A package for combinatorial probability theory

Franz Lehner

Preliminary version, October 13, 2011

Abstract

We present a package for combinatorial manipulation of probability distributions, moments, cumulants, orthogonal polynomials and convolutions. The central object is the `FriCAS` domain `Distribution`, which represents probability distributions as streams of moments and cumulants.

Contents

1 A few words about Axiom/FriCAS

The axiom clones (Axiom/FriCAS/OpenAxiom) are the only CAS supporting streams, i.e., lists of indefinite length, which allow to compute with the potential infinite (in the sense of Aristotle), and delayed computation (things are computed when needed). All this is necessary to compute with infinite sequences like moments and cumulants.

`FriCAS` can do most things other systems can do

1a ⟨*example1.input* 1a⟩≡ 1b▷
```
p:= 1+x+x^2
```

$$x^2 + x + 1 \tag{1}$$

```
Type: Polynomial(Integer)
```

like differentiation

1b ⟨*example1.input* 1a⟩+≡ ◁1a 1c▷
```
D(p,x)
```

$$2\,x + 1 \tag{2}$$

```
Type: Polynomial(Integer)
```

and integration (it has the most complete implementation of the Risch algorithm):

1c ⟨*example1.input* 1a⟩+≡ ◁1b 2a▷
```
integrate(p,x)
```

$$\frac{1}{3}\,x^3+\frac{1}{2}\,x^2+x \tag{3}$$

Type: Polynomial(Fraction(Integer))

The main difference to other CAS is *strong typing*, i.e., every object has a type. This is very handy when it comes to computing with non-standard things like finite fields or noncommutative rings:

2a ⟨*example1.input* 1a⟩+≡ ◁1c 2b▷

```
FG:= FreeGroup Symbol
a:FG := 'a
b:FG:= 'b

a*b*a^-1 *b^-1
```

$$a\ b\ a^{(-1)}\ b^{(-1)} \tag{4}$$

Type: FreeGroup(Symbol)

Domains can be built on top of each other:

2b ⟨*example1.input* 1a⟩+≡ ◁2a 2c▷

```
FGA:=MonoidRing (Integer, FG)
a1:= a::FGA
a1i:= (a^-1)::FGA
b1:= b::FGA
b1i:= (b^-1)::FGA
T:=a1+a1i+b1+b1i
```

$$b^{(-1)}+a^{(-1)}+a+b \tag{5}$$

Type: MonoidRing(Integer,FreeGroup(Symbol))

Through the type system, FriCAS knows which multiplication it is meant to perform:

2c ⟨*example1.input* 1a⟩+≡ ◁2b 2d▷

```
T^2
```

$$b^2+b\ a+b\ a^{(-1)}+a\ b+a^2+a\ b^{(-1)}+a^{(-1)}\ b+a^{(-2)}+a^{(-1)}\ b^{(-1)}+4+b^{(-1)}\ a+b^{(-1)}\ a^{(-1)}+b^{(-2)} \tag{6}$$

Type: MonoidRing(Integer,FreeGroup(Symbol))

2d ⟨*example1.input* 1a⟩+≡ ◁2c 2e▷

```
A:Matrix FGA := matrix [[a1,a1i],[b1,b1i]]
```

$$\begin{bmatrix} a & a^{(-1)} \\ b & b^{(-1)} \end{bmatrix} \tag{7}$$

Type: Matrix(MonoidRing(Integer,FreeGroup(Symbol)))

2e ⟨*example1.input* 1a⟩+≡ ◁2d 2f▷

```
A*A
```

$$\begin{bmatrix} a^{(-1)}\ b+a^2 & a^{(-1)}\ b^{(-1)}+1 \\ 1+b\ a & b^{(-2)}+b\ a^{(-1)} \end{bmatrix} \tag{8}$$

Type: Matrix(MonoidRing(Integer,FreeGroup(Symbol)))

