CORRIGÉ EXERCICES TD N°4.

Fonction factorielle récursive

```
> factr := proc(n:nonnegint)
> if n = 0 then RETURN(1)
>                 else RETURN(n*factr(n-1))
> fi;end;
```

factr :=

proc(*n*::*nonnegint*) **if** $n = 0$ **then** RETURN(1) **else** RETURN($n \times$ factr($n-1$)) **fi end**

```
> factr(10);
```

3628800

Fonction factorielle itérative.

```
> facti := proc(n:nonnegint)
> local i, p;
> p := 1;
> for i from 1 to n do
>          p := p*i
>          od;
> RETURN(p)
> end;
```

facti := **proc**(*n*::*nonnegint*) **local***i*, *p*; $p := 1$; **for** *i* **to** *n* **do** $p := p \times i$ **od** ; RETURN(*p*) **end**

```
> facti(10);
```

3628800

Fonction qui calcule la somme des chiffres d'un nombre (version récursive)

```
> somme_chiffrer := proc(n:nonnegint)
> if n = 0 then 0
>                 else irem(n,10) + somme_chiffre(iquo(n,10))
  fi;
  end;
```

somme_chiffre :=

proc(*n*::*nonnegint*) **if** $n = 0$ **then** 0 **else** irem(n, 10) + somme_chiffre(iquo(n, 10)) **fi end**

```
> somme_chiffrer(114505);
```

16

Fonction qui calcule la somme des chiffres d'un nombre (version itérative)

```
> somme_chiffrei := proc(n : nonnegint)
> local q, r, s;
> s := 0; q := n;
> while q <> 0 do
>        q := iquo(q,10,'r');
>        s := s + r
>        od;
> RETURN(s)
> end;
```

$$somme\_chiffrei := \mathbf{proc}(n::nonnegint)\ \mathbf{local} q, r, s;\ s := 0\,;\ q := n\,;\ \mathbf{while}\, q \neq 0\, \mathbf{do}\, q := \mathrm{iquo}(q,\ 10,\ 'r')\,;\ s := s + r\, \mathbf{od}\,;\ \mathrm{RETURN}(s)\ \mathbf{end}$$

```
> somme_chiffrei(114505);
```

16

fonction de fibonacci

```
> fibo := proc(n:nonnegint)
> if (n=0 or n=1) then 1 else fibo(n-1)+fibo(n-2) fi;
> end;
```

$$fibo := \mathbf{proc}(n::integer)\, \mathbf{if}\, n = 0\ \mathbf{or}\ n = 1\, \mathbf{then}\, 1\, \mathbf{else}\, \mathrm{fibo}(n-1) + \mathrm{fibo}(n-2)\, \mathbf{fi}\ \mathbf{end}$$

```
> st := time(): fibo(28); `temps écoulé (en s)`= time()-st;
```

514229

$$temps\ ecoule(en\ s) = 50.000$$

Fonction de fibonacci avec l'option remember;

```
> fibobis := proc(n:nonnegint)
> option remember;
> if (n=0 or n=1) then 1 else fibobis(n-1)+fibobis(n-2) fi;
> end;
```

fibobis := **proc**(*n*::*integer*)

option*remember*;

if $n = 0$ **or** $n = 1$ **then** 1 **else** fibobis$(n-1)$ + fibobis$(n-2)$ **fi**

end

```
> st := time(): fibobis(28); ‘temps écoulé (en
  s)‘ = time()-st;
```

514229

temps ecoule (*en s*) = 0

Fonction de fibonacci itérative

```
> fiboi := proc(n : nonnegint)
> local a, b, c ,i;
> a := 1; b := 1;
> for i from 2 to n do
>          c := a + b; # calcul du nombre de fibonacci suivant
>          a := b; b := c;
>          od;
> RETURN(b)
> end;
```

fiboi := **proc**(*n*::*nonnegint*)

local*a*, *b*, *c*, *i*;

