Julia pour les mathématiques: une introduction

Olivier Garet

Université de Lorraine – IECL

22 juin 2021

Plan

Positionnement de Julia

- Rapide, compilation Just In Time

Positionnement de Julia

- Rapide, compilation Just In Time
- Utilisable simplement en ligne de commande, en script, en IDE

Positionnement de Julia

- Rapide, compilation Just In Time
- Utilisable simplement en ligne de commande, en script, en IDE
- Syntaxe naturelle pour un.e mathématicien.ne

Positionnement de Julia

- Rapide, compilation Just In Time
- Utilisable simplement en ligne de commande, en script, en IDE
- Syntaxe naturelle pour un.e mathématicien.ne
  - usage matriciel (à la Scilab, Matlab)

Positionnement de Julia

- Rapide, compilation Just In Time
- Utilisable simplement en ligne de commande, en script, en IDE
- Syntaxe naturelle pour un.e mathématicien.ne
  - usage matriciel (à la Scilab, Matlab)
  - Les fonctions sont des objets ≪ comme les autres ≫

Positionnement de Julia

- Rapide, compilation Just In Time
- Utilisable simplement en ligne de commande, en script, en IDE
- Syntaxe naturelle pour un.e mathématicien.ne
  - usage matriciel (à la Scilab, Matlab)
  - Les fonctions sont des objets « comme les autres »
  - Facilité pour créer de nouveaux types

Positionnement de Julia

- Rapide, compilation Just In Time
- Utilisable simplement en ligne de commande, en script, en IDE
- Syntaxe naturelle pour un.e mathématicien.ne
  - usage matriciel (à la Scilab, Matlab)
  - Les fonctions sont des objets « comme les autres »
  - Facilité pour créer de nouveaux types
- Souplesse du typage ; POO sans larmes.

Bibliographie

- https://olivier.garet.xyz/livre_julia/
- George Root : The Julia Language Handbook
- https://learnxinyminutes.com/docs/julia/

Incontournable !

```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.0 (2021-03-24)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> using Plots

julia> plot(sin,0:0.01:pi)

julia> savefig("sinus.png")
```

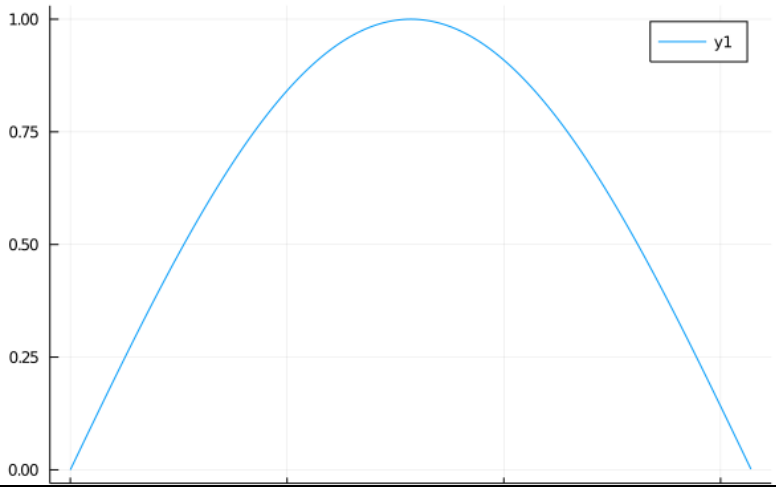
Incontournable !

```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.0 (2021-03-24)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> using Plots

julia> plot(sin,0:0.01:pi)

julia> savefig("sinus.png")
```

