CORRECTION DE LA SERIE 1

```
>
```

Serie 1 - Exercice 2 -1

```
> restart;
> factorielle1:=proc(n)
  local i,r:
  if(n<0) then print(`Il faut saisir un entier positif`)
  else
     if n=0 then
        r:=1:
     else
        r:=1:
        for i from 2 to n by 1 do
           r:=r*i:
        od:
     fi:
     RETURN(r);
  fi:
  end:
> factorielle1 (0);
```

1

Serie 1 - Exercice 2 -2

```
> restart;
> factorielle2:=proc(n)
  if n<0 then print(`Il faut saisir un entier positif`)
  else
      if n=0 then RETURN(1) # cas d'arret
      else RETURN (n*factorielle2(n-1)) # on appelle la procédure
  avec l'entrée n-1
      fi:
  fi:
  end:

> factorielle2(6);
```

720

Serie 1 - Exercice 2 -3

```
> restart;
> exponentielle:=proc(x)
  local i,eps,term,ex:
  i:=1:
  eps:=10^(-10):
  term:=1:
  ex:=1:
  while(term>eps) do
     term:=(x^i)/i!:
     ex:=ex+term:
```

```
    i:=i+1:
od:
RETURN(evalf(ex)):
end:
> exponentielle(1);
```

2.718281828

```
> exponentielle(2);
```

7.389056099

Serie 1 - Exercice 2 -4

```
> restart;
> bino:=proc(x,n)
local i,comb,res:
res:=1:
for i from 1 to n by 1 do
    comb:=n!/(i!*(n-i)!):
    res:=res+comb*(x^i):
od:
RETURN(res):
end:
> bino(2,3);
```

27

Serie 1 - Exercice 2 -5

```
> restart;
> suite:=proc(n)
local i,U0,U1,U2,Ui:
U0:=1:
U1:=1:
U2:=U0/2:
if (n=0) then Ui:=1:
elif (n=1) then Ui:=1:
elif(n=2) then Ui:=U2:
else
    for i from 3 to n do
       if(is(i,odd)) then
           Ui:=U2+U0:
       else
           Ui:=U2/2:
           U0:=U2:
           U2:=Ui:
       fi:
    od:
fi:
RETURN (evalf(Ui,3)); #printf(`U%d = %f    `,n,Ui):
end:
> seq(suite(i),i=1..5);
```

1., .500, 1.50, .250, .750

Serie 1 - Exercice 3-1

```
> restart;
> Eq_SD:=proc(a,b,c)
  local delta:
  if ((a=0) and (b=0) and (c=0)) then print(`Tout reel est
  solution de l'equation`):
  elif((a=0) and (b=0) and (c<>0)) then print(`Pas solutions dans
  IR`):
  elif(a=0 and b<>0) then print(`la solution est
  `,x=evalf(-c/b,3)):
  else
      delta:=b^2-4*a*c:
      if(delta<0)then print(`Pas solutions dans IR`):
       elif(delta=0)then print(`la solution est
  `,x=evalf(-b/2/a,3)):
       else print(`les solutions sont
  `,x1=evalf((-b-sqrt(delta))/(2.0*a),3),
  x2=evalf((-b+sqrt(delta))/(2.0*a),3)):
       fi
  fi
  end:
> Eq_SD(0,0,0);
```

Tout reel est solution de l'equation

```
> Eq_SD(0,0,1);
```

Pas solutions dans IR

```
> Eq_SD(0,2,1);
```

la solution est , $x = -.500$

```
> Eq_SD(4,1,2);
```

Pas solutions dans IR

```
> Eq_SD(1,-2,1);
```

la solution est , $x = 1.$

```
> Eq_SD(3,6,1);
```

les solutions sont , $x1 = -1.82$, $x2 = -.182$

Serie 1 - Exercice 3.2a

```
> restart;
> with(linalg):
Warning, new definition for norm
Warning, new definition for trace
> somme_matricielle := proc(A,B)
  local m_som, i, j:
  m_som := evalm(A):
  for i from 1 to rowdim(A) do
      for j from 1 to coldim(A) do
          m_som[i,j] :=A[i,j]+B[i,j] :
      od:
```

```
   od:
   evalm(m_som):
   end;
```

somme_matricielle := **proc**(*A*, *B*)
local *m_som*, *i*, *j*;
    m_som := evalm(*A*);
    for *i* **to** rowdim(*A*) **do for** *j* **to** coldim(*A*) **do** *m_som*[*i*, *j*] := *A*[*i*, *j*] + *B*[*i*, *j*] **od od**;
    evalm(*m_som*)
end

```
> A:=array([[2,2],[3,4]]);
```

