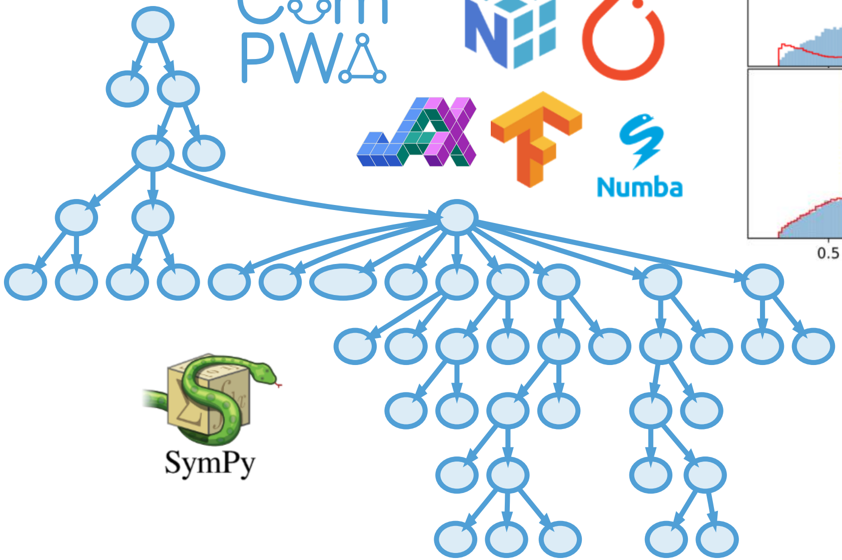
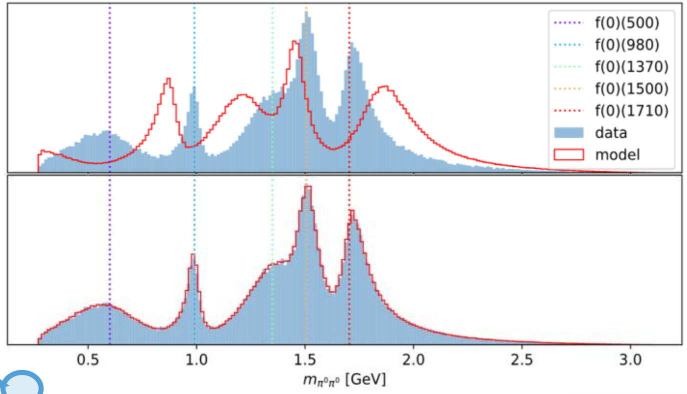
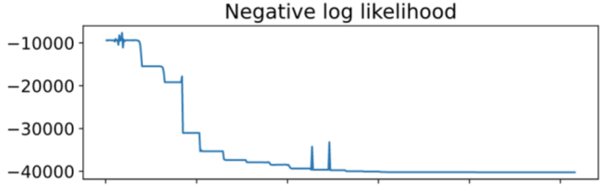
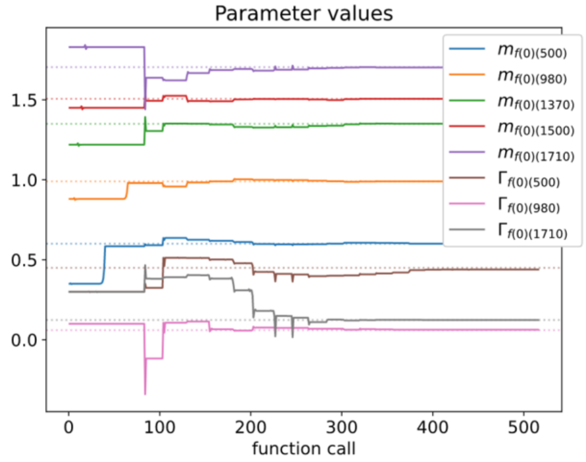








RUHR-UNIVERSITÄT BOCHUM

The ComPWA project

Self-documenting model building and improved fit performance with a Computer Algebra System

26 July 2023 PyHEP.dev Workshop Remco de Boer

Self-introduction

Remco de Boer, PhD Student at Ruhr University Bochum
(finishing April, then postdoc)

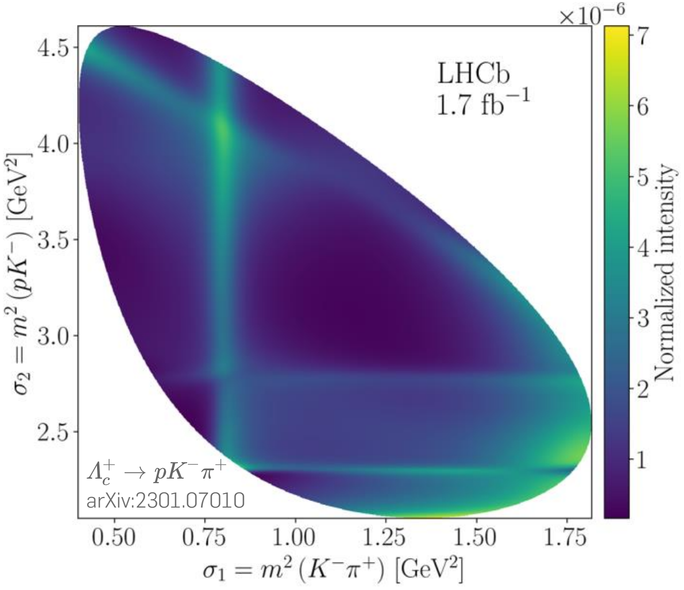
- Research: amplitude analysis (partial wave analysis)
  - Analyzing N^* resonances in J/ψ decays for the BESIII collaboration
  - Also in PANDA collaboration at FAIR, Darmstadt
- Developer for the Common Partial Wave Analysis project (ComPWA)
  → Next slides: symbolic amplitude models + JAX, TF, …
- Additional interests:
  - Differentiable programming for PWA
  - Documentation: big fan of Executable Book Project (Jupyter Book etc.)
  - Code quality and software maintainability

  → Narrowing gab between users and developers

Context: amplitude analysis software

What makes amplitude analysis challenging?