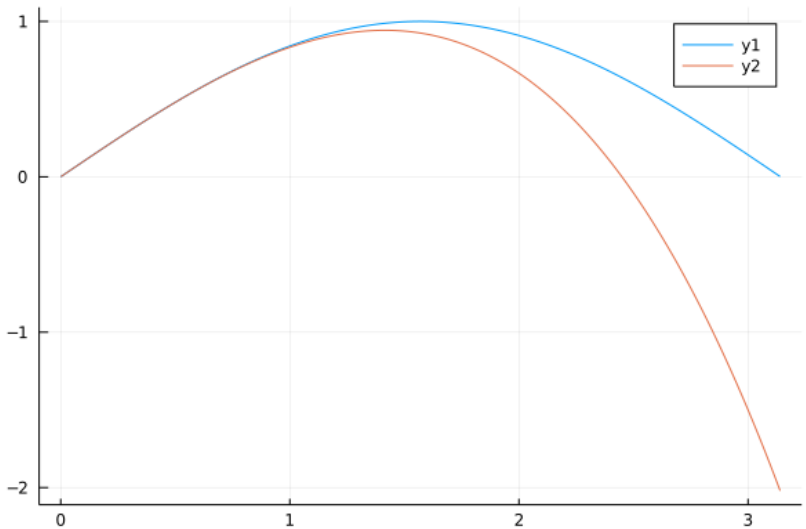


Incontournable !

```
julia> plot!(x->x-x^3/6,0:0.01:pi)
```

Incontournable !

```
julia> plot!(x->x-x^3/6,0:0.01:pi)
```



Un langage qui aime les fonctions

```
julia> g(x)=x^2
g (generic function with 1 method)
```

Un langage qui aime les fonctions

```
julia> g(x)=x^2
g (generic function with 1 method)
```

```
julia> g(5)
25
```

Un langage qui aime les fonctions

```
julia> g(x)=x^2
g (generic function with 1 method)
```

```
julia> g(5)
25
```

```
julia> delta(f)=(x->f(x+1)-f(x))
delta (generic function with 1 method)
```

Un langage qui aime les fonctions

```
julia> g(x)=x^2
g (generic function with 1 method)

julia> g(5)
25

julia> delta(f)=(x->f(x+1)-f(x))
delta (generic function with 1 method)

julia> h=delta(g)
#7 (generic function with 1 method)
```

Un langage qui aime les fonctions

```
julia> g(x)=x^2
g (generic function with 1 method)

julia> g(5)
25

julia> delta(f)=(x->f(x+1)-f(x))
delta (generic function with 1 method)

julia> h=delta(g)
#7 (generic function with 1 method)

julia> h(1)
3
```

Un langage qui aime les fonctions

```
julia> g(x)=x^2
g (generic function with 1 method)
```

```
julia> g(5)
25
```

```
julia> delta(f)=(x->f(x+1)-f(x))
delta (generic function with 1 method)
```

```
julia> h=delta(g)
#7 (generic function with 1 method)
```

```
julia> h(1)
3
```

```
julia> h.([1 10 100 1000])
1×4 Matrix{Int64}:
 3  21  201  2001
```

Itérateurs et tableaux

```
julia> iterateur=1:5
1:5
```

Itérateurs et tableaux

```
julia> iterateur=1:5
1:5
```

```
julia> collect(iterateur)
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

Itérateurs et tableaux

```
julia> iterateur=1:5
1:5
```

```
julia> collect(iterateur)
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

```
julia> iterateur=1:2:5
1:2:5
```

Itérateurs et tableaux

```
julia> iterateur=1:5
1:5
```

```
julia> collect(iterateur)
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

```
julia> iterateur=1:2:5
1:2:5
```

```
julia> collect(iterateur)
3-element Vector{Int64}:
 1
 3
 5
```

Un petit programme