Types can be built on each other only if it makes sense:

2f ⟨*example1.input* 1a⟩+≡ ◁2e 2g▷

```
B:Matrix String := matrix [["a", "b"],["c", "d"]]

 Matrix(String) is not a valid type.
```

FriCAS allows infinite objects (potentially infinite in the sense of Aristoteles):

2g ⟨*example1.input* 1a⟩+≡ ◁2f 3a▷

```
nat:= [n for n in 1..]
```

$$[1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ ...] \tag{9}$$

Type: Stream(PositiveInteger)

3a ⟨*example1.input* 1a⟩+≡ ◁2g 3b▷

```
prim:=[n for n in nat | prime? n]
```

$$[2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19,\ 23,\ 29,\ ...] \tag{10}$$

Type: Stream(PositiveInteger)

Streams are used for the representation of power series, this way we don't have to worry about the number of terms we want to compute

3b ⟨*example1.input* 1a⟩+≡ ◁3a 3c▷

```
exps := series(exp x, x=0)
```

$$1+x+\frac{1}{2}\ x^2+\frac{1}{6}\ x^3+\frac{1}{24}\ x^4+\frac{1}{120}\ x^5+\frac{1}{720}\ x^6+\frac{1}{5040}\ x^7+\frac{1}{40320}\ x^8+\frac{1}{362880}\ x^9+\frac{1}{3628800}\ x^{10}+O\left(x^{11}\right) \tag{11}$$

Type: UnivariatePuiseuxSeries(Expression(Integer),x,0)

The number of terms computed by default is controlled with the command

3c ⟨*example1.input* 1a⟩+≡ ◁3b 3d▷

```
)set stream calculate 5
exps := series(exp x, x=0)
```

$$1+x+\frac{1}{2}\ x^2+\frac{1}{6}\ x^3+\frac{1}{24}\ x^4+\frac{1}{120}\ x^5+O\left(x^6\right) \tag{12}$$

Type: UnivariatePuiseuxSeries(Expression(Integer),x,0)

Yet we can compute as far as we want

3d ⟨*example1.input* 1a⟩+≡ ◁3c

```
coefficient(exps,50)
```

$$\frac{1}{3041409320171337804361260816606476884437764156896051200000000000} \tag{13}$$

Type: Expression(Integer)

2 Usage of the Distribution package

FriCAS can be extended by the user with her own data types. In our case, a probability distribution is represented by its sequence of moments

$$[m_1, m_2, \dots]$$

over a ring. With this package, various kinds of cumulants can be computed (classical, free and boolean), as well as classical, free additive and multiplicative, boolean and monotone convolutions and orthogonal polynomials. You can see all available commands by typing

```
)sh Distribution
```