$$A := \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> B:=array([[1,1],[1,1]]);
```

$$B := \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
> somme_matricielle(A,B);
```

$$\begin{bmatrix} 3 & 3 \\ 4 & 5 \end{bmatrix}$$

<u>**Serie 1 - Exercice 3.2b**</u>

```
> restart;
> with(linalg):
Warning, new definition for norm
Warning, new definition for trace
> produit_matricielle := proc(A,B)
   local m_prod, i, j, k:
   m_prod := evalm(A):
   for i from 1 to rowdim(A) do
       for j from 1 to coldim(A) do
           m_prod[i,j] :=0:
           for k from 1 to coldim(A) do
               m_prod[i,j]:=m_prod[i,j] + A[i,k]*B[k,j] :
           od:
       od:
   od:
   evalm(m_prod):
   end;
```

produit_matricielle := **proc**(*A*, *B*)
local *m_prod*, *i*, *j*, *k*;
    m_prod := evalm(*A*);
    for *i* **to** rowdim(*A*) **do for** *j* **to** coldim(*A*) **do**
        m_prod[*i*, *j*] := 0;
        for *k* **to** coldim(*A*) **do** *m_prod*[*i*, *j*] := *m_prod*[*i*, *j*] + *A*[*i*, *k*]**B*[*k*, *j*] **od**
    od
    od;

evalm(*m_prod*)

end

```
> A:=array([[1,2],[3,4]]);
> B:=array([[1,1],[1,1]]);
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B := \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
> produit_matricielle(A,B);
```

$$\begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$$

Serie 1 - Exercice 3.3

```
> restart;with(linalg):
Warning, new definition for norm
Warning, new definition for trace
> op_mat:=proc(A,lambda)
     local res,id,i,j:
     res:=evalm(A):
     id:=array(identity,1..coldim(A),1..coldim(A)):
     for i from 1 to rowdim(A) do
      for j from 1 to coldim(A) do
          res[i,j] :=A[i,j]+lambda*id[i,j] :
      od:
     od:
     evalm(res):
     end;
```

op_mat := **proc**(*A*, λ)

local *res*, *id*, *i*, *j*;

res := evalm(*A*);

id := array(*identity*, 1 .. coldim(*A*), 1 .. coldim(*A*));

for *i* **to** rowdim(*A*) **do for** *j* **to** coldim(*A*) **do** *res*[*i*, *j*] := *A*[*i*, *j*] + λ**id*[*i*, *j*] **od od**;

evalm(*res*)

end

```
> A:=array([[1,2],[3,4]]);
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> op_mat(A,3);
```

$$\begin{bmatrix} 4 & 2 \\ 3 & 7 \end{bmatrix}$$

Serie 1 - Exercice 4.1

```
> restart;
> integral:=proc(f,a,b,n)
  local h,i,som:
  h:=(b-a)/n:
  som:=(f(a)+f(b))/2:
```

```
for i from 1 to n-1 do
   som:=som+f(a+i*h):
od:
som:=som*h;
RETURN (evalf(som)):
end;
```

integral := **proc**(*f*, *a*, *b*, *n*)
local *h*, *i*, *som*;
h := (*b* – *a*) / *n*;
som := 1 / 2∗*f*(*a*) + 1 / 2∗*f*(*b*);
for *i* **to** *n* – 1 **do** *som* := *som* + *f*(*a* + *i*∗*h*) **od**;
som := *som*∗*h*;
RETURN(evalf(*som*))
end

```
> f:=x->cos(x);
```

$$f := \cos$$

```
> integral(f,0,Pi/2,1000);
```

$$.9999997935$$

```
> Int(cos(x),x=0..Pi/2)=int(cos(x),x=0..Pi/2);
```

$$\int_0^{1/2\,\pi} \cos(x)\,dx = 1$$

```
> g:=x->1/x;
```

$$g := x \to \frac{1}{x}$$

```
> integral(g,1,3,1000);
```

$$1.098612585$$

```
> Int(1/t,t=1..3)=evalf(int(1/t,t=1..3));
```

$$\int_1^3 \frac{1}{t}\,dt = 1.098612289$$

<u>Serie 1 - Exercice 4.2</u>

```
> restart;
> Euler := proc(F,X0,Y0,Xfin,n)
    local XM, YM, XE, YE, h, i:
    global Points:
    XM := evalf(X0):
    YM := evalf(Y0):
    h := evalf((Xfin-X0)/n):
    Points := [[XM, YM]]:
    for i from 1 to n do
       YE := YM+h*F(XM, YM):
       XE := XM+h:
```

```
        Points := [op(Points), [XE, YE]]:
        XM := XE:
        YM := YE:
     od:
     RETURN (Points):
     #plot(Points);
  end;
```