```
function cofactor(A)
 (a,b)=axes(A)
 return([(-1)^(col + row) *
  det(A[filter(x->(x!=col),a),
        filter(x->(x!=row),b)])
        for col=a, row=b])
end
```

Un petit programme (variante)

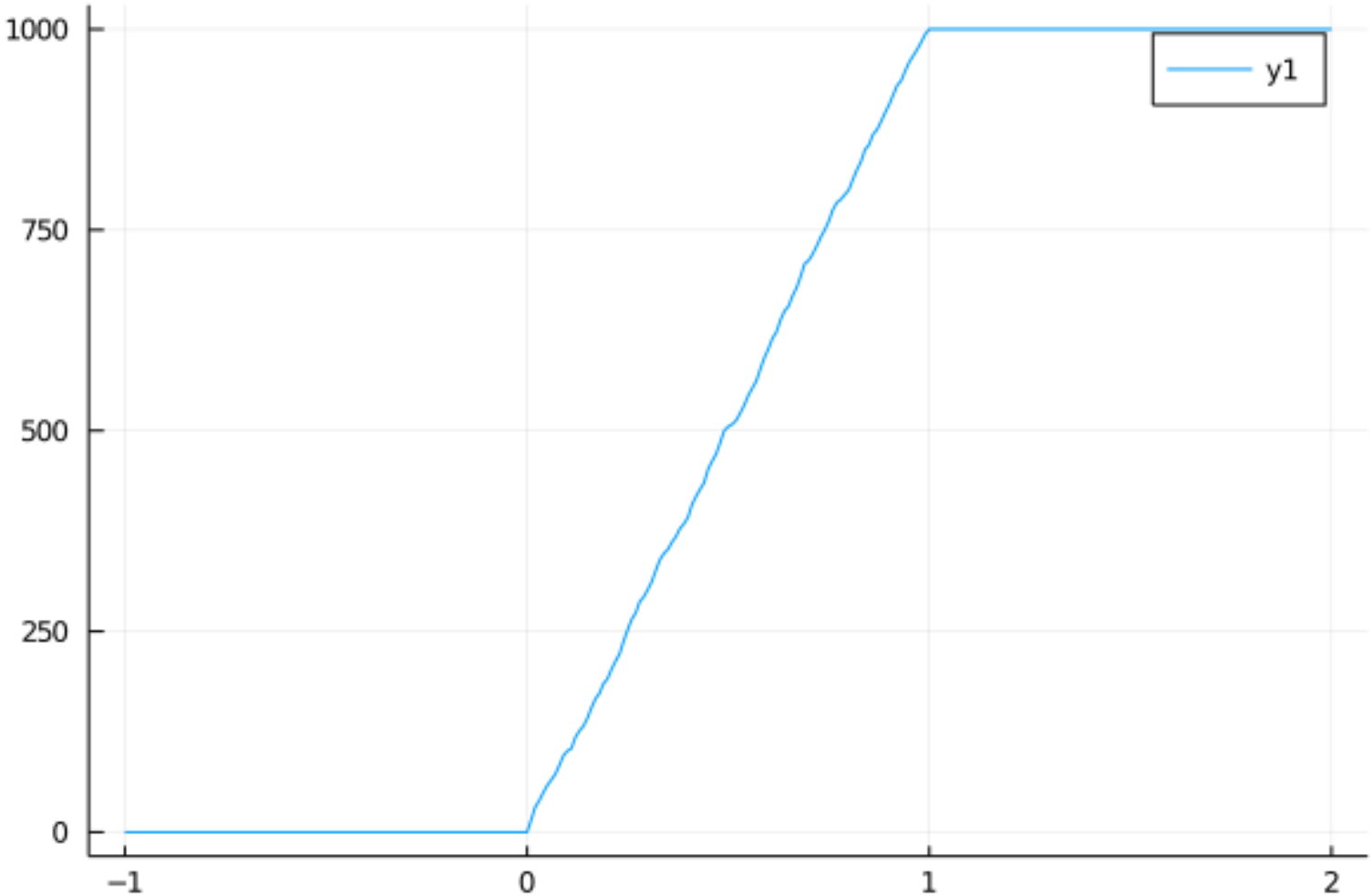
```
function tcofactor(A)
 (a,b)=axes(A)
 return([(-1)^(col + row) *
  det(A[filter(x->(x!=col),a),
        filter(x->(x!=row),b)])
        for row=b, col=a])
