Julia, l'unique solution d'un problème d'optimisation.

ou *de la difficulté d'estimer des convolutions de lois gammas.*

O. Laverny [1,2]

June 22, 2021

[1] Institut Camille Jordan, UMR 5208, Université Claude Bernard Lyon 1, Lyon, France

[2] SCOR SE

Table of contents

Un problème d’optimisation mathématiquement ’simple’

Multivariate Gamma Convolutions

Recall: A random vector $\boldsymbol{X} \in \mathbb{R}_+^d$ can be characterized through it's cumulant generating function:

$$K(\boldsymbol{t}) = \ln \mathbb{E}\left(e^{\langle \boldsymbol{t}, \boldsymbol{X} \rangle}\right)$$

Definition (Multivariate Gamma Convolutions, see Bondesson[1])

$\boldsymbol{X} \sim \mathcal{G}_{d,n}(\alpha, \boldsymbol{s}) \Leftrightarrow K(\boldsymbol{t}) = \sum_{i=1}^{n} -\alpha_i \ln\left(1 - \langle \boldsymbol{s}_i, \boldsymbol{t} \rangle\right)$

Goal: Estimation form multivariate dataset, $d \leq\approx 4$ **and** $n \leq\approx 100$

Problem: No estimation procedures are available in the literature. Indeed, this is a hard inverse problem: to estimate some distribution in $\mathcal{G}_{d,n}$ for the random vector $\boldsymbol{X}$, we seek parameters of distributions of independent gamma random variables $Y_1, ..., Y_n$ such that $\boldsymbol{X} = \boldsymbol{s}'\boldsymbol{Y}$, that is:

$$X_1 = s_{1,1}Y_1 + \ldots + s_{1,n}Y_n$$

$$...$$

$$X_d = s_{d,1}Y_1 + \ldots + s_{d,n}Y_n$$

[1] Lennart Bondesson. *Generalized Gamma Convolutions and Related Classes of Distributions and Densities.* en. Ed. by J. Berger, S. Fienberg, J. Gani, K. Krickeberg, I. Olkin, and B. Singer. Vol. 76. Lecture Notes in Statistics. New York, NY: Springer New York, 1992. ISBN: 978-0-387-97866-6 978-1-4612-2948-3.

The orthonormal Laguerre basis of $L_2(\mathbb{R}_+^d)$

Definition (Tensorized Laguerre basis, see Comte[2], Mabon[3] and Dussap[4])

For all $\boldsymbol{p} \in \mathbb{N}^d$, $\varphi_{\boldsymbol{p}}(\boldsymbol{x}) = \prod_{i=1}^{d} \varphi_{p_i}(x_i)$ where $\varphi_p(x) = \sqrt{2} \sum_{k=0}^{p} \binom{p}{k} \frac{(-2x)^k}{k!} e^{-x}$.

These functions from an orthonormal basis of $L_2(\mathbb{R}_+^d)$.

Therefore, every density f that is square-integrable can be expended as :

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{p} \in \mathbb{N}^d} a_{\boldsymbol{p}} \varphi_{\boldsymbol{p}}(\boldsymbol{x}) \text{ where } a_{\boldsymbol{p}} = \int \varphi_{\boldsymbol{p}}(\boldsymbol{x}) f(\boldsymbol{x}) \partial \boldsymbol{x} = \sqrt{2}^d \sum_{\boldsymbol{k} \leq \boldsymbol{p}} \binom{\boldsymbol{p}}{\boldsymbol{k}} \frac{(-2)^{|\boldsymbol{k}|}}{\boldsymbol{k}!} \mathbb{E}\left(\boldsymbol{X}^{\boldsymbol{k}} e^{\langle -\mathbf{1}, \boldsymbol{X} \rangle}\right)$$

Final loss: $L(\alpha, \boldsymbol{s}) = \|f - \hat{f}\|_2^2 = \sum_{\boldsymbol{k} \leq \boldsymbol{m}} \left(\hat{a}_{\boldsymbol{k}} - a_{\boldsymbol{k}}(\alpha, \boldsymbol{s})\right)^2$

[2] Fabienne Comte and Valentine Genon-Catalot. "Adaptive Laguerre Density Estimation for Mixed Poisson Models". en. In: *Electronic Journal of Statistics* 9.1 (2015), pp. 1113–1149. ISSN: 1935-7524.