- Unbinned, multidimensional problem set
- Complicated (complex!) parametrizations and estimators
  - need to quickly try out different parameterizations
  - fits can take several weeks
- Theory is hard to get into
- Relatively small community (but growing interest!)



Context: amplitude analysis software

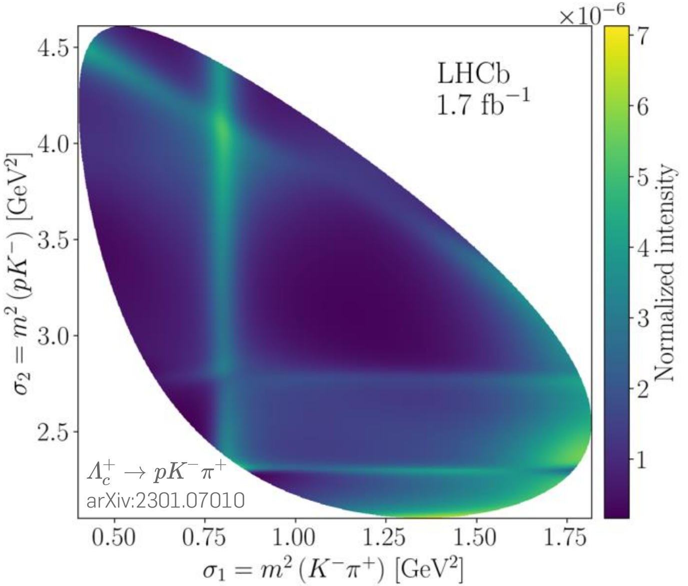
What makes amplitude analysis challenging?

- Unbinned, multidimensional problem set
- Complicated (complex!) parametrizations and estimators
  - need to quickly try out different parameterizations
  - fits can take several weeks
- Theory is hard to get into
- Relatively small community (but growing interest!)

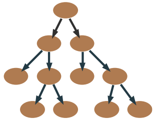
fast computations

flexibility

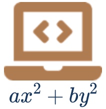
documentation



Mission: bring code closer to theory



High performance through **computational back-ends** from ML and data science



Flexibility through a **CAS-assisted model building**

Academic continuity through **living documentation**

Computational backends

Tools from the ML and data science community that allow us to outsource heavy computations:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation
- Support for multithreading, GPUs, …

```
@tf.function(jit_compile=True)
def my_expression(x, y, z):
    return x + y * z
```

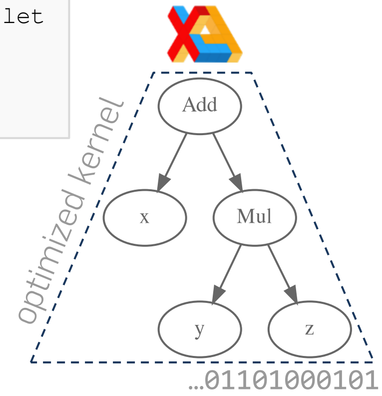
Converted to device-agnostic XLA code

```
{ lambda ; a:i32[] b:i32[] c:i32[]. let
    d:i32[] = mul b c
    e:i32[] = add a d
  in (e,) }
```

Heavy lifting by optimized backend

```
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j];
  }
}
```

Numba



Computational backends

Tools from the ML and data science community that allow us to outsource heavy computations:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation
- Support for multithreading, GPUs, …

```python
@tf.function(jit_compile=True)
def my_expression(x, y, z):
    return x + y * z
```

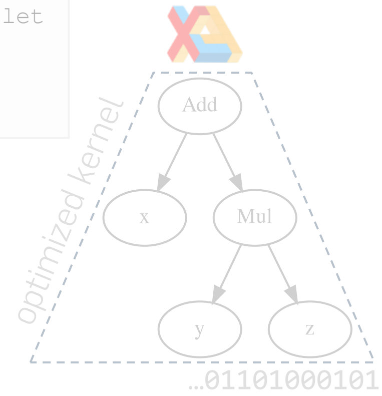
Converted to device-agnostic XLA code

Usually all that the user needs to do

```
{ lambda ; a:i32[] b:i32[] c:i32[]. let
    d:i32[] = mul b c
    e:i32[] = add a d
  in (e,) }
```

```c
for (i = 0; i < rows; i++): {
  for (j = 0; j < columns; j++): {
    c[i][j] = a[i][j]*b[i][j];
  }
}
```

Heavy lifting by optimized backend



Numba

RUHR UNIVERSITÄT BOCHUM RUB

Symbolic amplitude models

A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

```
import sympy as sp
N, s, m0, w0 = sp.symbols("N s m0 Gamma0")
N / (m0**2 - sp.I * m0 * w0 - s)
```

SymPy

$$\frac{N}{m_0^2 - i m_0 \Gamma_0 - s}$$

Quite common already for theoreticians: quickly inspect and visualize some lineshape with Maple, Mathematica, Matlab, etc...

Symbolic amplitude models

A new technique: formulate your amplitude model with a Computer Algebra System

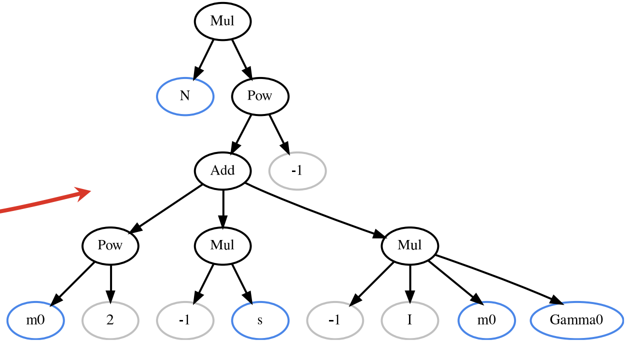
- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

```python
import sympy as sp
N, s, m0, w0 = sp.symbols("N s m0 Gamma0")
N / (m0**2 - sp.I * m0 * w0 - s)
```