end
```

Probabilités, simulation à la main

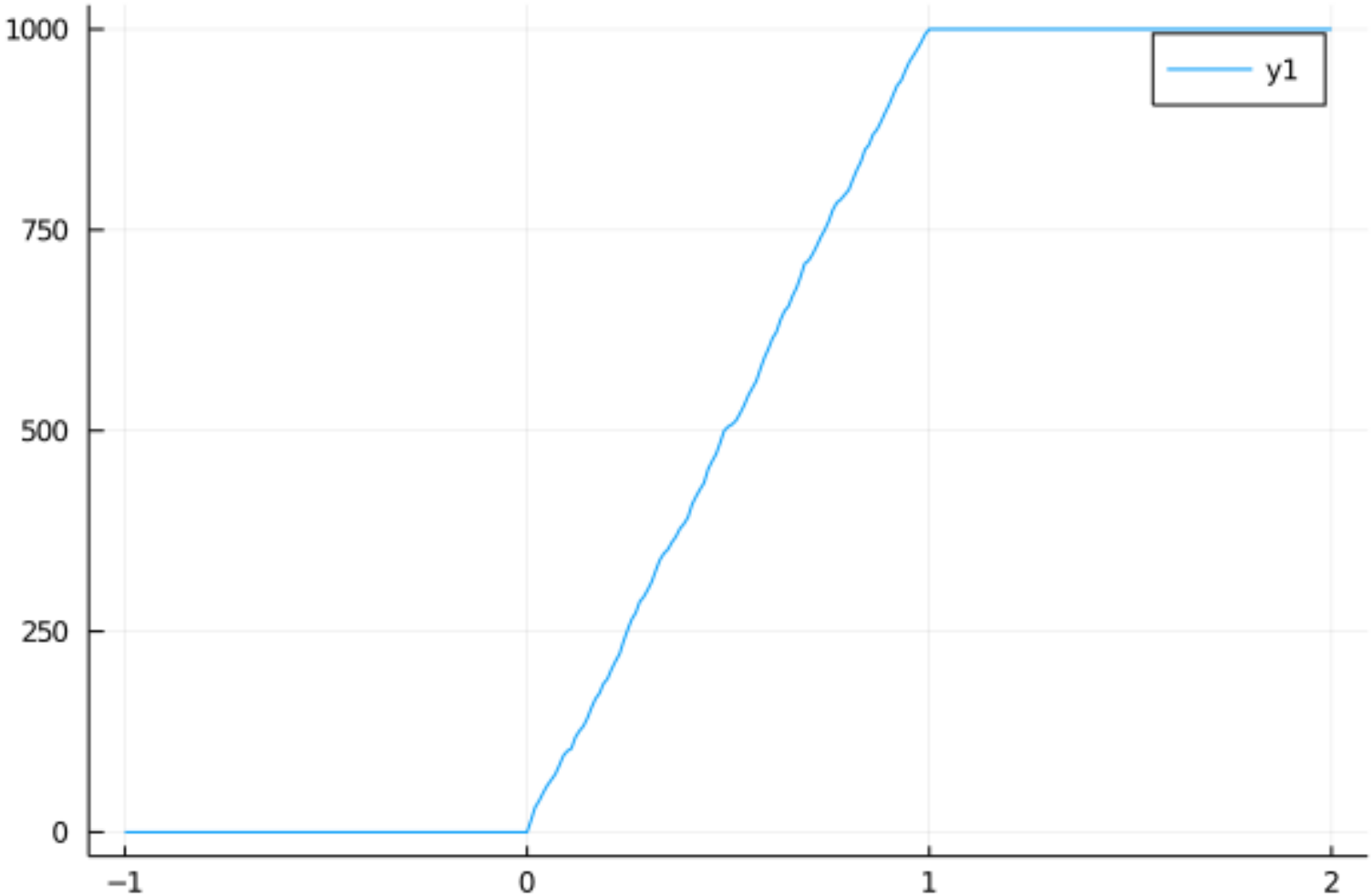
```
julia> u=rand(1000)
1000-element Array{Float64,1}:
 0.9714304962818339
 0.1032241601893073 6
 0.0474446686387286 5
 0.3248143304861721 4
 0.996049166254568 5
 0.944604358207946 2
  :
 0.2498187835548841 8
 0.6859241745874931
 0.590074318077292
 0.9534866112715394
 0.4498381373224380 5