[3] Gwennaëlle Mabon. "Adaptive Deconvolution on the Non-Negative Real Line: Adaptive Deconvolution on R+". en. In: *Scandinavian Journal of Statistics* 44.3 (Sept. 2017), pp. 707–740. ISSN: 03036898.

[4] Florian Dussap. "Anisotropic Multivariate Deconvolution Using Projection on the Laguerre Basis". In: (2020).

Une vision problème des moments

If our observations are exact, from a density in the class, this is equavalent to a moment problem on the Thorin measure ν with atoms $\boldsymbol{s}$ and weights $\boldsymbol{\alpha}$. But:

(i) Denote $\mu_{\boldsymbol{k}} = \mathbb{E}\left(\boldsymbol{X}^{\boldsymbol{k}} e^{\langle -\mathbf{1}, \boldsymbol{X} \rangle}\right)$, the $\boldsymbol{k}$th derivative of $M(\boldsymbol{t}) = \exp \circ K(\boldsymbol{t})$ taken at $\boldsymbol{t} = -\mathbf{1}$.

(ii) Then (Faa di bruno) there is a bijection between derivatives of K and Laguerre coefficients. The derivatives of $K(\boldsymbol{t}) = \int \ln\left(1 - \langle \boldsymbol{s}, \boldsymbol{t} \rangle\right) \nu(\boldsymbol{s})$ are given as a linear combination of moments of the Thorin measure.

(iii) Unfortunately observations are usually outside the multivariate cone of moments, and Lassere hierarchies are not tractable.

The L2 Laguerre loss is more stable.

An algorithm with a lot of flaws

Algorithm: Laguerre coefficients $(a_{\boldsymbol{k}}(\boldsymbol{\alpha}, \boldsymbol{s}))_{\boldsymbol{k} \leq \boldsymbol{m}}$ of $\mathcal{G}_{d,n}(\boldsymbol{\alpha}, \boldsymbol{s})$ random vectors.

```
Input: Shapes $\boldsymbol{\alpha} \in \mathbb{R}_+^d$, scales $\boldsymbol{s} \in \mathcal{M}_{n,d}(\mathbb{R}_+)$, and truncation threshold $\boldsymbol{m} \in \mathbb{N}^d$
Result: Laguerre coefficients $(a_{\boldsymbol{k}})_{\boldsymbol{k} \leq \boldsymbol{m}}$ of the $\mathcal{G}_{d,n}(\boldsymbol{\alpha}, \boldsymbol{s})$ density
Compute the simplex version of the scales $\boldsymbol{x}_i = \frac{\boldsymbol{s}_i}{1+|\boldsymbol{s}_i|}$ for all $i \in 1, \dots, n$.
Let $\kappa_{\boldsymbol{0}} = -\sum_{i=1}^{n} \alpha_i \ln(1 - |\boldsymbol{x}_i|)$ and $a_{\boldsymbol{0}} = \mu_{\boldsymbol{0}} = \exp(\kappa_{\boldsymbol{0}})$
foreach $\boldsymbol{0} \neq \boldsymbol{k} \leq \boldsymbol{m}$ do
    Let $a_{\boldsymbol{k}} = \mu_{\boldsymbol{k}} = 0$, $j$ be the index of the first $k_i$ that is non-zero, $\boldsymbol{p} = \boldsymbol{k}$ and set $p_j = p_j - 1$.
    Let $\kappa_{\boldsymbol{k}} = (|\boldsymbol{k}| - 1)! \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i^{\boldsymbol{k}}$
    foreach $\boldsymbol{l} \leq \boldsymbol{p}$ do
        Set $\mu_{\boldsymbol{k}} \mathrel{+}= (\mu_{\boldsymbol{l}})(\kappa_{\boldsymbol{k}-\boldsymbol{l}})\binom{\boldsymbol{p}}{\boldsymbol{l}}$ according to efficient Faà di Bruno's algorithm from Miatto[a]
        Set $a_{\boldsymbol{k}} \mathrel{+}= \mu_{\boldsymbol{l}} \binom{\boldsymbol{k}}{\boldsymbol{l}} \frac{(-2)^{|\boldsymbol{l}|}}{\boldsymbol{l}!}$
    end
    Set $a_{\boldsymbol{k}} \mathrel{+}= \mu_{\boldsymbol{k}} \frac{(-2)^{|\boldsymbol{k}|}}{\boldsymbol{k}!}$
end
$\boldsymbol{a} = \sqrt{2}^d \boldsymbol{a}$
Return $L(\boldsymbol{\alpha}, \boldsymbol{s}) = \|f - \hat{f}\|_2^2 = \sum_{\boldsymbol{k} \leq \boldsymbol{m}} (\hat{a}_{\boldsymbol{k}} - a_{\boldsymbol{k}})^2$
```