which currently shows the following:

```
 Distribution(R: CommutativeRing)  is a domain constructor
 Abbreviation for Distribution is DISTRO
 This constructor is exposed in this frame.
------------------------------ Operations ------------------------------
 ?=? : (%,%) -> Boolean                0 : () -> %
 ?^? : (%,PositiveInteger) -> %        booleanConvolution : (%,%) -> %
 coerce : % -> OutputForm              freeConvolution : (%,%) -> %
 freeCumulants : % -> Sequence(R)      hash : % -> SingleInteger
 latex : % -> String                   moments : % -> Sequence(R)
 ?~=? : (%,%) -> Boolean
 booleanCumulant : (%,PositiveInteger) -> R
 booleanCumulantFromJacobi : (Integer,Sequence(R),Sequence(R)) -> R
 booleanCumulants : % -> Sequence(R)
 classicalConvolution : (%,%) -> %
 classicalCumulant : (%,PositiveInteger) -> R
 classicalCumulants : % -> Sequence(R)
 construct : (Sequence(R),Sequence(R),Sequence(R),Sequence(R)) -> %
 distributionByBooleanCumulants : Stream(R) -> %
 distributionByBooleanCumulants : Sequence(R) -> %
 distributionByClassicalCumulants : Stream(R) -> %
 distributionByClassicalCumulants : Sequence(R) -> %
 distributionByEvenMoments : Stream(R) -> %
 distributionByEvenMoments : Sequence(R) -> %
 distributionByFreeCumulants : Stream(R) -> %
 distributionByFreeCumulants : Sequence(R) -> %
 distributionByJacobiParameters : (Stream(R),Stream(R)) -> %
 distributionByJacobiParameters : (Sequence(R),Sequence(R)) -> %
 distributionByMoments : Stream(R) -> %
 distributionByMoments : Sequence(R) -> %
 distributionBySTransform : (Fraction(Integer),Fraction(Integer),Sequence(R)) -> % if R has ALGEB
 distributionBySTransform : Record(puiseux: Fraction(Integer),laurent: Fraction(Integer),coef: Se
 freeCumulant : (%,PositiveInteger) -> R
 freeMultiplicativeConvolution : (%,%) -> % if R has ALGEBRA(FRAC(INT))
 hankelDeterminants : % -> Stream(R)
 jacobiParameters : % -> Record(an: Stream(Fraction(R)),bn: Stream(Fraction(R))) if R has INTDOM
 jacobiParameters : % -> Record(an: Stream(R),bn: Stream(R)) if R has FIELD
```

```
moment : (%,NonNegativeInteger) -> R
monotoneConvolution : (%,%) -> %
monotoneCumulants : % -> Sequence(R) if R has ALGEBRA(FRAC(INT))
orthogonalPolynomials : % -> Stream(SparseUnivariatePolynomial(Fraction(R))) if R has INTDOM and
orthogonalPolynomials : % -> Stream(SparseUnivariatePolynomial(R)) if R has FIELD
```

Other commands are in different packages

```
)sh STransformPackage
)sh DistributionPolynomialPackage
)sh DistributionContinuedFractionPackage
)sh DISTEX DistributionPackage
```

All parameters are modeled as `Streams`, which has the following advantages:

- Calculation of moments and cumulants is delayed until demand
- once calculated, they are cached.
- the user does not have to worry about a maximal order of moments beforehand, the system computes as far as time and RAM restrictions of the underlying LISP and hardware allow it.

Once an interesting sequence is found, the `GUESS` package by M. Rubey (`arXiv:math/0702086`, also part of `FriCAS`) can be used to guess a recurrence or closed formula.

2.1 Example: Creating distributions via cumulant and moment sequences

Say we want to check whether the normal distribution is infinite divisible with respect to free convolution. There are several ways to construct the normal distribution.

1. The quickest way to get the normal distribution is via its classical cumulants, because only the second cumulant is nonzero. We need a declaration of an integer stream, because by default a nonnegative stream is generated and nonnegative integers do not form a ring.

5a ⟨*example2.input* 5a⟩≡ 5b▷

```
gausscum:Stream Integer := concat([0,1], repeating [0])
```

$$\left[0,\ 1,\ \overline{0}\right] \tag{14}$$

Type: Stream(Integer)

5b ⟨*example2.input* 5a⟩+≡ ◁5a 5c▷

```
gauss:=distributionByClassicalCumulants gausscum
```

$$[0,\ 1,\ 0,\ 3,\ 0,\ 15,\ 0,\ 105,\ 0,\ 945,\ ...] \tag{15}$$

Type: Distribution(Integer)

2. By the explicit formula for its moments $m_{2k} = \frac{2k!}{2^k k!}$:

5c ⟨*example2.input* 5a⟩+≡ ◁5b 6a▷

```
gaussmom k ==
   odd? k => return 0
   k2:= exquo(k,2)
   return (factorial(k)/(2^k2*factorial k2))::Integer

gauss:=distributionByMoments [gaussmom k for k in 1..]
```

$$[0,\ 1,\ 0,\ 3,\ 0,\ 15,\ 0,\ 105,\ 0,\ 945,\ ...] \tag{16}$$

Type: Distribution(Integer)

