# Fungrim: The Mathematical Functions Grimoire

Fredrik Johansson

Inria Bordeaux

FastRelax Workshop, ENS de Lyon, France
May 23, 2019

# What is Fungrim?

`http://fungrim.org`

An attempt to make a better

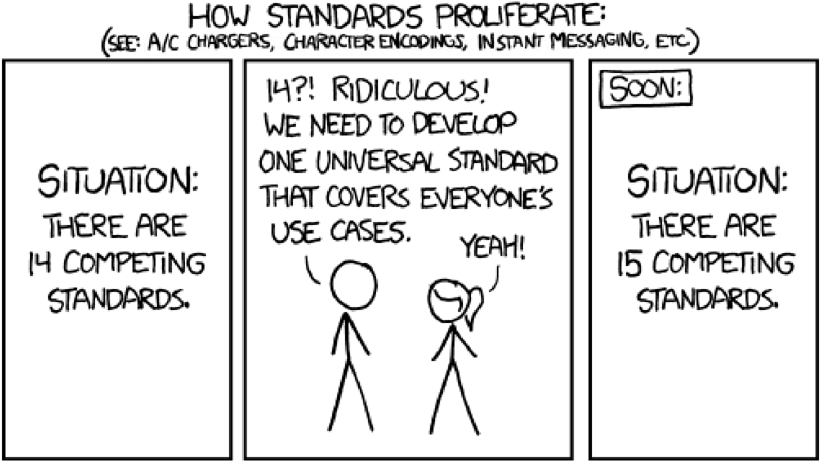
1. reference work
2. computer algebra library

for special functions

---

*grimoire* = book of magic formulas

# Relevant XKCD




https://xkcd.com/927/

# My motivation

I have spent a lot of time implementing special (and general) functions in: SymPy, mpmath, SageMath, FLINT, Arb, Nemo

What's hard? 50 / 50:

- Finding the right formulas/theorems
- Implementation aspects

# My motivation

I have spent a lot of time implementing special (and general) functions in: SymPy, mpmath, SageMath, FLINT, Arb, Nemo

What's hard? 50 / 50:

- Finding the right formulas/theorems
- Implementation aspects

**Short-term goal**: collect knowledge about special functions, present it in the form *I would have found useful*

**Long-term goal**: tools for symbolic computation and symbolic-numeric algorithms (integration, code generation...)

# Some reasons why the literature is frustrating to use

1. Vague or missing definitions
2. Conditions on variables not stated, ambiguous, or depend on non-local context
3. The formula I want can be derived by combining equation (43) with theorems 5 and 12... in a simple 10-page calculation, left as an exercise for the reader
4. The dreaded "$\approx$" sign
5. Errors (typos or more serious)
6. Text text text text text text text text text text

# Some reasons why the literature is frustrating to use

1. Vague or missing definitions
2. Conditions on variables not stated, ambiguous, or depend on non-local context
3. The formula I want can be derived by combining equation (43) with theorems 5 and 12… in a simple 10-page calculation, left as an exercise for the reader
4. The dreaded “$\approx$” sign
5. Errors (typos or more serious)
6. Text text text text text text text text text text

*I’m personally as guilty as anyone, on all counts*

# Problems with existing reference works

| | dlmf.nist.gov | functions.wolfram.com | wikipedia.org |
|---|---|---|---|
| Open source? | × | × | ✓ |
| Symbolic? | × | ✓ | × |
| Misc pros | Good presentation (edited by experts) | Well-structured, exhaustive | Usually comprehensive |
| Misc cons | Terse, missing useful formulas, sometimes vague | Mathematica quirks and bugs, sometimes ugly formulas, missing some categories of info | Text-heavy, much trivia, often vague, inconsistent |

# Content goals for Fungrim

- Formulas as symbolic, machine-readable theorems
- Functions/operators have a globally consistent meaning (integration paths, values on branch cuts, limits, etc.)
- Formulas include full conditions of validity ("assumptions") for all free variables (e.g. $x \in \mathbb{C} \setminus \{0\}$)
- Comprehensive: aim for good coverage of all the common special functions in mathematics
- Good coverage of inequalities, with explicit constants

# Presentation goals for Fungrim

- Simple and fast to browse (including mobile!)
- Permanent ID and URL for each formula
- Beautiful formula rendering (Fungrim formula language $\rightarrow$ TeX $\rightarrow$ KaTeX $\rightarrow$ HTML)
- Instant access to TeX code to copy and paste
- Instant access to symbolic representation
- Hyperlinked symbol definitions
- TODO: export to other languages, search functionality, browsing based on metadata

# Non-goals (for now)

**Formal proofs**

- Randomized testing (to be done!) should be adequate to provide a high level of reliability
- Of course, future integration with formal proof efforts would make sense

**Fully computer-generated content**

- Related: the Dynamic Dictionary of Mathematical Functions (`http://ddmf.msr-inria.inria.fr/1.9.1/ddmf`)

**Covering all of mathematics**

- Just special functions and elements of classical analysis

# Long-term goal: symbolic computation

Three essential parts of a computer algebra system:

1. Symbolic representation of mathematical objects
2. Mathematical algorithms / rewrite rules
3. The surrounding interface (programming language, etc.)

**Idea: build an open source library of symbolic data and rewrite rules for special functions, independent of other features of computer algebra systems**

(plus applications!)