$a := 1$; $b := 1$; **for** i **from** 2 **to** n **do** $c := a + b$; $a := b$; $b := c$ **od** ; RETURN(b)

end

```
> st := time(): fiboi(28); ‘temps écoulé (en
  s)‘ = time()-st;
```

514229

temps ecoule (*en s*) = 0

La fonction g retourne la liste des couples (i, u_i) pour i variant de 1 à n.

```
> g := proc(lambda,n)
> local i, u, l, f;
> u := 0.5; l[0] := [0,0.5];
> for i from 1 to n do
>          u := 4*lambda*u*(1-u);
>          l[i] := [i,u];
>          od;
> RETURN([seq(l[i],i=0..n)])
> end;
```

$$g := \mathbf{proc}(\lambda,\ n)$$

$$\mathbf{local} i,\ u,\ l,\ f;$$

$$u := .5\,;$$

$$l_0 := [0,\ .5]\,;$$

$$\mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}\ u := 4 \times \lambda \times u \times (1-u)\,;\ l_i := [i,\ u]\ \mathbf{od}\,;$$

$$\mathrm{RETURN}([\mathrm{seq}(l_i,\ i = 0..n)])$$

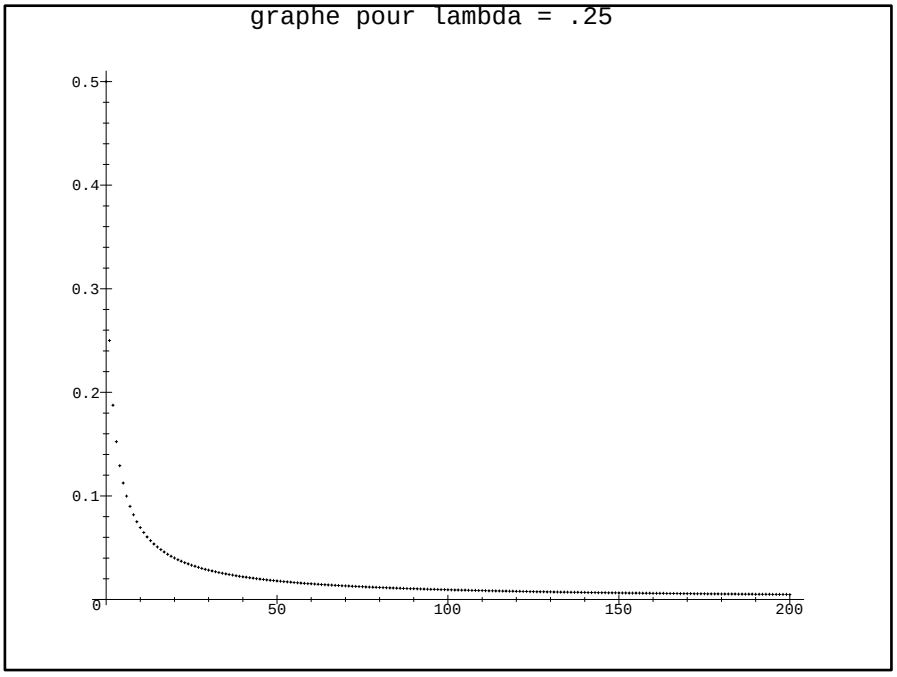
$$\mathbf{end}$$

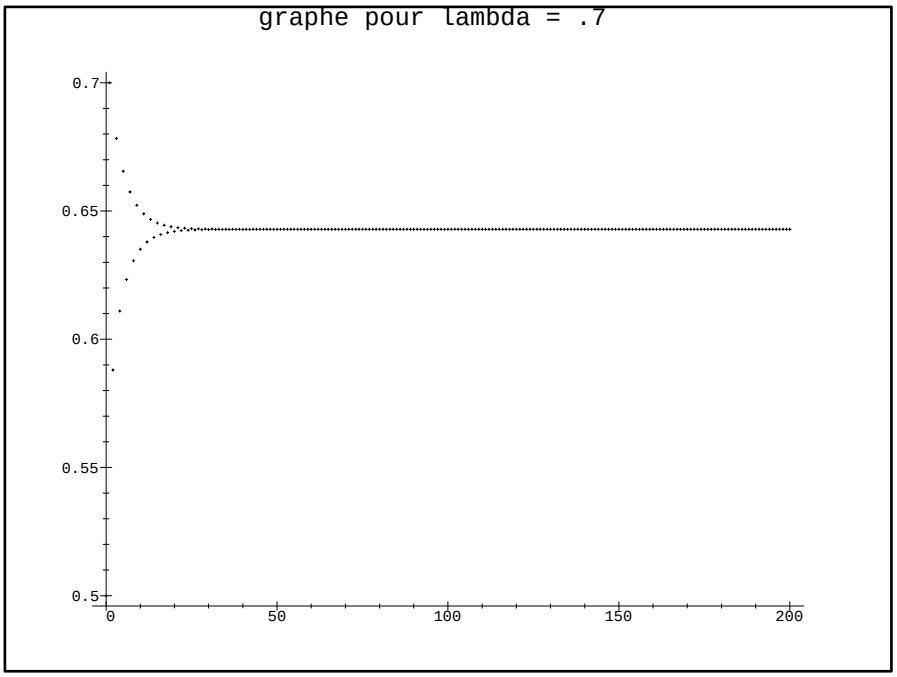
```
> g(0.7,10);
```

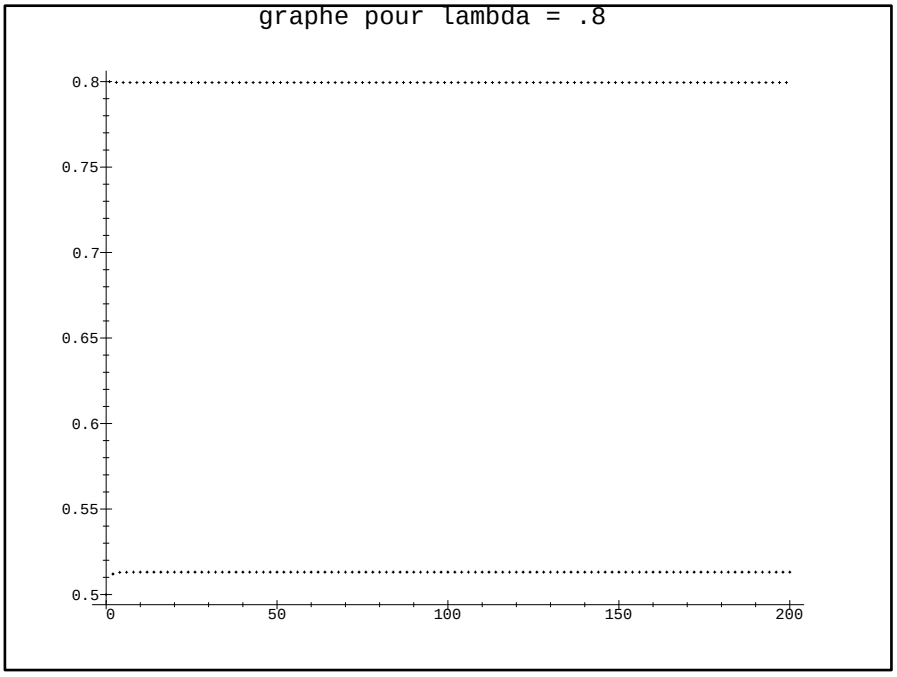
[[0, .5], [1, .700], [2, .5880000], [3, .6783168000], [4, .6109687328], [5, .6655206332], [6, .6232881760], [7, .6574400720], [8, .6305953464], [9, .6522455956], [10, .6350995800]]