SymPy

$$\frac{N}{m_0^2 - i m_0 \Gamma_0 - s}$$

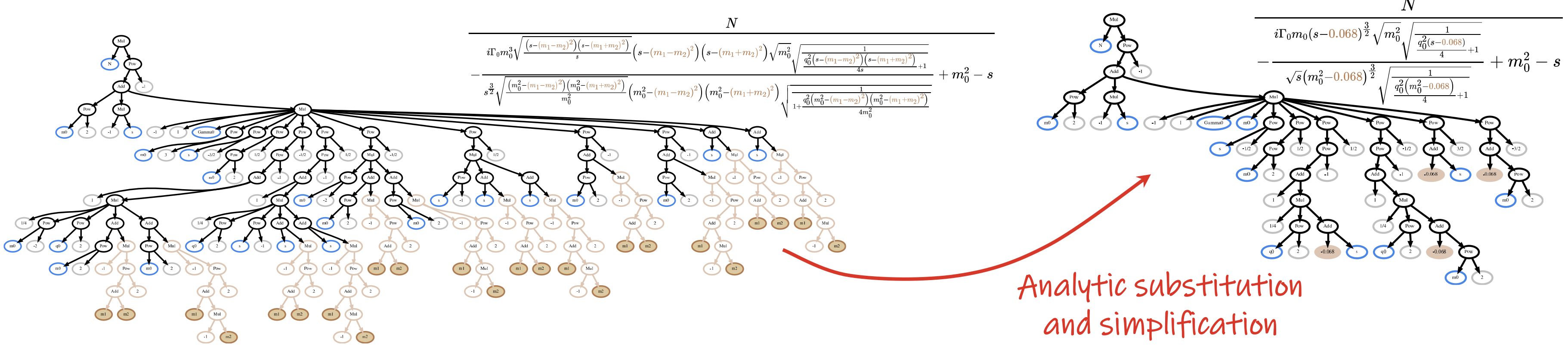
CAS represents expression as a tree



Symbolic amplitude models

A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

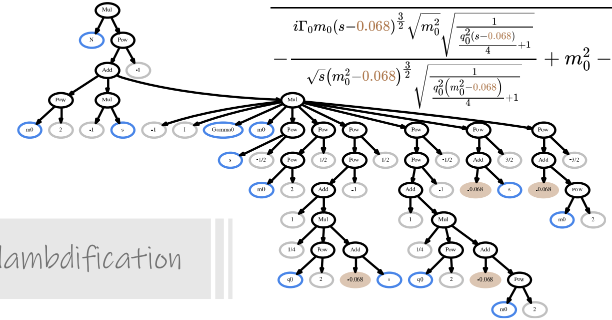


Symbolic amplitude models

A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

$$\frac{N}{-\frac{i\Gamma_0 m_0 (s-0.068)^{\frac{3}{2}}\sqrt{m_0^2}\sqrt{\frac{1}{\frac{q_0^2(s-0.068)}{4}+1}}}{\sqrt{s}\left(m_0^2-0.068\right)^{\frac{3}{2}}\sqrt{\frac{1}{\frac{q_0^2\left(m_0^2-0.068\right)}{4}+1}}}+m_0^2-s}$$




```
function out1 = my_expr(Gamma0, N, m0, s)

 out1 = N./(-1i*Gamma0.*m0.^3.*sqrt((s - 0.25).*(s - 0.01)./s).*(1
+ (m0.^2 - 0.25).*(m0.^2 - 0.01)./(4*m0.^2)).*(s - 0.25).*(s -
0.01).*sqrt(m0.^2)./(s.^(3/2).*sqrt((m0.^2 - 0.25).*(m0.^2 -
0.01)./m0.^2).*(1 + (s - 0.25).*(s - 0.01)./(4*s)).*(m0.^2 -
0.25).*(m0.^2 - 0.01)) + m0.^2 - s);

end
```

Symbolic amplitude models

$ax^2 + by^2$

A new technique: formulate your amplitude model with a Computer Algebra System

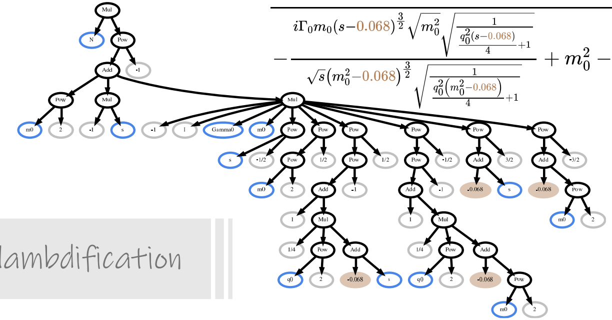
- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

```fortran
REAL*8 function my_expr(Gamma0, N, m0, s)
implicit none
REAL*8, intent(in) :: Gamma0
REAL*8, intent(in) :: N
REAL*8, intent(in) :: m0
REAL*8, intent(in) :: s

my_expr = N/(-cmplx(0,1)*Gamma0*m0**3*sqrt((s - 0.25d0)*(s - 0.01d0)/s)* & (1 +
     (1.0d0/4.0d0)*(m0**2 - 0.25d0)*(m0**2 - 0.01d0)/m0**2)*(s - & 0.25d0)*(s -
     0.01d0)*sqrt(m0**2)/(s**(3.0d0/2.0d0)*sqrt((m0**2 - & 0.25d0)*(m0**2 -
     0.01d0)/m0**2)*(1 + (1.0d0/4.0d0)*(s - 0.25d0)*( & s - 0.01d0)/s)*(m0**2 -
     0.25d0)*(m0**2 - 0.01d0)) + m0**2 - s)
end function
```