# Inspiration 1: Rubi by Albert D. Rich

`https://rulebasedintegration.org`

> *[Rubi] uses pattern matching to uniquely determine which of its over 6600 integration rules to apply to a given integrand*

> *Rubi dramatically out-performs other symbolic integrators, including Maple and Mathematica*

> *Certainly much of analysis including equation solving, expression simplification, differentiation, summation, limits, etc. can be automated using this paradigm*

# Inspiration 2: design flaws in current computer algebra systems

# Inspiration 2: design flaws in current computer algebra systems

**Implementations that mix/confuse abstractions**

- Exhibit A: much of SageMath
- Possible solution: clear separation of concerns

# Inspiration 2: design flaws in current computer algebra systems

**Implementations that mix/confuse abstractions**

- Exhibit A: much of SageMath
- Possible solution: clear separation of concerns

**Unwanted or opaque automatic “simplification”**

- Possible solution: give user more control, inspection and choice of rewrite rules

# Inspiration 2: design flaws in current computer algebra systems

**Implementations that mix/confuse abstractions**

- Exhibit A: much of SageMath
- Possible solution: clear separation of concerns

**Unwanted or opaque automatic “simplification”**

- Possible solution: give user more control, inspection and choice of rewrite rules

**Incorrect algorithms / rewrite rules**

- Algorithms that ignore conditions, simplify “modulo special cases”
- Possible solution: track conditions and base rules on theorems instead of wishful thinking

# A simple symbolic integral: $\int_1^2 x^a dx$

Mathematica:

---

In[5]:= `Integrate[x ^ (-1) , {x, 1, 2}]`

Out[5]= `Log[2]`

In[7]:= `Integrate[x ^ a, {x, 1, 2}]`

Out[7]= $\frac{-1+2^{1+a}}{1+a}$

In[8]:= `Integrate[x ^ a, {x, 1, 2}] /. (a → -1)`

Power: Infinite expression $\frac{1}{0}$ encountered.

Infinity : Indeterminate expression 0 ComplexInfinity encountered.

Out[8]= `Indeterminate`

# A simple symbolic integral: $\int_1^2 x^a dx$

SymPy does the right thing:

---

```
>>> integrate(x**a, (x, 1, 2))
Piecewise((2**(a + 1)/(a + 1) - 1/(a + 1),
    (a > -oo) & (a < oo) & Ne(a, -1)), (log(2), True))

>>> integrate(x**a, (x, 1, 2)).subs(a, -1)
log(2)
```

# A simple symbolic integral: $\int_1^2 x^a dx$

SymPy does the right thing:

```
>>> integrate(x**a, (x, 1, 2))
Piecewise((2**(a + 1)/(a + 1) - 1/(a + 1),
    (a > -oo) & (a < oo) & Ne(a, -1)), (log(2), True))

>>> integrate(x**a, (x, 1, 2)).subs(a, -1)
log(2)
```

(Well, almost:)

```
>>> integrate(x**a, (x, 1, 2)).subs(a, I)
Traceback (most recent call last):
  ...
TypeError: Invalid comparison of complex I
```

# A simple symbolic integral: $\int_1^2 x^a dx$

SageMath... at least tries to help, in this case:

---

```
sage: var("x a")
(x, a)
sage: integrate(x**a, x, 1, 2)
---------------------------------------------------------------------------
  ...
ValueError: Computation failed since Maxima requested
additional constraints; using the 'assume' command
before evaluation *may* help (example of legal syntax
is 'assume(a>0)', see `assume?` for more details)
Is a positive, negative or zero?
```

# A simplification: ${}_1F_1(-1,-1,x) = e^x \ldots$ or $1+x$?

Mathematica:

```
In[∘]:= Hypergeometric1F1[n, m, 1] /. {m → -1, n → -1, x → 1}
Out[∘]= 2

In[∘]:= (Hypergeometric1F1[n, m, x] /. {m → n}) /. {n → -1, x → 1}
Out[∘]= e
```

SymPy:

```
>>> simplify(hyper([n],[m],x).subs({m:-1, n:-1, x:1}))
2
>>> simplify(hyper([n],[m],x).subs(m, n)).subs({n:-1, x:1})
E
```

# Computing the wrong thing by design?

Which is better?

1. Do something fast/simple (but possibly incorrect) – perhaps we can check the result later?
2. Do something guaranteed to be correct (but possibly slow/complicated)

Analogy with ordinary numerics / interval arithmetic

# Computing the wrong thing by design?

R. Corless and D. Jeffrey, “Well… It Isn’t Quite That Simple”, ACM SIGSAM Bulletin, 1992:

> *The automatic exploration of conditions or alternative results requires considerable computational resources, and for the sake of speed there is an attraction to picking one ’obvious’ answer. [...] The difficulty is to balance efficiency against correctness.*

# Computing the wrong thing by design?

R. Corless and D. Jeffrey, “Well… It Isn’t Quite That Simple”, ACM SIGSAM Bulletin, 1992:

> *The automatic exploration of conditions or alternative results requires considerable computational resources, and for the sake of speed there is an attraction to picking one ’obvious’ answer. [...] The difficulty is to balance efficiency against correctness.*

---

Something seems wrong when 27 years later, even trivial cases don’t work by default

No new mathematical ideas are needed here, just working from correct foundations