[a] Filippo M. Miatto. "Recursive Multivariate Derivatives of \$e^{f(X_1,\dots, X_n)}\$ of Arbitrary Order". en. In: *arXiv:1911.11722 [cs, math]* (Nov. 2019).

Les factorielles, c'est dangeureux.

Algorithm: Laguerre coefficients $(a_{\boldsymbol{k}}(\boldsymbol{\alpha}, \boldsymbol{s}))_{\boldsymbol{k}\leq\boldsymbol{m}}$ of $\mathcal{G}_{d,n}(\boldsymbol{\alpha}, \boldsymbol{s})$ random vectors.

```
Input: Shapes $\boldsymbol{\alpha} \in \mathbb{R}_+^d$, scales $\boldsymbol{s} \in \mathcal{M}_{n,d}(\mathbb{R}_+)$, and truncation threshold $\boldsymbol{m} \in \mathbb{N}^d$
Result: Laguerre coefficients $(a_{\boldsymbol{k}})_{\boldsymbol{k}\leq\boldsymbol{m}}$of the $\mathcal{G}_{d,n}(\boldsymbol{\alpha}, \boldsymbol{s})$ density
Compute the simplex version of the scales $\boldsymbol{x}_i = \frac{\boldsymbol{s}_i}{1+|\boldsymbol{s}_i|}$ for all $i \in 1, ..., n$.
Let $\kappa_{\boldsymbol{0}} = -\sum_{i=1}^{n} \alpha_i \ln(1 - |\boldsymbol{x}_i|)$ and $a_{\boldsymbol{0}} = \mu_{\boldsymbol{0}} = \exp(\kappa_{\boldsymbol{0}})$
foreach $\boldsymbol{0} \neq \boldsymbol{k} \leq \boldsymbol{m}$ do
    Let $a_{\boldsymbol{k}} = \mu_{\boldsymbol{k}} = 0$, $j$ be the index of the first $k_i$ that is non-zero, $\boldsymbol{p} = \boldsymbol{k}$ and set $p_j = p_j - 1$.
    Let $\kappa_{\boldsymbol{k}} = (|\boldsymbol{k}| - 1)! \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i^{\boldsymbol{k}}$
    foreach $\boldsymbol{l} \leq \boldsymbol{p}$ do
        Set $\mu_{\boldsymbol{k}}$ += $(\mu_{\boldsymbol{l}})(\kappa_{\boldsymbol{k}-\boldsymbol{l}})\binom{\boldsymbol{p}}{\boldsymbol{l}}$
        Set $a_{\boldsymbol{k}}$ += $\mu_{\boldsymbol{l}}\binom{\boldsymbol{k}}{\boldsymbol{l}}\frac{(-2)^{|\boldsymbol{l}|}}{\boldsymbol{l}!}$
    end
    Set $a_{\boldsymbol{k}}$ += $\mu_{\boldsymbol{k}}\frac{(-2)^{|\boldsymbol{k}|}}{\boldsymbol{k}!}$
end
$\boldsymbol{a} = \sqrt{2}^d \boldsymbol{a}$
Return $L(\boldsymbol{\alpha}, \boldsymbol{s}) = \|f - \hat{f}\|_2^2 = \sum_{\boldsymbol{k}\leq\boldsymbol{m}} (\hat{a}_{\boldsymbol{k}} - a_{\boldsymbol{k}})^2$
```