SymPy lambdification

$$\frac{N}{-\frac{i\Gamma_0 m_0 (s-0.068)^{\frac{3}{2}} \sqrt{m_0^2} \sqrt{\frac{1}{\frac{q_0^2(s-0.068)}{4}+1}}}{\sqrt{s}\left(m_0^2-0.068\right)^{\frac{3}{2}} \sqrt{\frac{1}{\frac{q_0^2\left(m_0^2-0.068\right)}{4}+1}}} + m_0^2 - s}$$



RUHR UNIVERSITÄT BOCHUM RUB

Symbolic amplitude models

$ax^2 + by^2$

A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

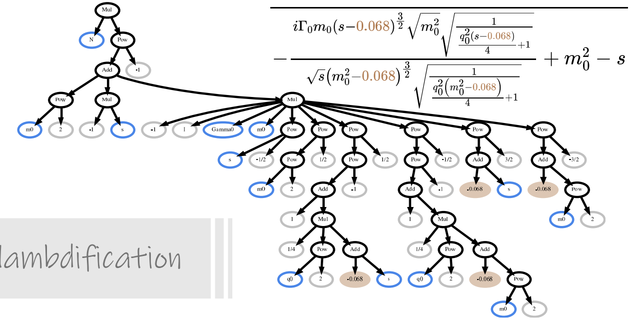
```c
// my_expr.h
#ifndef PROJECT__MY_EXPR__H
#define PROJECT__MY_EXPR__H
double my_expr(double Gamma0, double N, double m0, double s);
#endif

// my_expr.c
#include "my_expr.h"
#include <math.h>

double my_expr(double Gamma0, double N, double m0, double s) {
   double my_expr_result;
   return N/(-I*Gamma0*pow(m0, 3)*sqrt((s - 0.25)*(s - 0.01)/s)*(1 + (1.0/4.0)*(pow(m0, 2) -
      0.25)*(pow(m0, 2) - 0.01)/pow(m0, 2))*(s - 0.25)*(s - 0.01)*sqrt(pow(m0, 2))/(pow(s,
      3.0/2.0)*sqrt((pow(m0, 2) - 0.25)*(pow(m0, 2) - 0.01)/pow(m0, 2))*(1 + (1.0/4.0)*(s -
      0.25)*(s - 0.01)/s)*(pow(m0, 2) - 0.25)*(pow(m0, 2) - 0.01)) + pow(m0, 2) - s);
```

SymPy lambdification

$$\frac{N}{-\frac{i\Gamma_0 m_0 (s-0.068)^{\frac{3}{2}} \sqrt{m_0^2} \sqrt{\frac{1}{\frac{q_0^2(s-0.068)}{4}+1}}}{\sqrt{s}\left(m_0^2-0.068\right)^{\frac{3}{2}} \sqrt{\frac{1}{\frac{q_0^2\left(m_0^2-0.068\right)}{4}+1}}} + m_0^2 - s}$$



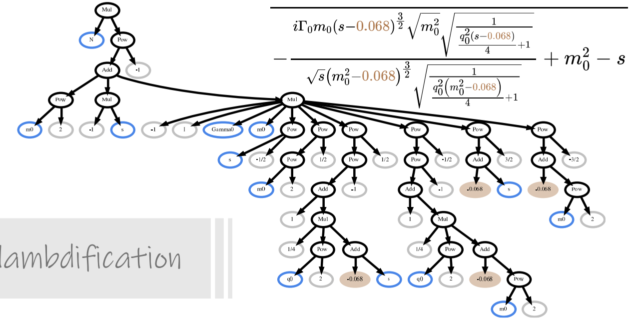
Symbolic amplitude models

A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

```python
@jax.jit
def _lambdifygenerated(Gamma0, N, m0, s):
    return N / (
        -1j
        * Gamma0
        * m0
        * ((1 / 4) * m0**2 + 0.9831)
        * (s - 0.0676) ** (3 / 2)
        * sqrt(m0**2)
        / (sqrt(s) * (m0**2 - 0.0676) ** (3 / 2) * ((1 / 4) * s + 0.9831))
        + m0**2
        - s
    )
```

$$\frac{N}{-\frac{i\Gamma_0 m_0 (s-0.068)^{\frac{3}{2}}\sqrt{m_0^2}\sqrt{\frac{1}{\frac{q_0^2(s-0.068)}{4}+1}}}{\sqrt{s}\left(m_0^2-0.068\right)^{\frac{3}{2}}\sqrt{\frac{1}{\frac{q_0^2\left(m_0^2-0.068\right)}{4}+1}}}+m_0^2-s}$$

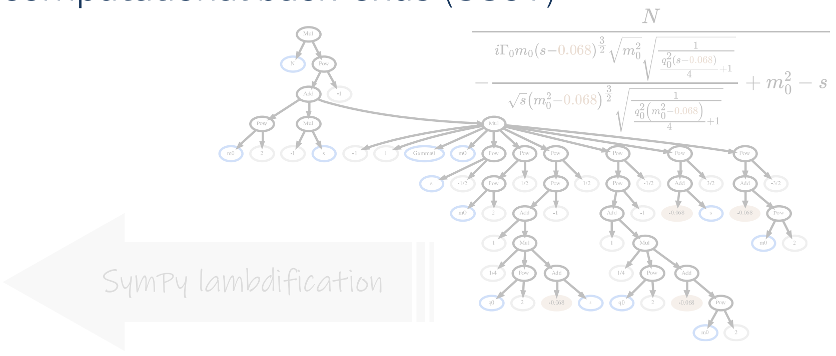


Symbolic amplitude models

A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

```
@jax.jit
def _lambdifygenerated(Gamma0, N, m0, s):
    return N / (
        -1j
        * Gamma0
        * m0
        * ((1 / 4) * m0**2 + 0.9831)
        * (s - 0.0676) ** (3 / 2)
        * sqrt(m0**2)
        / (sqrt(s) * (m0**2 - 0.0676) ** (3 / 2) * ((1 / 4) * s + 0.9831))
        + m0**2
        - s
    )
```