3. By dissecting the Taylor series of the exponential moment generating function $e^{z^2/2}$:

6a ⟨*example2.input* 5a⟩+≡ ◁5c 6b▷

```
gaussexp:= series(exp(z^2/2), z=0)
```

$$1+\frac{1}{2}\ z^2+\frac{1}{8}\ z^4+\frac{1}{48}\ z^6+\frac{1}{384}\ z^8+\frac{1}{3840}\ z^{10}+O\left(z^{11}\right) \tag{17}$$

Type: UnivariatePuiseuxSeries(Expression(Integer),z,0)

Next we must extract the coefficients of this series

6b ⟨*example2.input* 5a⟩+≡ ◁6a 6c▷

```
coefficients gaussexp
```

$$\left[1,\ 0,\ \frac{1}{2},\ 0,\ \frac{1}{8},\ 0,\ \frac{1}{48},\ 0,\ \frac{1}{384},\ 0,\ ...\right] \tag{18}$$

Type: Stream(Expression(Integer))

and multiply term by term with $n!$:

6c ⟨*example2.input* 5a⟩+≡ ◁6b 6d▷

```
gaussmom := [c*factorial n  for n in 1.. for c in coefficients gaussexp]
```

$$[1,\ 0,\ 3,\ 0,\ 15,\ 0,\ 105,\ 0,\ 945,\ 0,\ ...] \tag{19}$$

Type: Stream(Expression(Integer))

4. As moments of the position operator in the Weyl algebra of differential operators, which is already implemented in FriCAS.

6d ⟨*example2.input* 5a⟩+≡ ◁6c 6e▷

```
R:= UP(z,Integer)
W:=LODO1 R
```

$$LinearOrdinaryDifferentialOperator1(Fraction(UnivariatePolynomial(z, Integer))) \tag{20}$$

Now z and d verify the canonical commutation relation:

6e ⟨*example2.input* 5a⟩+≡ ◁6d 6f▷

```
z:R:='z
d:W:= D()
d*z-z*d
```

$$1 \tag{21}$$

Type: LinearOrdinaryDifferentialOperator1(Fraction(UnivariatePolynomial(z,Integer)))

we can compute the moments in this algebra by extracting the constant coefficient:

6f ⟨*example2.input* 5a⟩+≡ ◁6e 6g▷

```
g6:=(d+z)^6
```

(10)
$D^6+6\ z\ D^5+\left(15\ z^2+15\right)\ D^4+\left(20\ z^3+60\ z\right)\ D^3+\left(15\ z^4+90\ z^2+45\right)\ D^2+\left(6\ z^5+60\ z^3+90\ z\right)\ D+z^6+15$

Type: LinearOrdinaryDifferentialOperator1(UnivariatePolynomial(z,Integer))

The Weyl algebra is generated by D with coefficient ring $\mathbf{C}[z]$:

6g ⟨*example2.input* 5a⟩+≡ ◁6f 7a▷

```
coefficient(g6, 0)
```

(12) $$z^6 + 15\ z^4 + 45\ z^2 + 15$$

```
Type:  UnivariatePolynomial(z,Integer)
```

Therefore we must extract the constant coefficient of this polynomial:

7a ⟨*example2.input* 5a⟩+≡ ◁6g 7b▷

```
coefficient(coefficient(g6, 0), 0)
```

and finally we obtain the gaussian moments

7b ⟨*example2.input* 5a⟩+≡ ◁7a 7c▷

```
g:= distributionByMoments([coefficient(coefficient((d+z)^k, 0), 0) for k in 1..])
```

(13) [0, 1, 0, 3, 0, 15, 0, 105, 0, 945, ...]

```
Type:  Distribution(Integer)
```

