

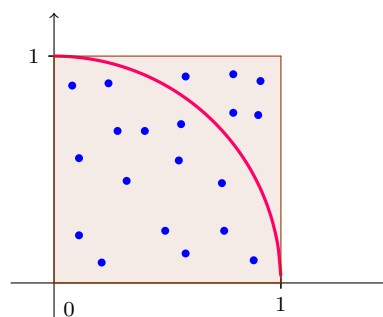
Exemples d'algorithmes probabilistes

1 Méthode de Monté-Carlo

1.1 Principe

Le calcul de π par la méthode de Monte-Carlo consiste à tirer au hasard des nombres x et y dans l'intervalle $[0; 1]$.

Si $x^2 + y^2 < 1$ le point $M(x, y)$ appartient à un quart de disque de rayon 1. La probabilité pour qu'il en soit ainsi est le rapport des aires du quart de disque de rayon 1 et du carré de côté 1 et soit $\frac{\pi}{4}$.



1.2 Mise en œuvre

Ecrire un algorithme, qui pour une valeur n donnée, restitue une estimation de π par la méthode de Monté-Carlo, en procédant à n tirages.

Solution :

Version SCILAB

```
function mc=MC(n)
S=0;
for i=1:n do,
a=rand();
b=rand();
if (a^2+b^2<1) then, S=S+1; end;
end;
mc=4*S/n;
endfunction
```

Version XCAS

```
MC(n):={
local k,S,a,b;
pour k de 1 jusque n faire
a:=alea([0,1]);
b:=alea([0,1]);
si a^2+b^2<1 alors S:=S+1; fsi;
fpour;
return 4*S/n;
}
```

☞ Si cette méthode est élégante, elle n'est par-contre pas des plus efficaces pour le calcul d'une valeur approchée de π . La méthode suivante donne, par exemple, de bien meilleurs résultats :

1.3 Formule de Bailey–Borwein–Plouffe

Il a été établi que :

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

Ecrire un algorithme réalisant le calcul de

$$S(n) = \sum_{k=0}^n \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right], \text{ pour } n \text{ donné.}$$

Traduire, sous SCILAB et sous XCAS, cet algorithme.

Comparer les résultats obtenus avec les valeurs approchées de π fournies par le logiciel. **Solution :**

Version SCILAB

```
function m=S(n)
sp=0;
for i=0:1:n do,
sp=sp+1/16^k*(4/(8k+1)-2/(8k+4)-1/(8k+5)-
1/(8k+6));;
end;
s=sp,
endfunction
```

Version XCAS

```
S(n):={
local k,Sp;
Sp:=0;
pour k de 0 jusque n faire
Sp:=Sp+1/16^k*(4/(8k+1)-2/(8k+4)-1/(8k+5)-
1/(8k+6));
fpour;
return Sp;
}
```

2 Comparaison d'algorithmes probabilistes et exhaustifs

On considère l'ensemble Ω constitué des entiers $\{1; 2; 3; \dots; 15\}$.

On se propose de déterminer le nombre de triplets $(a; b; c) \in \Omega^3$ vérifiant $a < b < c$.

Nous allons considérer deux approches du problème :

Méthode probabiliste

Ecrire un algorithme simulant n tirages de trois éléments de l'ensemble Ω et comptabilisant le nombre de triplets vérifiant la relation $a < b < c$.

Méthode exhaustive

Ecrire un algorithme parcourant tous les triplets possibles de l'ensemble Ω et comptabilisant le nombre de triplets vérifiant la relation $a < b < c$.

On va essayer, dans la suite, de mettre en évidence quelques différences entre ces deux méthodes.

1. Dans un premier temps traduire, sous SCILAB et sous XCAS, ces algorithmes.

☞ Pour la version probabiliste, on réalisera un programme recevant en entrée le cardinal de l'ensemble Ω et le nombre de tirages à effectuer, et restituant en sortie l'estimation du nombre de triplets vérifiant la relation $a < b < c$.