Works just as well for models with **tens of thousands of nodes**

Symbolic amplitude models

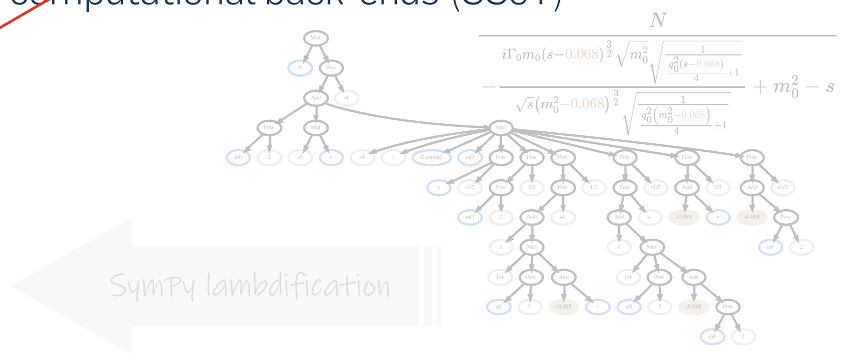
A new technique: formulate your amplitude model with a Computer Algebra System

- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

Can also be used for **serialization**!

```python
@jax.jit
def _lambdifygenerated(Gamma0, N, m0, s):
    return N / (
        -1j
        * Gamma0
        * m0
        * ((1 / 4) * m0**2 + 0.9831)
        * (s - 0.0676) ** (3 / 2)
        * sqrt(m0**2)
        / (sqrt(s) * (m0**2 - 0.0676) ** (3 / 2) * ((1 / 4) * s + 0.9831))
        + m0**2
        - s
    )
```

Works just as well for models with **tens of thousands of nodes**



lc2pkpi-polarimetry.docs.cern.ch

Self-documenting workflow

$\Lambda_c \to p\,K\,\pi$ polarimetry

Search the docs ...

TABLE OF CONTENTS

EXTERNAL LINKS

- arXiv:2301.07010
- ComPWA
- GitHub repository
- CERN GitLab (frozen)

Version 0.0.9 (18/01/2023 23:05:35)

Polarimetry in $\Lambda_c^+ \to p\,K^-\,\pi^+$

DOI 10.48550/arXiv.2301.07010 DOI 10.5281/zenodo.7544989

Λ_c^+ polarimetry using the dominant hadronic mode

The polarimeter vector field for multibody decays of a spin-half baryon is introduced as a generalisation of the baryon asymmetry parameters. Using a recent amplitude analysis of the $\Lambda_c^+ \to pK^-\pi^+$ decay performed at the LHCb experiment, we compute the distribution of the kinematic-dependent polarimeter vector for this process in the space of Mandelstam variables to express the polarised decay rate in a model-agnostic form. The obtained representation can facilitate polarisation measurements of the Λ_c^+ baryon and eases inclusion of the $\Lambda_c^+ \to pK^-\pi^+$ decay mode in hadronic amplitude analyses.

Σ Symbolic expressions

Compute the amplitude model over large data samples with symbolic expressions.

JSON grids

Reuse the computed polarimeter field in any amplitude analysis involving Λ_c^+.

Inspect interactively

Investigate how parameters in the amplitude model affect the polarimeter field.

Compute polarization

Learn how to determine the polarization vector using the polarimeter field.

Download this website as a single PDF file

This website shows all analysis results that led to the publication of LHCb-PAPER-2022-044. More information on this publication can be found on the following pages:

Workflow powered recent study by LHCb (arXiv:2301.07010, JHEP):

- Complete polarimetry analysis performed with symbolic expressions in **Jupyter notebooks**
- Automatically rendered as webpages as the research progressed
- Analysis results **fully reproducible** in around 2 hours

lc2pkpi-polarimetry.docs.cern.ch

Self-documenting workflow

$\Lambda_c \to p\, K\, \pi$ polarimetry

Search the docs ...

TABLE OF CONTENTS

EXTERNAL LINKS

arXiv:2301.07010
ComPWA
GitHub repository
CERN GitLab (frozen)

Version 0.0.9 (18/01/2023 23:05:35)

Contents

1.2.1. Spin-alignment amplitude

The full intensity of the amplitude model is obtained by summing the following aligned amplitude over all helicity values λ_i in the initial state 0 and final states $1, 2, 3$:

```
model_choice = 0
amplitude_builder = load_model_builder(
    model_file="../data/model-definitions.yaml",
    particle_definitions=particles,
    model_id=model_choice,
)
model = amplitude_builder.formulate()
```

▶ Show code cell source

$$\sum_{\lambda_0'=-1/2}^{1/2}\sum_{\lambda_1'=-1/2}^{1/2} A^1_{\lambda_0',\lambda_1'} d^{\frac{1}{2}}_{\lambda_1',\lambda_1}\left(\zeta^1_{1(1)}\right) d^{\frac{1}{2}}_{\lambda_0,\lambda_0'}\left(\zeta^0_{1(1)}\right) + A^2_{\lambda_0',\lambda_1'} d^{\frac{1}{2}}_{\lambda_1',\lambda_1}\left(\zeta^1_{2(1)}\right) d^{\frac{1}{2}}_{\lambda_0,\lambda_0'}\left(\zeta^0_{2(1)}\right) + A^3_{\lambda_0',\lambda_1'} d^{\frac{1}{2}}_{\lambda_1',\lambda_1}\left(\zeta^1_{3(1)}\right) d^{\frac{1}{2}}_{\lambda_0,\lambda_0'}\left(\zeta^0_{3(1)}\right)$$

Note that we simplified notation here: the amplitude indices for the spinless states are not rendered and their corresponding Wigner-d alignment functions are simply 1.