julia> empirique(t,x)=sum(t.<x)
empirique (generic function with 1 method)
```

Probabilités (à la main)



Probabilités (à la main)



Statistiques (très élémentaires)

On peut faire des statistiques sur notre échantillon.

```
julia> using Statistics

julia> mean(u)
0.49557632029841153

julia> mean(u.^2)
0.3302776339365932
```

Statistiques (très élémentaires)

On peut faire des statistiques sur notre échantillon.

```
julia> using Statistics

julia> mean(u)
0.49557632029841153

julia> mean(u.^2)
0.3302776339365932
```

C'est cohérent avec le calcul des moments d'ordre 1 et 2.
Si $X \sim \mathcal{U}([0,1])$:

$$\mathbb{E}(X) = \int_0^1 x \, dx = \frac{1}{2}.$$

Statistiques (très élémentaires)

On peut faire des statistiques sur notre échantillon.

```
julia> using Statistics

julia> mean(u)
0.49557632029841153

julia> mean(u.^2)
0.3302776339365932
```

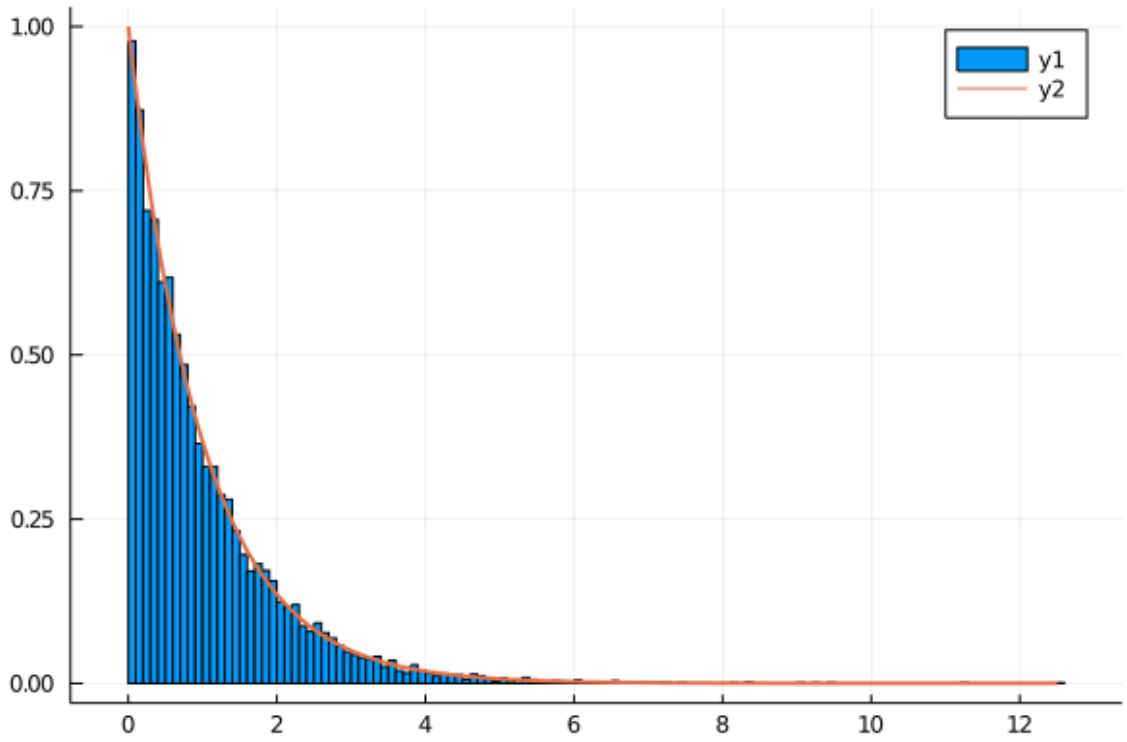
C'est cohérent avec le calcul des moments d'ordre 1 et 2.
Si $X \sim \mathcal{U}([0,1])$:

$$\mathbb{E}(X) = \int_0^1 x \, dx = \frac{1}{2}. \quad \mathbb{E}(X^2) = \int_0^1 x^2 \, dx = \frac{1}{3}.$$

De même, `var(u)` donne la variance empirique d'un échantillon. On peut aussi traiter des covariances, les quantiles...

Simulation avec des bibliothèques

```
using Distributions
E=Exponential(1)
t=rand(E,10000)
histogram(t,normalize=true)
plot!(0:0.1:maximum(t),x->pdf(E,x),linewidth=2)
```



Dispatch multiple

```
function puissance(a,b,n)
 return(a*b^n)
end
```

```
julia> puissance(3,0.1,3)
0.003000000000000001
```

Dispatch multiple

```
function puissance(a,b,n)
 return(a*b^n)
end
```

```
julia> puissance(3,0.1,3)
0.003000000000000001
julia> puissance(3,1//10,3)
3//1000
```

Dispatch multiple

```
function puissance(a,b,n)
 return(a*b^n)
end
```

```
julia> puissance(3,0.1,3)
0.003000000000000001
julia> puissance(3,1//10,3)
3//1000
julia> puissance("c","ab",3)
"cababab"
```

Dispatch multiple

```
function puissance(a,b,n)
 return(a*b^n)
end
```

```
julia> puissance(3,0.1,3)
0.003000000000000001
julia> puissance(3,1//10,3)
3//1000
julia> puissance("c","ab",3)
"cababab"
julia> puissance([1 2; 3 4],[1 2;4 -1],3)
2×2 Matrix{Int64}:
  81   0
 171  18
```

Un nouveau type : la transformation affine

```
mutable struct affinite
      a
      b
 end

import Base.*
*(x::affinite,y::affinite)=
affinite(y.a*x.a,y.a*x.b+y.b)