Mince, recursif donc non distribuable..

Algorithm: Laguerre coefficients $(a_{\mathbf{k}}(\boldsymbol{\alpha},\mathbf{s}))_{\mathbf{k}\leq\mathbf{m}}$ of $\mathcal{G}_{d,n}(\boldsymbol{\alpha},\mathbf{s})$ random vectors.

```
Input: Shapes $\boldsymbol{\alpha} \in \mathbb{R}_+^d$, scales $\mathbf{s} \in \mathcal{M}_{n,d}(\mathbb{R}_+)$, and truncation threshold $\mathbf{m} \in \mathbb{N}^d$
Result: Laguerre coefficients $(a_{\mathbf{k}})_{\mathbf{k}\leq\mathbf{m}}$of the $\mathcal{G}_{d,n}(\boldsymbol{\alpha},\mathbf{s})$ density
Compute the simplex version of the scales $\mathbf{x}_i = \frac{\mathbf{s}_i}{1+|\mathbf{s}_i|}$ for all $i \in 1,...,n$.
Let $\kappa_{\mathbf{0}} = -\sum_{i=1}^{n} \alpha_i \ln(1-|\mathbf{x}_i|)$ and $a_{\mathbf{0}} = \mu_{\mathbf{0}} = \exp(\kappa_{\mathbf{0}})$
foreach $\mathbf{0} \neq \mathbf{k} \leq \mathbf{m}$ do
    Let $a_{\mathbf{k}} = \mu_{\mathbf{k}} = 0$, $j$ be the index of the first $k_i$ that is non-zero, $\mathbf{p} = \mathbf{k}$ and set $p_j = p_j - 1$.
    Let $\kappa_{\mathbf{k}} = (|\mathbf{k}|-1)! \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^{\mathbf{k}}$
    foreach $\mathbf{l} \leq \mathbf{p}$ do
        Set $\mu_{\mathbf{k}} \mathrel{+}= (\mu_{\mathbf{l}})(\kappa_{\mathbf{k}-\mathbf{l}})\binom{\mathbf{p}}{\mathbf{l}}$
        Set $a_{\mathbf{k}} \mathrel{+}= \mu_{\mathbf{l}}\binom{\mathbf{k}}{\mathbf{l}}\frac{(-2)^{|\mathbf{l}|}}{\mathbf{l}!}$
    end
    Set $a_{\mathbf{k}} \mathrel{+}= \mu_{\mathbf{k}}\frac{(-2)^{|\mathbf{k}|}}{\mathbf{k}!}$
end
$\mathbf{a} = \sqrt{2}^d \mathbf{a}$
Return $L(\boldsymbol{\alpha},\mathbf{s}) = \|f-\hat{f}\|_2^2 = \sum_{\mathbf{k}\leq\mathbf{m}} (\hat{a}_{\mathbf{k}} - a_{\mathbf{k}})^2$
```

Euh, c'est convexe ?