Euler := **proc**(*F*, *X0*, *Y0*, *Xfin*, *n*)
local *XM*, *YM*, *XE*, *YE*, *h*, *i*;
global *Points*;
$XM := \mathrm{evalf}(X0)$;
$YM := \mathrm{evalf}(Y0)$;
$h := \mathrm{evalf}((Xfin - X0) / n)$;
$Points := [[XM, YM]]$;
for *i* **to** *n* **do**
$YE := YM + h*F(XM, YM)$;
$XE := XM + h$;
$Points := [\mathrm{op}(Points), [XE, YE]]$;
$XM := XE$;
$YM := YE$
od;
RETURN(*Points*)
end

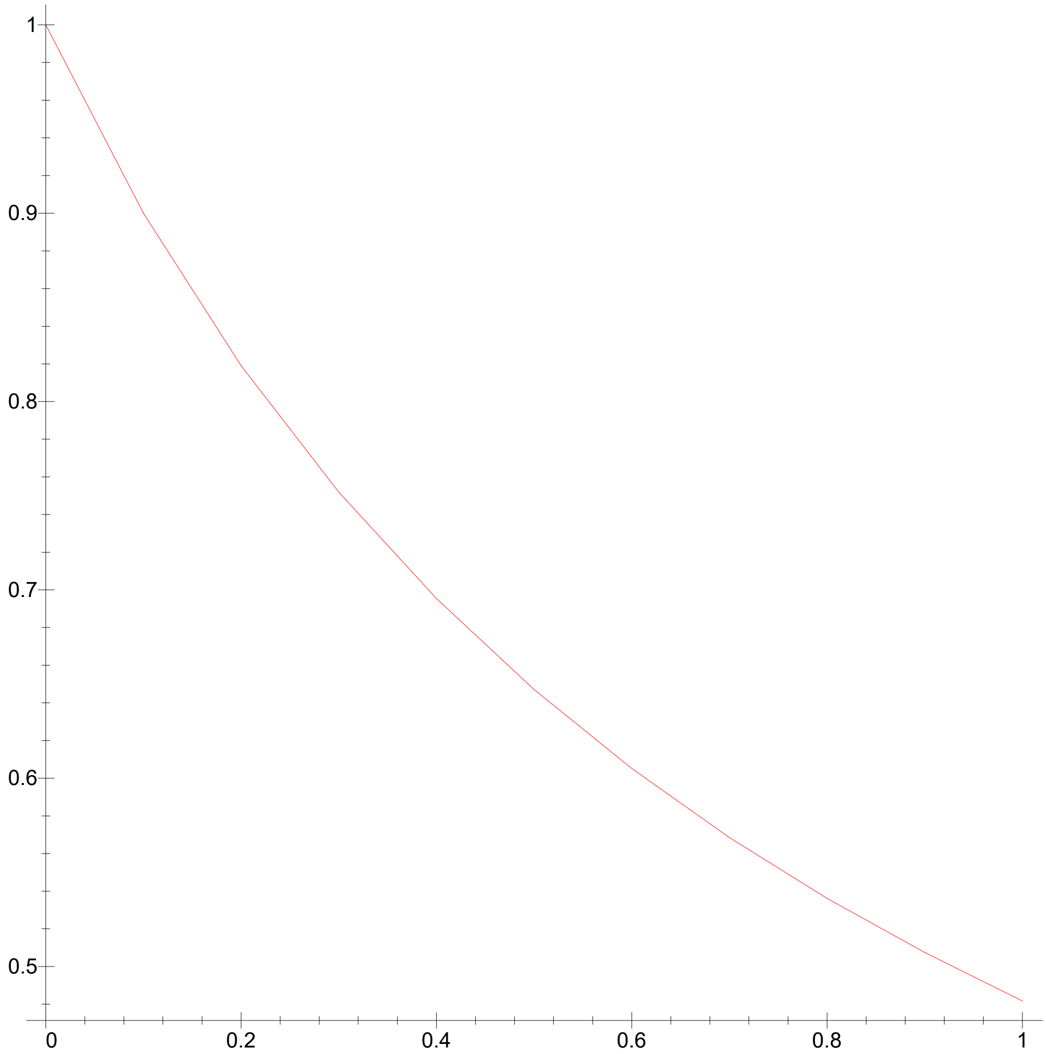
```
> F:=(x,y)->-y^2;
```

$$F := (x, y) \rightarrow -y^2$$

```
> Euler(F,0,1,1,10);
```

[[0, 1.], [.1000000000, .9000000000], [.2000000000, .8190000000],
[.3000000000, .7519239000], [.4000000000, .6953849449], [.5000000000, .6470289227],
[.6000000000, .6051642800], [.7000000000, .5685418994], [.8000000000, .5362179103],
[.9000000000, .5074649456], [1.000000000, .4817128785]]

```
> plot(Points);
```