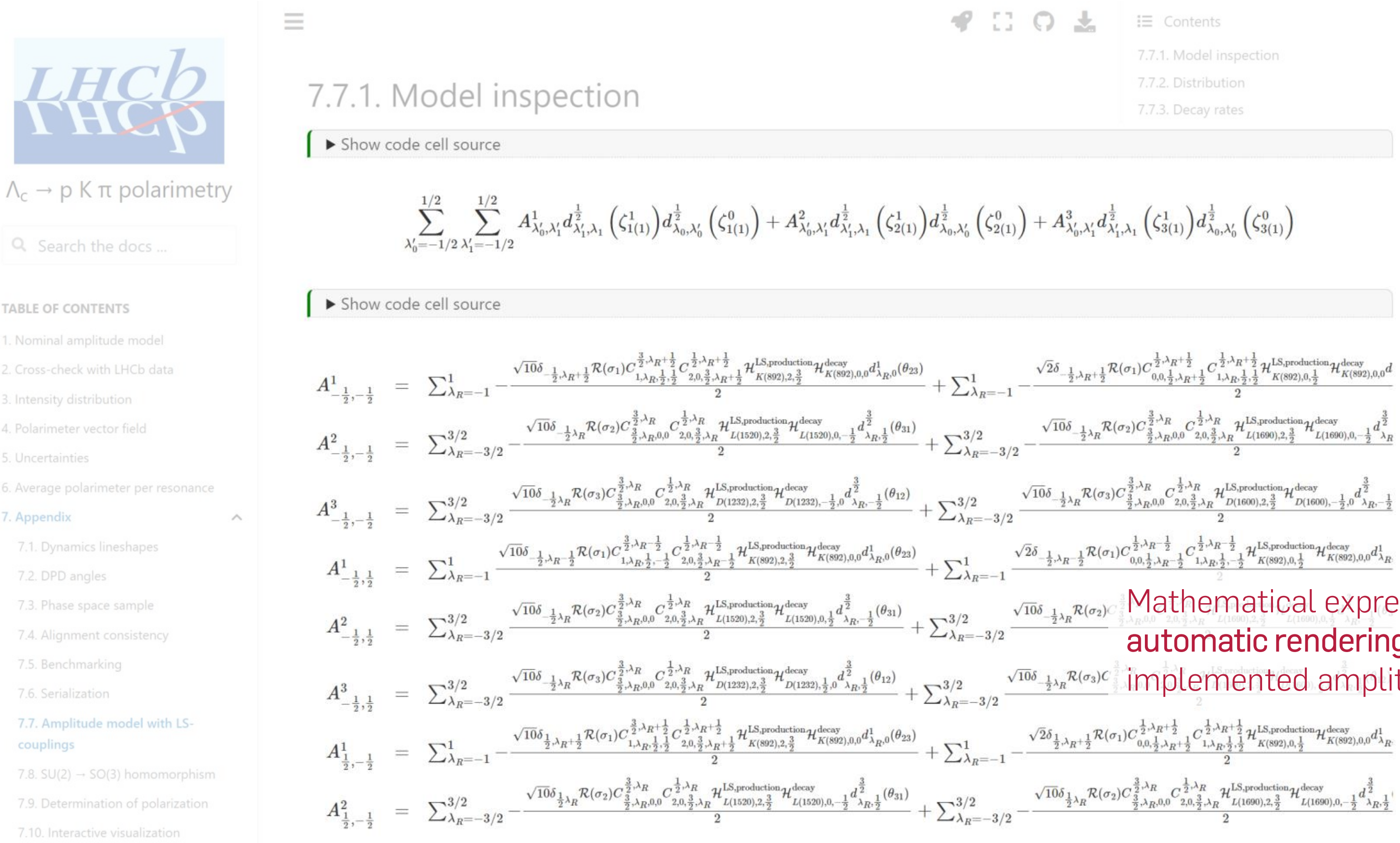
The relevant $\zeta^i_{j(k)}$ angles are defined as:

▶ Show code cell source

$$\begin{aligned}
\zeta^0_{1(1)} &= 0 \\
\zeta^1_{1(1)} &= 0 \\
\zeta^0_{2(1)} &= -\operatorname{acos}\left(\frac{-2m_0^2(-m_1^2-m_2^2+\sigma_3)+(m_0^2+m_1^2-\sigma_1)(m_0^2+m_2^2-\sigma_2)}{\sqrt{\lambda(m_0^2,m_2^2,\sigma_2)}\sqrt{\lambda(m_0^2,\sigma_1,m_1^2)}}\right)
\end{aligned}$$