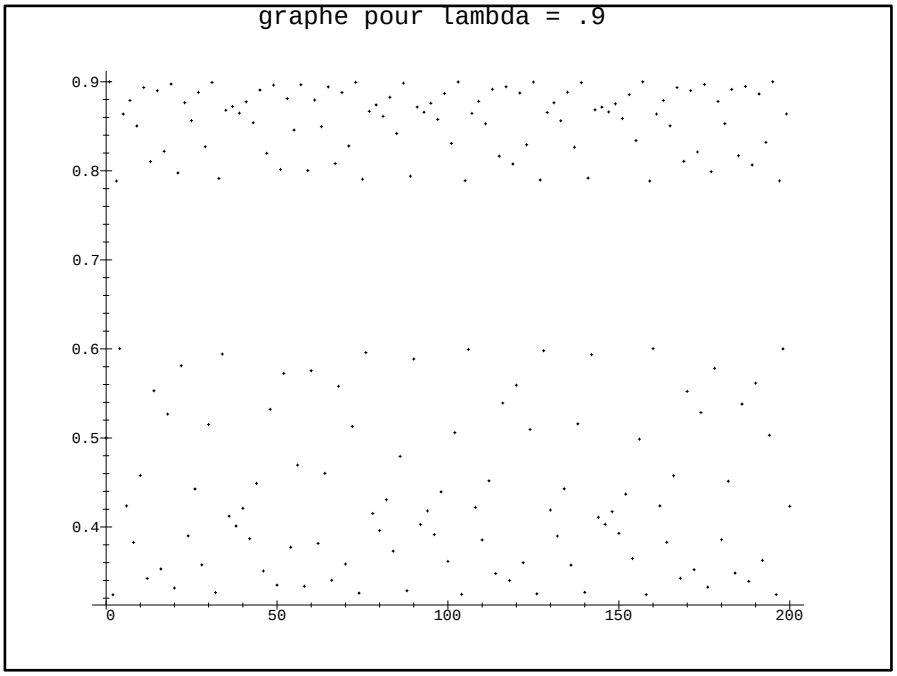
On fait une boucle pour afficher tous les graphes demandés.

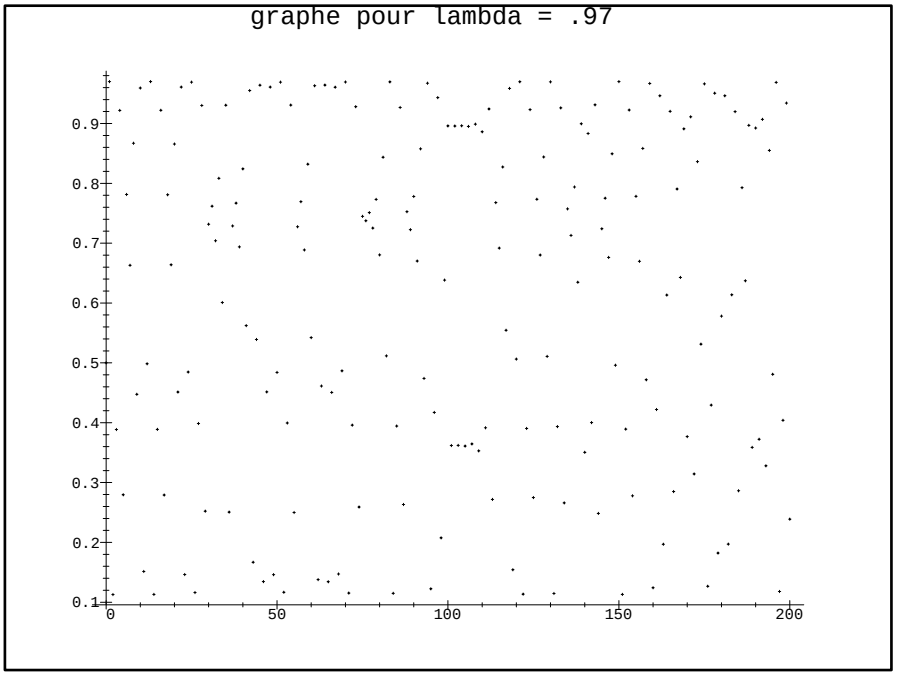
```
> for lambda in [0.25,0.7,0.8,0.9,0.97] do
> plots[pointplot](g(lambda,200),
> title=cat(‘graphe pour lambda = ‘, convert(lambda,string)))
> od;
```


graphe pour lambda = .25
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200


graphe pour lambda = .7
0.7
0.65
0.6
0.55
0.5
0
50
100
150
200


graphe pour lambda = .8
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
50
100
150
200


graphe pour lambda = .9
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200


graphe pour lambda = .97
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200