Base.inv(x::affinite)=
 affinite(inv(x.a),-x.b*inv(x.a))

 Base.one(x::affinite)
 =affinite(one(x.a),zero(x.a))
```

Un nouveau type : la transformation affine

```
function power(x,n)
    if (n==0) return(one(x))
    elseif (n==1) return(x)
    elseif (n==-1) return(inv(x))
    elseif (n>0)
     return(Base.power_by_squaring(x,n))
   else return(Base.power_by_squaring(inv(x),-n))
    end
end

import Base.^
(^)(x::affinite,n::Integer)=power(x,n)
```

Un nouveau type : la transformation affine

```
julia> f=affinite(2,1)
affinite(2, 1)
```

Un nouveau type : la transformation affine

```
julia> f=affinite(2,1)
affinite(2, 1)
```

```
julia> f^6
affinite(64, 63)
```

Un nouveau type : la transformation affine

```
julia> f=affinite(2,1)
affinite(2, 1)
```

```
julia> f^6
affinite(64, 63)
```

```
julia> f^(-6)
affinite(0.015625, -0.984375)
```

Un nouveau type : la transformation affine

```
julia> f=affinite(2,1)
affinite(2, 1)
```

```
julia> f^6
affinite(64, 63)
```

```
julia> f^(-6)
affinite(0.015625, -0.984375)
```

```
julia> affinite(2//1,1//1)^(-6)
affinite(1//64, -63//64)
```

Un nouveau type : la transformation affine

```
julia> f=affinite(2,1)
affinite(2, 1)
```

```
julia> f^6
affinite(64, 63)
```

```
julia> f^(-6)
affinite(0.015625, -0.984375)
```

```
julia> affinite(2//1,1//1)^(-6)
affinite(1//64, -63//64)
```

Et on réutilise notre code précédent

Un nouveau type : la transformation affine

```
julia> f=affinite(2,1)
affinite(2, 1)
```

```
julia> f^6
affinite(64, 63)
```

```
julia> f^(-6)
affinite(0.015625, -0.984375)
```

```
julia> affinite(2//1,1//1)^(-6)
affinite(1//64, -63//64)
```

Et on réutilise notre code précédent

```
julia> puissance(affinite(1//4,-1),f,2)
affinite(1//1, -1)
```

Motivations et outils

On veut s'épargner des calculs simples, de la vie de tous les jours du mathématicien ; les calculs de l'algèbre élémentaire.

- SymPy
- AbstractAlgebra/Nemo
- Symbols (jeune ; à surveiller)

Un problème de chaînes de Markov

A et B se confrontent dans un jeu, la probabilité que A gagne une manche est a, que B gagne une manche est b et qu'ils fassent manche nulle est $c = 1 - a - b$. Est déclaré gagnant celui qui gagne 2 manches consécutives pour la première fois. Calculer la probabilité que A gagne.

Un problème de chaînes de Markov

A et B se confrontent dans un jeu, la probabilité que A gagne une manche est a, que B gagne une manche est b et qu'ils fassent manche nulle est $c = 1 - a - b$. Est déclaré gagnant celui qui gagne 2 manches consécutives pour la première fois. Calculer la probabilité que A gagne.On trouve

$$\begin{pmatrix} \mathbb{P}^{\text{égalité}}(A \text{ gagne }) \\ \mathbb{P}^{\text{avantage A}}(A \text{ gagne }) \\ \mathbb{P}^{\text{avantage B}}(A \text{ gagne }) \end{pmatrix} = (I - M_1)^{-1} N_1.$$

avec

$$M_1 = \begin{pmatrix} 1-a-b & a & b \\ 1-a-b & 0 & b \\ 1-a-b & a & 0 \end{pmatrix} \text{ et } N_1 = \begin{pmatrix} 0 \\ a \\ 0 \end{pmatrix}.$$

Résolution avec SymPy

```
using SymPy,LinearAlgebra

@vars a b

M=[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println([simplify(a) for a in P],
    "/(",simplify(r),")")
```

Résolution avec SymPy

```
using SymPy,LinearAlgebra

@vars a b

M=[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println([simplify(a) for a in P],
    "/(",simplify(r),")")
```

```
Sym[a^2*(b + 1), a*(a*b + a + b^2), a^2]/(a^2*b + a^2 + a*b^2 + b^2)
```

Résolution avec SymPy