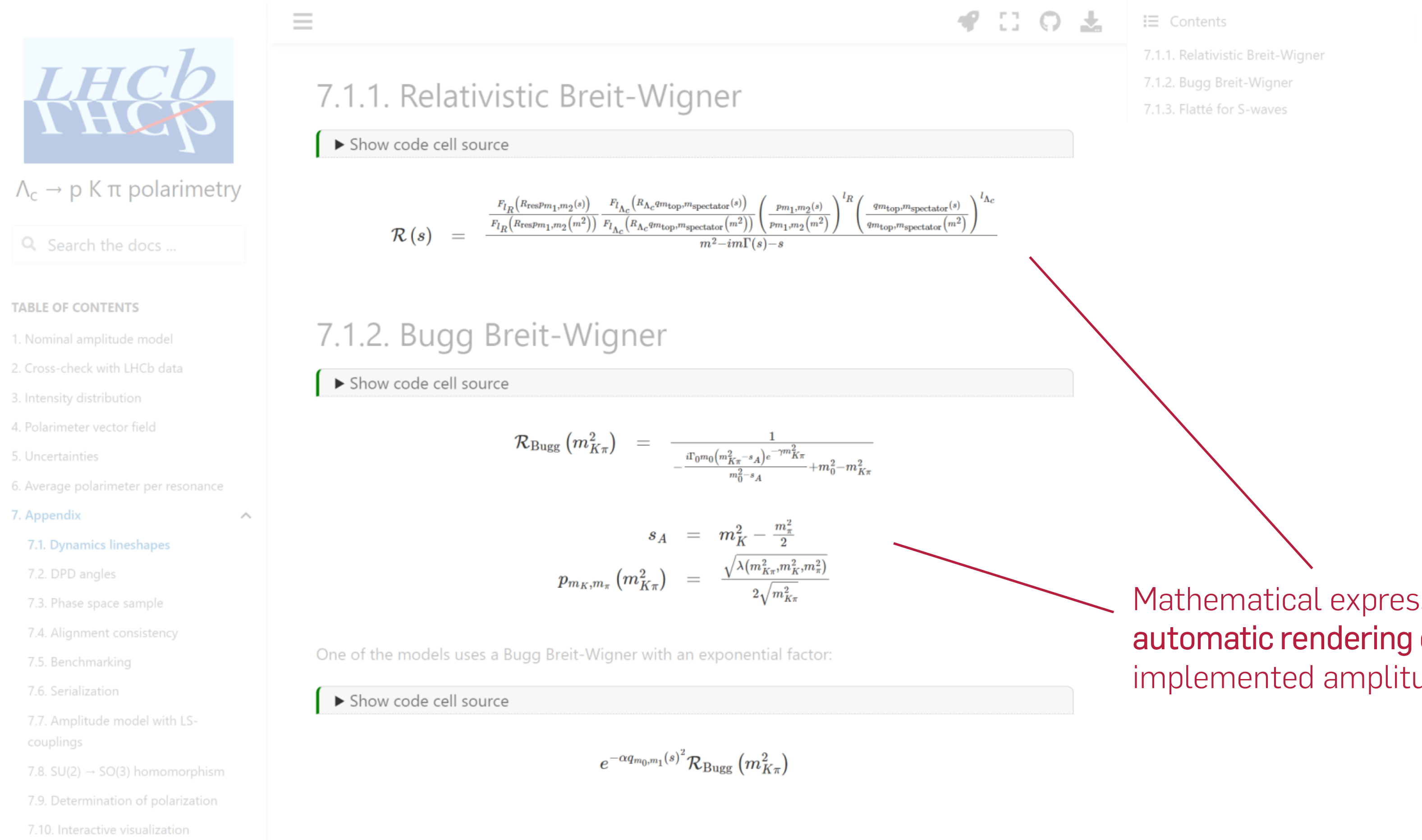
Mathematical expressions are **automatic rendering** of the implemented amplitude models

Self-documenting workflow



Mathematical expressions are **automatic rendering** of the implemented amplitude models

Self-documenting workflow



Λc → p K π polarimetry

Search the docs ...

TABLE OF CONTENTS

Contents

7.1.1. Relativistic Breit-Wigner

▶ Show code cell source

$$\mathcal{R}\left(s\right) = \frac{\frac{F_{l_R}\left(R_\mathrm{res} p_{m_1,m_2}(s)\right)}{F_{l_R}\left(R_\mathrm{res} p_{m_1,m_2}\left(m^2\right)\right)} \frac{F_{l_{\Lambda_c}}\left(R_{\Lambda_c} q_{m_\mathrm{top},m_\mathrm{spectator}}(s)\right)}{F_{l_{\Lambda_c}}\left(R_{\Lambda_c} q_{m_\mathrm{top},m_\mathrm{spectator}}\left(m^2\right)\right)} \left(\frac{p_{m_1,m_2}(s)}{p_{m_1,m_2}\left(m^2\right)}\right)^{l_R} \left(\frac{q_{m_\mathrm{top},m_\mathrm{spectator}}(s)}{q_{m_\mathrm{top},m_\mathrm{spectator}}\left(m^2\right)}\right)^{l_{\Lambda_c}}}{m^2 - im\Gamma(s) - s}$$

7.1.2. Bugg Breit-Wigner

▶ Show code cell source

$$\mathcal{R}_\mathrm{Bugg}\left(m_{K\pi}^2\right) = \frac{1}{- \frac{i\Gamma_0 m_0 \left(m_{K\pi}^2 - s_A\right) e^{-\gamma m_{K\pi}^2}}{m_0^2 - s_A} + m_0^2 - m_{K\pi}^2}$$

$$s_A = m_K^2 - \frac{m_\pi^2}{2}$$

$$p_{m_K,m_\pi}\left(m_{K\pi}^2\right) = \frac{\sqrt{\lambda\left(m_{K\pi}^2, m_K^2, m_\pi^2\right)}}{2\sqrt{m_{K\pi}^2}}$$

One of the models uses a Bugg Breit-Wigner with an exponential factor:

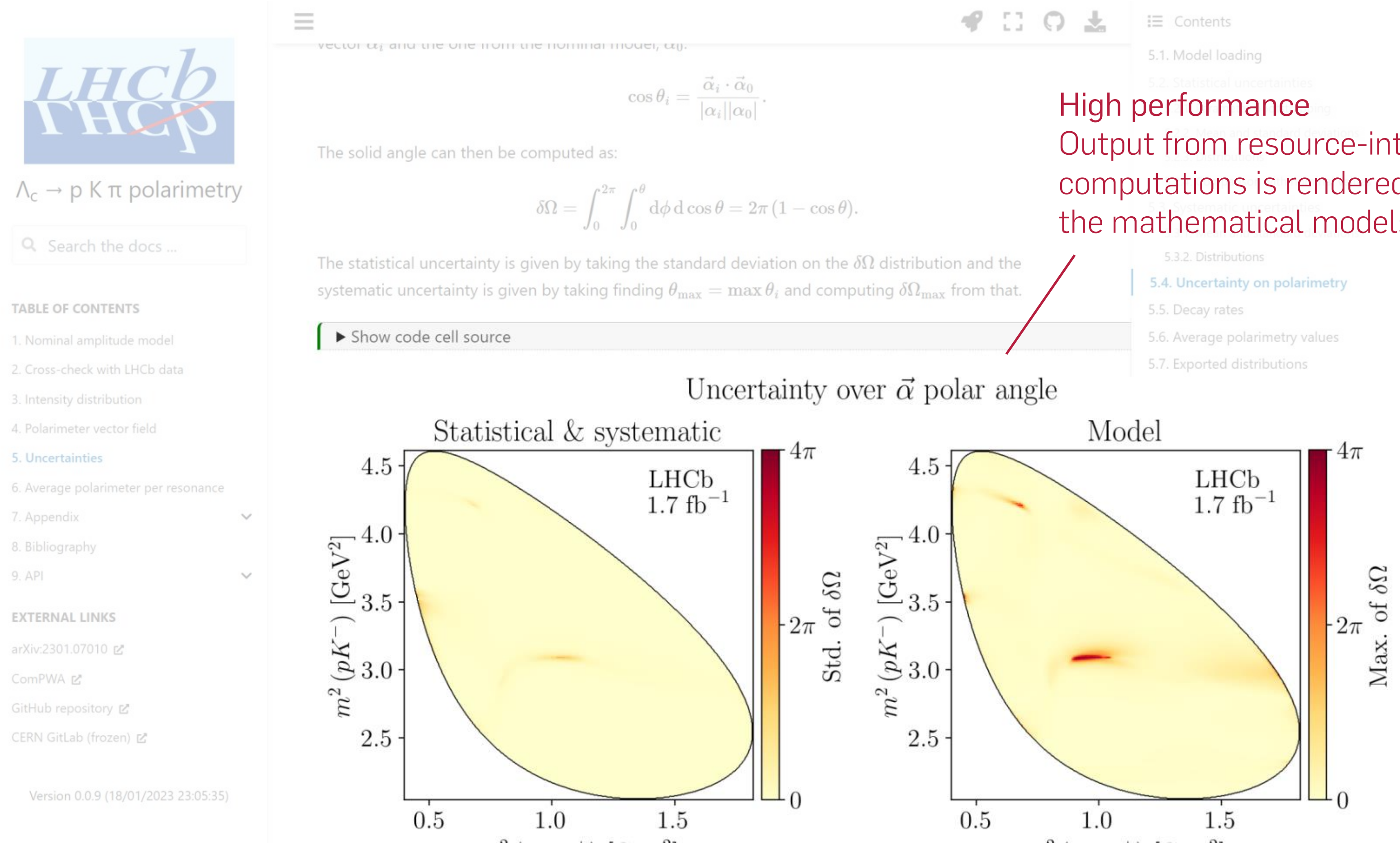
▶ Show code cell source

$$e^{-\alpha q_{m_0,m_1}(s)^2} \mathcal{R}_\mathrm{Bugg}\left(m_{K\pi}^2\right)$$

Mathematical expressions are **automatic rendering** of the implemented amplitude models

lc2pkpi-polarimetry.docs.cern.ch

Self-documenting workflow



High performance

Output from resource-intensive computations is rendered alongside the mathematical models

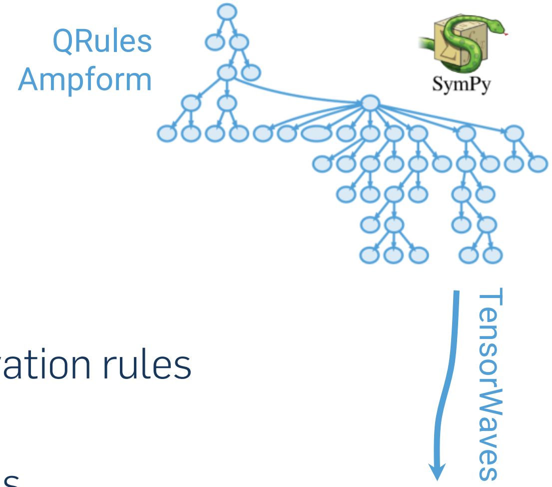
Layered software development

- CAS allows us to **separate physics from number crunching**
- Symbolic expressions become a **Single Source of Truth** for physics implementations
- Model building through layers of configurability and generalization
  1. Build up symbolic models directly in a script
  2. Generalize model building with functions and classes
  3. Project evolves into generalized library
- Result: **grow a self-documenting collection of tools** for amplitude model building

1.
```python
import sympy as sp
N, s, m0, w0 = sp.symbols("N s m0 Gamma0")
N / (m0**2 - sp.I * m0 * w0 - s)
```

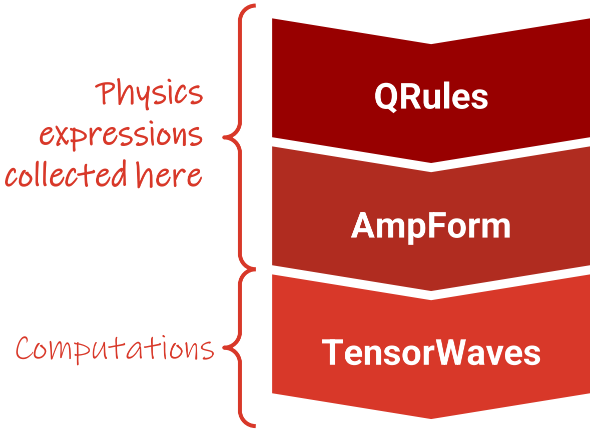
2.
```python
builder = ampform.get_builder(reaction)
for particle in reaction.get_intermediate_particles():
    builder.dynamics.assign(particle.name,
create_relativistic_breit_wigner)
model = builder.formulate()
```

Proof of concept | the ComPWA project



Common Partial Wave Analysis

Three main Python packages that together cover a full amplitude analysis:



Automated quantum number conservation rules

Formulate symbolic amplitude models

Fit models to data and generate data samples with multiple computational back-ends

All are designed as **libraries**, so they can be used by other packages by installing through pip or Conda

Topics to discuss

- **Main ideas presented:**
  - CAS-assisted model building:
    symbolic expressions as template to numerical functions
  - Amplitude analysis **documentation**
  - Improved **modularity** and improved interoperability
- **Related topics:**
  - Fit **performance** with JAX, TensorFlow, Numba, etc. on different devices
  - Effect and useability of **autodiff** for amplitude models
  - Standardization and **serialization** of amplitude models like HS3
    (reproducibility, metadata etc.?)