5. By its jacobi parameters:

7c ⟨*example2.input* 5a⟩+≡ ◁7b 7d▷

```
an:Stream Integer := repeating [0]
```

$$[\overline{0}] \tag{22}$$

```
Type:  Stream(Integer)
```

7d ⟨*example2.input* 5a⟩+≡ ◁7c 7e▷

```
bn:Stream Integer := [k for k in 1..]
```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...] (23)

```
Type:  Stream(Integer)
```

7e ⟨*example2.input* 5a⟩+≡ ◁7d 7f▷

```
gauss:=distributionByJacobiParameters(an, bn)
```

[0, 1, 0, 3, 0, 15, 0, 105, 0, 945, ...] (24)

```
Type:  Distribution(Integer)
```

6. In the case of the normal distribution we can also use the built-in function

7f ⟨*example2.input* 5a⟩+≡ ◁7e 7h▷

```
gauss:=gaussianDistribution 1
```

[0, 1, 0, 3, 0, 15, 0, 105, 0, 945, ...] (25)

```
Type:  Distribution(Integer)
```

Infinite divisibility means that the sequence of free cumulants,

7g ⟨*example.input* 7g⟩≡

```
freeCumulants gauss
```

[0, 1, 0, 1, 0, 4, 0, 27, 0, 248, ...] (26)

```
Type:  Sequence(Integer)
```

starting with the third element, is positive definite and thus forms the moment sequence of the Levy measure:

7h ⟨*example2.input* 5a⟩+≡ ◁7f 8a▷

```
levymom:= rest rest freeCumulants gauss;
levy:= distributionByMoments levymom
```

$$[0,\ 1,\ 0,\ 4,\ 0,\ 27,\ 0,\ 248,\ 0,\ 2830,\ ...] \tag{27}$$

Type: Distribution(Integer)

Now we need to check whether the Hankel determinants are nonnegative:

8a ⟨*example2.input* 5a⟩+≡ ◁7h 8b▷

```
hankelDeterminants(levy)
```

$$[1,\ 3,\ 33,\ 1837,\ 586337,\ 1319746279,\ 23073023457505,\ 3650976798436751385,\ 5654872627743587044800033,\ 9678677 \tag{28}$$

Type: Stream(Integer)

Equivalently, we can check the positivity of the Jacobi parameters:

8b ⟨*example2.input* 5a⟩+≡ ◁8a

```
jacobiParameters levy
```

$$[an = [0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ ...],$$

$$bn = \left[1,\ 3,\ \frac{11}{3},\ \frac{167}{33},\ \frac{10533}{1837},\ \frac{4134779}{586337},\ \frac{10250885015}{1319746279},\ \frac{208831020924783}{23073023457505},\ \frac{11912337984453599843}{1216992266145583795},\ \frac{694320125446772}{62831918086039}\right.$$

Type: Record(an: Stream(Fraction(Integer)),bn: Stream(Fraction(Integer)))

2.2 Example: Convolutions

Computing convolutions is easy, using the commands `classicalConvolution`, `freeConvolution`, `freeMultiplicativeConvolution`,...

We investigate the behaviour of Jacobi Parameters under free convolution with a Wigner distribution.

8c ⟨*example3.input* 8c⟩≡ 8d▷

```
w:=wignerDistribution t
```

$$\left[0,\ t,\ 0,\ 2\ t^2,\ 0,\ 5\ t^3,\ 0,\ 14\ t^4,\ 0,\ 42\ t^5,\ ...\right] \tag{30}$$

Type: Distribution(Polynomial(Integer))

8d ⟨*example3.input* 8c⟩+≡ ◁8c 8e▷

```
x:=distributionByJacobiParameters([a[k] for k in 0..], [b[k] for k in 0..])
```

$$\left[a_0,\ b_0 + {a_0}^2,\ (a_1 + 2\ a_0)\ b_0 + {a_0}^3,\ ...\right] \tag{31}$$