Solution :

versions probabilistes

Version SCILAB

```
function card=cardpb(n,p)
S=0;
for i=1:1:p do,
a=floor(rand()*n)+1;
b=floor(rand()*n)+1;
c=floor(rand()*n)+1;
if (a<b) & (b<c) then, S=S+1; end;
end;
card=S/p^n^3;
endfunction
```

Version XCAS

```
cardpb(n,p):={
local k,a,b,c,S;
S:=0;
pour k de 1 jusque p faire
a:=alea(n-1);
b:=alea(n-1);
c:=alea(n-1);
si a<b && b<c alors S:=S+1 fsi;
fpour;
return approx(n*n*n*S/p,2);
}
```

versions exhaustives

Version SCILAB

```
function cdt=cardit(n)
S=0;
for a=1:1:n do,
for b=1:1:n do,
for c=1:1:n do,
if (a<b) & (b<c) then S=S+1 ; end ;
end ;
end ;
end ;
cdt=S;
endfunction
```

Version XCAS

```
cardit(n):={
local a,b,c,S;
S:=0;
pour a de 1 jusque n faire
pour b de 1 jusque n faire
pour c de 1 jusque n faire
si a<b && b<c alors S:=S+1 fsi ;
fpour ;
fpour ;
fpour ;
return S ;
}
```

- Exécuter ces programmes pour différentes valeurs du cardinal de $n = \text{card}(\Omega)$ (par exemple, $n = 10$ $n = 50$ $n = 100$).

☞ pour la version probabiliste, on pourra faire varier le nombre de tirages.

- Comparer les temps de calculs de chacun des programmes et les résultats obtenus.

La comparaison de ces temps de calculs peut être davantage mise en évidence par l'utilisation de la fonction `time`, disponible dans beaucoup de langages de programmation. Elle permet d'obtenir le temps d'exécution d'une commande ou d'un programme. Par exemple :

- sous XCAS, l'instruction `time(cardit(10)) [1]` renvoie le temps d'exécution de l'instruction `cardit(10)` donc du temps d'exécution du programme itératif pour $n = 10$.
- sous SCILAB, l'instruction `time()` renvoie le temps écoulé depuis l'appel précédent de `time()`. Ainsi, pour obtenir le temps d'exécution du programme itératif, on procèdera par différence : `time(); cardit(10); t=time()`
On peut également utiliser les instructions `tic` et `toc` fournies dans le module `lycée`, `tic()` déclenchant un chronomètre et `toc()` arrêtant ce chronomètre. Ainsi, l'instruction précédente devient : `tic(); cardit(10); t=toc()`

Voici quelques indications, permettant d'orienter la réflexion sur ces temps de calculs :

- Les temps de calcul semblent-ils proportionnels à la valeur de n . Pourquoi ?
- Comment améliorer le temps de calcul du programme dans sa version exhaustive ? **Solution :**

Version SCILAB

```
function cdt=cardit(n)
S=0;
for a=1:1:n do,
for b=a+1:1:n do,
for c=b+1:1:n do,
if (a<b) & (b<c) then S=S+1 ; end ;
end ;
end ;
end ;
cdt=S;
endfunction
```

Version XCAS

```
cardit(n):={
local a,b,c,S;
S:=0;
pour a de 1 jusque n faire
pour b de a+1 jusque n faire
pour c de b+1 jusque n faire
si a<b && b<c alors S:=S+1 fsi ;
fpour ;
fpour ;
fpour ;
return S ;
}
```

- Suivant les valeurs de n , le temps gagné est-il significatif ?
- Quelles sont les limites de cette méthode en termes de temps d'exécution ?

☞ Afin de mieux visualiser ces temps d'exécution, on peut créer une liste des temps d'exécution du programme pour différentes valeurs de n , puis représenter géométriquement ces valeurs obtenues.

Version SCILAB

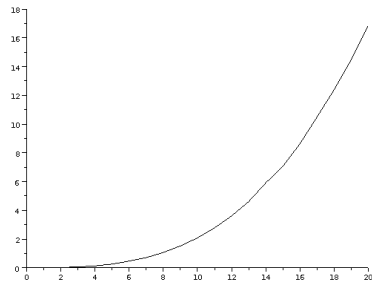
```
for k=10:10:200 do, timer();
cardit(k);
t(k/10)=timer();
end;
plot2d(t)
```

Version XCAS

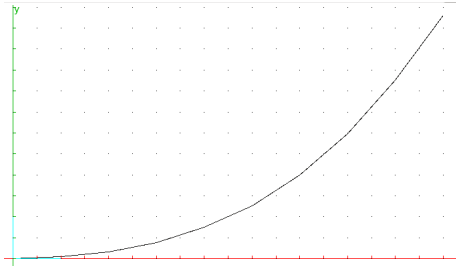
```
L:=seq(time(cardit(n))[0],n,10,200,10))
plotlist(L)
```

Solution :

version SCILAB



version XCAS



4. Rechercher un algorithme plus rapide, que celui dans sa version probabiliste, et donnant la solution exacte du problème.
 ☞ il s'agit ici de résoudre un problème mathématique...