```
using SymPy,LinearAlgebra

@vars a b

M=[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println([simplify(a) for a in P],
    "/(",simplify(r),")")
```

```
Sym[a^2*(b + 1), a*(a*b + a + b^2), a^2]/(a^2*b + a^2 + a*b^2 + b^2)
```

Résultat final : $\mathbb{P}^{\text{égalité}}(A \text{ gagne }) = \frac{a^2(b+1)}{a^2(b+1)+b^2(a+1)}$.

Résolution avec Nemo

```
using Nemo

B,(a,b)=PolynomialRing(ZZ,["a","b"])

M=B[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println(P,"/(",r,")")
```

```
Any[a^2*b + a^2, a^2*b + a^2 + a*b^2, a^2]/(a^2*b + a^2 + a*b^2 + b^2
```

Résultat final : $\mathbb{P}^{\text{égalité}}(A \text{ gagne }) = \frac{a^2(b+1)}{a^2(b+1)+b^2(a+1)}$.

Résolution avec Nemo

```
using Nemo

B,(a,b)=PolynomialRing(ZZ,["a","b"])

M=B[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println(P,"/(",r,")")
```

Résolution avec Nemo

```
using Nemo

B,(a,b)=PolynomialRing(ZZ,["a","b"])

M=B[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println(P,"/(",r,")")
```

```
Any[a^2*b + a^2, a^2*b + a^2 + a*b^2, a^2]/(a^2*b + a^2 + a*b^2 + b^2)
```

Résolution avec Nemo

```
using Nemo

B,(a,b)=PolynomialRing(ZZ,["a","b"])

M=B[1-a-b a b ; 1-a-b 0 b;1-a-b a 0]
r=det(one(M)-M)
P=permutedims(cofactor(one(M)-M))*[0 ;a;0]
# ou P=tcofactor(one(M)-M)*[0 ;a;0]
println(P,"/(",r,")")
```

```
Any[a^2*b + a^2, a^2*b + a^2 + a*b^2, a^2]/(a^2*b + a^2 + a*b^2 + b^2
```

Résultat final : $\mathbb{P}^{\text{égalité}}(A \text{ gagne }) = \frac{a^2(b+1)}{a^2(b+1)+b^2(a+1)}$.

Et avec Sage ?

```
 SageMath version 9.0, Release Date: 2020-01-01
 Using Python 3.8.5. Type "help()" for help.
```

```
sage: a=var('a')
....: b=var('b')
....: M1=matrix([[1-a-b,a,b],[1-a-b,0,b],[1-a-b,a,0]])
sage: N1=matrix([[0],[a],[0]])
....: P=(1-M1)^(-1)*N1
....: print(factor(P))
....:
[       a^2*(b + 1)/(a^2*b + a*b^2 + a^2 + b^2)]
[(a*b + b^2 + a)*a/(a^2*b + a*b^2 + a^2 + b^2)]
[                  a^2/(a^2*b + a*b^2 + a^2 + b^2)]
sage:
```

Résultat final : $\mathbb{P}^{\text{égalité}}(A \text{ gagne }) = \frac{a^2(b+1)}{a^2(b+1)+b^2(a+1)}$.

Résolution d'une récurrence linéaire

```
using SymPy

u=symbols("u",cls=sympy.Function)
n=symbols("n")
f=u(n+2)-3//2*u(n+1)+1//2*u(n)-2^n

println(rsolve(f,u(n)))
bord=sympify("{u(0):0,u(1):1}")
println(rsolve(f,u(n),bord))
```

Résolution d'une récurrence linéaire

```
using SymPy

u=symbols("u",cls=sympy.Function)
n=symbols("n")
f=u(n+2)-3//2*u(n+1)+1//2*u(n)-2^n

println(rsolve(f,u(n)))
bord=sympify("{u(0):0,u(1):1}")
println(rsolve(f,u(n),bord))
```

Forme générale : `2*2^n/3 + C0 + 2^(-n)*C1`

Résolution d'une récurrence linéaire

```
using SymPy

u=symbols("u",cls=sympy.Function)
n=symbols("n")
f=u(n+2)-3//2*u(n+1)+1//2*u(n)-2^n

println(rsolve(f,u(n)))
bord=sympify("{u(0):0,u(1):1}")
println(rsolve(f,u(n),bord))
```

Forme générale : `2*2^n/3 + C0 + 2^(-n)*C1`

Avec équation au bord : `2*2^n/3 - 2*2^(-n)/3`