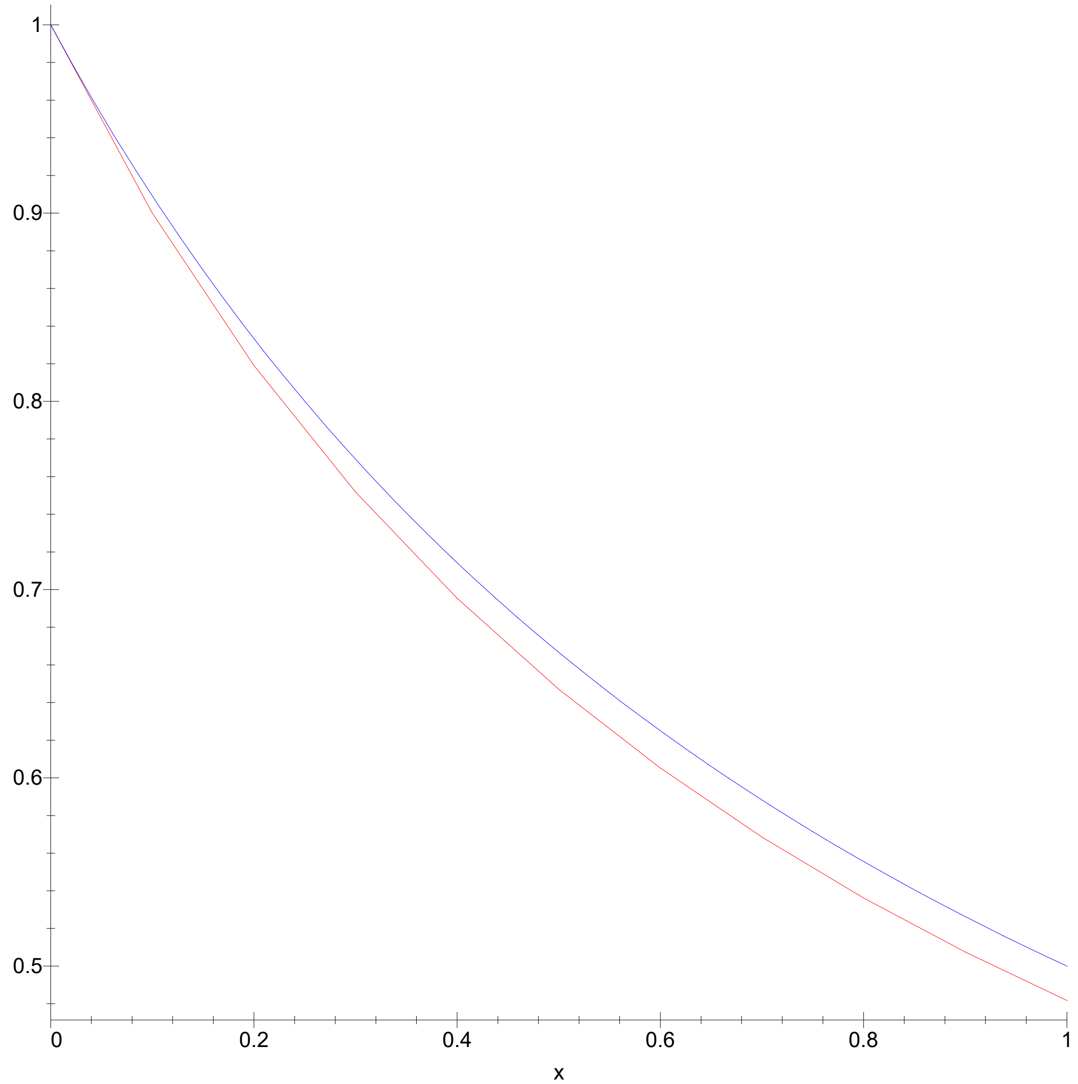

```
> eqd:=diff(y(x),x)=-y(x)^2;
```

$$eqd := \frac{\partial}{\partial x} y(x) = -y(x)^2$$

```
> dsolve({eqd,y(0)=1},y(x));
```

$$y(x) = \frac{1}{x+1}$$

```
> plot([Points,op(2,")],x=0..1,color=[red,blue]);
```


1
0.9
0.8
0.7
0.6
0.5
0
0.2
0.4
0.6
0.8
1
x

>

>

>