premiers  $\leq n$  obtenue à l'aide de la fonction « Erato.sci » (à charger sur la console) ou à l'aide de la commande « liste\_preemiers » et on regarde jusqu'à quelle puissance chacun d'eux divise  $n$ . Par exemple, si un tel nombre premier ne divise pas  $n$ , il le divise avec la puissance 0. On efface ces facteurs parasites avant d'obtenir le résultat définitif, qui est présenté sur la forme d'un tableau à 2 lignes, la première étant la ligne des facteurs, la seconde la ligne de leurs puissances.

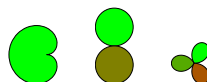
Listing 4 – Factorisation en nombres premiers

```
// Factoriser un entier donné n>=2.
// Charger la fonction Erato.sci.
function F=factorisation(n)
Facteurs=Erato(n);
Puissances=[];
for i=1:taille(Facteurs)
    j=1;
    while reste(n,Facteurs(i)^j)==0
        j=j+1
    end
    Puissances=[Puissances ,j-1];
end
K=find(Puissances==0);
Facteurs(K)=[];
Puissances(K)=[];
F=[Facteurs ;Puissances];
endfunction
```

**Exemple** : mettre 12345 sous la forme d'un produit de nombres premiers.

On charge dans la console « Erato.sci » puis « Factorisations.sci » (téléchargeable sur le site). Ensuite,

```
-->F=factorisation(12345)
F =
     3.     5.     823.
     1.     1.     1.
-->
```



---

**Énoncé 5** [\*\*\*] : Peut-on faire l'appoint ?

Quelqu'un doit payer un achat. Le commerçant n'ayant pas de monnaie lui demande de faire l'appoint, c'est à dire de donner exactement la somme due. Est-ce que c'est possible ? Si oui, comment faire, c'est à dire, quel algorithme utiliser ?

---

**Mots-clefs** : commandes « find », « diag », produit de 2 matrices, extraction de sous-matrices.

---

Les données sont précisées dans les commentaires de l'algorithme Algorithme « Appoint.sci » (téléchargeable sur le site) ci-dessous. C'est un algorithme trivial dans sa conception : on calcule toutes les sommes que l'on peut payer avec les pièces disponibles pour l'achat et quand on rencontre la somme que l'on a à payer, on note la répartition des pièces qui a permis d'obtenir ce total. La sortie de l'algorithme est une matrice éventuellement vide dont chaque ligne décrit une solution.

Listing 5 – En examinant tous les cas possibles

```
// Peut-on faire l'appoint ?
function solutions=achat(p,a,b,c,d,e,f,g,h)
// a,b,c,d,e,f,g,h désignent respectivement les
// nombres de pièces de 1 c, 2 c, 5 c, 10 c, 20 c, de 50 c, de 1 euro et
// de 2 euros disponibles pour l'achat (nombres entiers >=0) ; p désigne
// le prix de l'achat (en centimes).
solutions=[];
for ja=0:a
  for jb=0:b
    for jc=0:c
      for jd=0:d
        for je=0:e
          for jf=0:f
            for jg=0:g
              for jh=0:h
                s=200*jh+100*jg+50*jf+20*je+10*jd+5*jc+2*jb+ja;
                if s==p then
                  solutions=[solutions;[ja,jb,jc,jd,je,jf,jg,jh]];
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;
endfunction
```

```

-->Sol=achat(12,0,6,2,0,0,0,0)
Sol =

    0.    1.    2.    0.    0.    0.    0.    0.
    0.    6.    0.    0.    0.    0.    0.    0.

-->

```

Cela signifie qu'il y a 2 façons de payer un achat de 12 centimes. La première consiste à donner une pièce de 2 centimes et 2 pièces de 5 centimes. Au contraire, avec les mêmes pièces, on ne peut pas payer un achat de 13 centimes, ce qui donne :

```

-->Sol=achat(13,0,1,2,0,0,0,0)
Sol =

    []

-->

```

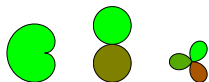
L'algorithme « Achat.sci » (téléchargeable sur le site) calcule aussi tous les paiements exacts possibles. Il repose sur des manipulations de matrices et utilise des commandes plus sophistiquées. On peut le considérer comme difficile.

Listing 6 – En examinant tous les cas possibles à l'aide de manipulations de matrices

```

// Peut-on faire l'appoint ?
function Sol=achat(p,a,b,c,d,e,f,g,h)
    E=[a,b,c,d,e,f,g,h];
    // a,b,c,d,e,f,g,h désignent respectivement les nombres de pièces de
    // 1 c, 2 c, 5 c, 10 c, 20 c, de 50 c, de 1 euro et de 2 euros
    // disponibles pour l'achat (nombres entiers >=0) ; p désigne le prix
    // en c.
    A=[0:E(1)]';
    for i=2:8
        B=[];
        r=size(A,'r');
        for j=0:E(i)
            C=[A,j*ones(r,1)];
            B=[B;C];
        end
        A=B;
    end
    val=diag([1,2,5,10,20,50,100,200]);
    sommes=sum(A*val,"c");
    lignessol=find(sommes==p*ones(sommes));
    Sol=A(lignessol,:);
endfunction

```



---

**Énoncé 6** [\*\*\*] : Records.

On appelle record d'une suite donnée  $S$  de nombres entiers son premier élément ainsi que tout terme de cette suite qui est strictement plus grand que tous les termes qui le précèdent. Le problème est de compter le nombre de records de  $S$  à l'aide d'un algorithme.

---

**Mots-clefs** : Boucles « tant que » et « pour », instruction conditionnelle, commande « find ».

---

La solution présentée ci-dessous consiste, à partir du premier élément de  $S$ , à supprimer tous les termes de  $S$  qui lui sont inférieurs ou égaux et qui ne peuvent être des records. On appelle encore  $S$  la suite ainsi expurgée. Si elle est de longueur au moins 2, on refait de même en partant de son deuxième élément (qui est le deuxième record de la suite  $S$  initiale). On réitère le procédé si c'est nécessaire. La dernier état de  $S$  (obtenu quand le dernier record calculé est le dernier élément de la liste) est la liste des records de la suite  $S$  initiale.

Listing 7 – RecordsParEffacement.sce

```
S=input('S=');// S est une liste de nombres entiers.
i=1;
while i<taille(S)
    J=find(S(i+1:taille(S))<=S(i));
    if taille(J)>0
        S(i+J)=[];
    end
    i=i+1;
end
afficher('La liste des records de S est ');
afficher(S);
```

Cette solution est inspirée par le crible d'Ératosthène (voir la fonction « scilab » « Erato.sci » de l'exercice 1). L'algorithme « RecordsParEffacement.sce » est téléchargeable.