Type: Distribution(Polynomial(Integer))

8e ⟨*example3.input* 8c⟩+≡ ◁8d 8f▷

```
xw:=freeConvolution(x,w)
```

$$\left[a_0,\ t + b_0 + {a_0}^2,\ 3\ a_0\ t + (a_1 + 2\ a_0)\ b_0 + {a_0}^3,\ ...\right] \tag{32}$$

Type: Distribution(Polynomial(Integer))

8f ⟨*example3.input* 8c⟩+≡ ◁8e ??▷

```
jacobiParameters xw
```

$$an = \left[a_0,\ \frac{a_0\ t + a_1\ b_0}{t + b_0},\ \frac{a_0\ t^4 + (a_1 + 3\ a_0)\ b_0\ t^3 + \left((a_2 + 2\ a_1 - 2\ a_0)\ b_0\ b_1 + 5\ a_0\ {b_0}^2 + \left({a_1}^3 - 2\ a_0\ {a_1}^2 + {a_0}^2\ a_1\right)\right.}{t^4 + 4\ b_0\ t^3 + \left(b_0\ b_1 + 5\ {b_0}^2 + ({a_1}^2 - 2\ a_0\ a_1 + {a_0}^2)\ b\right.}\right.$$

$$bn = \left[t + b_0,\ \frac{t^3 + 3\ b_0\ t^2 + \left(b_0\ b_1 + 2\ {b_0}^2 + \left({a_1}^2 - 2\ a_0\ a_1 + {a_0}^2\right)\ b_0\right)\ t + {b_0}^2\ b_1}{t^2 + 2\ b_0\ t + {b_0}^2},\ \frac{t^7 + 7\ b_0\ t^6 + \left(3\ b_0\ b_1 + 18\ {b_0}^2 + \right.}{}\right.$$

(33)

Type: Record(an: Stream(Fraction(Polynomial(Integer))),bn: Stream(Fraction(Polynomial(Integer))))

2.3 Example: Orthogonal polynomials

We verify that up to a rescaling the orthogonal polynomials of the Wigner distribution are indeed the Chebyshev polynomials, which can be obtained with the command `chebyshevU`:

9a ⟨*example4.input* 9a⟩≡ 9b▷

```
ch2:=orthogonalPolynomials(w)
```

$$\left[?,\ ?^2-1,\ ?^3-2\ ?,\ ?^4-3\ ?^2+1,\ ?^5-4\ ?^3+3\ ?,\ ...\right] \tag{34}$$

```
                                   Type:  Stream(SparseUnivariatePolynomial(Fraction(Integer)))
```

To give the unknown variable a name, convert it to the polynomial domain $\mathbf{Z}[t]$:

9b ⟨*example4.input* 9a⟩+≡ ◁9a 9c▷

```
ch2::Stream UP(t,Integer)
```

$$\left[t,\ t^2-1,\ t^3-2\ t,\ t^4-3\ t^2+1,\ t^5-4\ t^3+3\ t,\ ...\right] \tag{35}$$

```
                                   Type:  Stream(UnivariatePolynomial(t,Integer))
```

9c ⟨*example4.input* 9a⟩+≡ ◁9b 9d▷

```
chu:=[chebyshevU(k,t/2) for k in 1..]
```

$$\left[t,\ t^2-1,\ t^3-2\ t,\ t^4-3\ t^2+1,\ t^5-4\ t^3+3\ t,\ ...\right] \tag{36}$$

```
                                   Type:  Stream(Polynomial(Fraction(Integer)))
```

Now we compare the first ten polynomials:

9d ⟨*example4.input* 9a⟩+≡ ◁9c

```
ch2a:=ch2::Stream UP(t,Integer)
chua:=chu::Stream UP(t,Integer)
(entries complete first(ch2a, 10) = entries complete first(chua, 10))::Boolean
```

$$true \tag{37}$$

```
                                   Type:  Boolean
```