Algorithm: Laguerre coefficients $(a_{\boldsymbol{k}}(\boldsymbol{\alpha}, \boldsymbol{s}))_{\boldsymbol{k} \leq \boldsymbol{m}}$ of $\mathcal{G}_{d,n}(\boldsymbol{\alpha}, \boldsymbol{s})$ random vectors.

```
Input: Shapes $\boldsymbol{\alpha} \in \mathbb{R}_+^d$, scales $\boldsymbol{s} \in \mathcal{M}_{n,d}(\mathbb{R}_+)$, and truncation threshold $\boldsymbol{m} \in \mathbb{N}^d$
Result: Laguerre coefficients $(a_{\boldsymbol{k}})_{\boldsymbol{k} \leq \boldsymbol{m}}$of the $\mathcal{G}_{d,n}(\boldsymbol{\alpha}, \boldsymbol{s})$ density
Compute the simplex version of the scales $\boldsymbol{x}_i = \frac{\boldsymbol{s}_i}{1+|\boldsymbol{s}_i|}$ for all $i \in 1, ..., n$.
Let $\kappa_{\boldsymbol{0}} = -\sum_{i=1}^{n} \alpha_i \ln(1 - |\boldsymbol{x}_i|)$ and $a_{\boldsymbol{0}} = \mu_{\boldsymbol{0}} = \exp(\kappa_{\boldsymbol{0}})$
foreach $\boldsymbol{0} \neq \boldsymbol{k} \leq \boldsymbol{m}$ do
    Let $a_{\boldsymbol{k}} = \mu_{\boldsymbol{k}} = 0$, $j$ be the index of the first $k_i$ that is non-zero, $\boldsymbol{p} = \boldsymbol{k}$ and set $p_j = p_j - 1$.
    Let $\kappa_{\boldsymbol{k}} = (|\boldsymbol{k}| - 1)! \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i^{\boldsymbol{k}}$
    foreach $\boldsymbol{l} \leq \boldsymbol{p}$ do
        Set $\mu_{\boldsymbol{k}} \mathrel{+}= (\mu_{\boldsymbol{l}})(\kappa_{\boldsymbol{k}-\boldsymbol{l}})\binom{\boldsymbol{p}}{\boldsymbol{l}}$ Set $a_{\boldsymbol{k}} \mathrel{+}= \mu_{\boldsymbol{l}}\binom{\boldsymbol{k}}{\boldsymbol{l}}\frac{(-2)^{|\boldsymbol{l}|}}{\boldsymbol{l}!}$
    end
    Set $a_{\boldsymbol{k}} \mathrel{+}= \mu_{\boldsymbol{k}}\frac{(-2)^{|\boldsymbol{k}|}}{\boldsymbol{k}!}$
end
$\boldsymbol{a} = \sqrt{2}^d \boldsymbol{a}$
Return $L(\boldsymbol{\alpha}, \boldsymbol{s}) = \|f - \hat{f}\|_2^2 = \sum_{\boldsymbol{k} \leq \boldsymbol{m}} (\hat{a}_{\boldsymbol{k}} - a_{\boldsymbol{k}})^2$ as dense polynomials into $\mu_{\boldsymbol{0}}$ and $\boldsymbol{x}$ ...
```

En résumé

To sum up the problems and solutions we have for the moment, we can say that our loss is :

$$\begin{aligned} &\text{Combinatorial} &&\implies \text{Arbitrary precision} \\ &\text{Highly recursive} &&\implies \text{Compiled code} \\ &\text{Highly non-convex} &&\implies \text{Global optimization} \end{aligned}$$

(i) Arbitrary precision is available in many languages, usually with a wrapper of "mpfr"

(ii) Global optimization routines are available in many languages.

(iii) Compilation can be done in man languages also, beforehand or "JIT".

Few languages allows the three concepts together without recoding a whole library. Julia does.

La notion de nombre de Julia.

Un nombre, c'est vague.

Julia has no requisite for a type to be usable as a number. A number type is just any type that implements the things you need to do with it. Our algorithm requires only the functions $+, -$, $*$, $/$, exp and ln to be implemented (integer powers have a type-agnostic method in Base). Arbitrary precision:

(i) BigFloat: "mpfr" interface integrated in Base, but allocate like crazy
(ii) Arb: Not native code, crashes a lot.
(iii) DoubleFloats.jl
(iv) MultiFloats.jl

But more things can be done:

(i) Dual numbers to compute gradients with forward AD.
(ii) Measurements.jl

Thanks to multiple dispatch, we can compare all the different types of numbers through the same type-agnostic code.

Timings for $(d, n, m, N) = (2, 20, 80, 1000)$

Type	Error	Time	Allocations
Float64	2.68084e110	0.121672	286112.0
MultiFloat{Float64,1}	4.1367e109	0.124149	296656.0
Float128	4.51278e74	1.84456	467600.0
Double64	2.10301e78	0.516504	467600.0
MultiFloat{Float64,2}	3.84925e78	0.26139	486208.0
MultiFloat{Float64,3}	-6.13933e44	0.66715	676464.0
MultiFloat{Float64,4}	3.1583e14	1.33451	864800.0
MultiFloat{Float64,5}	2.367e-8	2.39511	1.05234e6
MultiFloat{Float64,6}	2.36701e-8	4.46531	1.25013e6
MultiFloat{Float64,7}	2.36701e-8	6.50383	1.44102e6
MultiFloat{Float64,8}	2.36701e-8	9.84249	1.63157e6
BigFloat	2.36701e-8	12.6092	8.16053e9

See my discourse thread (link) for more details

Multiple dispatch and type-agnostic code

Multiple dispatch : principe et bienfaits.

Running previous timings was as simple as:

```
...
for T in list_of_types
        inputs = T.(inputs)
        result[T] = Algorithm(inputs) # Type agnostic function
end
...
```

Seek type stability: A separate method of the Algorithm function will be compiled for each set of concrete input types, and if the rest of the stack is also type-stable, the compiler will specialize and compile everything to LLVM, providing a huge performance boost.

The compiler is there to help:

```
@code_native, @code_llvm, @code_warntype,
@btime, @ellapsed, @allocated...
```

Et les librairies d'optimisation ?

There is an insight in the community to code native type-agnostic stuff. This is a very good oportunity for us:

(i) ForwardDiff.jl: Automatic differentiation of loss functions.
(ii) Optim.jl: LBFGS, ParticleSwarm, Ants colonies, ...
(iii) Convex.jl: Disciplined Convex Programming (not for us)
(iv) COSMO.jl: Native general conic programming, to pair with Convex.jl.
(v) NonConvex.jl: Many Others algorithms

All these algorithms were coded once and works on all types. You could even differentiate through a gradient descent using Dual numbers, or optimize with ball arithmetics, it just works.

Easy DSL through Macros !

Il semble qu'Optim.ParticleSwarm ne parralélise pas tout seul...

Modifying a package is quite easy. Simply add the package and dev it:

```
julia> Pkg.add("Optim")
julia> Pkg.dev("Optim")
```

Then modify the source to parallelize the right loop:

```
...
Threads.@threads for i in 1:n_particles
        score[i] = value(f, X[:, i])
end
...
```

Commit, and a simple pull-request is enough to contribute your changes !

```
$ git add .
$ git commit -m "Threading the PSO loop"
```

Conclusion

Conclusion

Outside of the Julia ecosystem, optimisation in arbitrary precision usually requires re-implementation of at least one library:

(i) In python, you may use “mpmath” or “numba”, but you cannot use both. (and you still lack the optimisation routine)

(ii) In R, “rmpfr” cannot work with LBFS’s C++ routines: you need to recode everything.

Julia solves the two-language problem: Julia’s bare principles allows for a high level of code sharing and compatiblity between codebases that did not expect each other.

Many things are still to be done in the ecosystem of packages since the language is young, and your Julia code is waited !