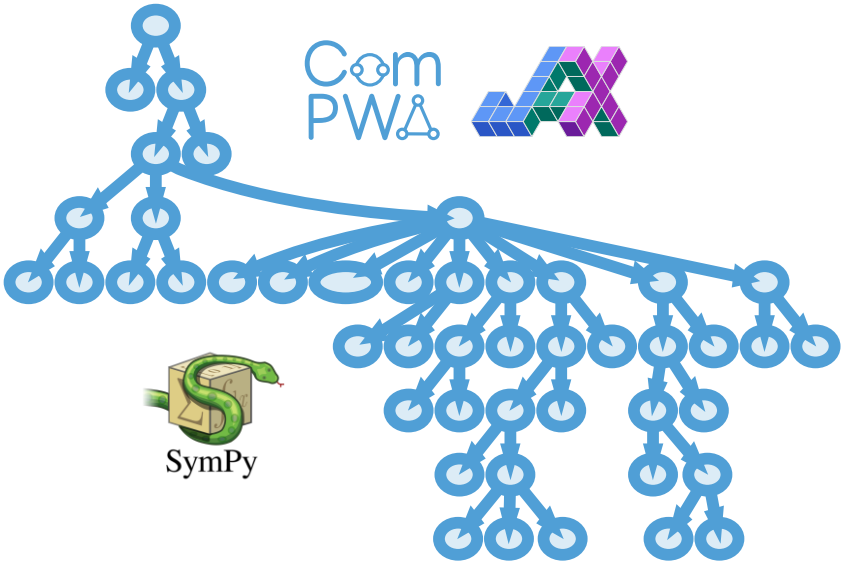
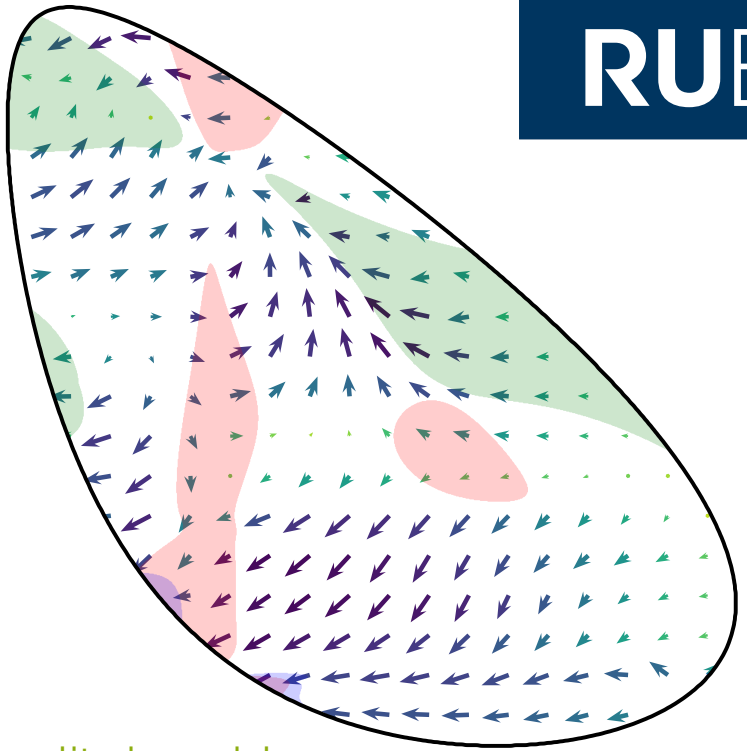


RUB

RUHR-UNIVERSITÄT BOCHUM

# The ComPWA project

## Computing polarimeter vector fields with symbolic amplitude models

7 July 2023 HADRON2023, Genova

**Remco de Boer**, Miriam Fritsch, Stefan Pflüger, Wolfgang Gradl (JGU Mainz),
Mikhail Mikhasenko (LHCb, for polarimetry)

# Overview

1. Intro: polarimeter vector field by LHCb (arXiv:2301.07010, JHEP)
2. Amplitude analysis with symbolic expressions
   - Computational backends from Machine Learning community
   - Inserting a Computer Algebra System
3. Self-documenting workflow in action
4. Layered software design with the ComPWA project

# What can we learn by measuring polarization of hadrons?

*Polarization = preferred orientation of the spin of a particle in space*

- Mechanism of quark hadronization [Brambilla:2010cs, Faccioli:2010kd, Butenschoen:2012px]
- BSM searches with $\Lambda_b^0 \to \Lambda_c^+ \ell^- \nu$
  E.g. sign of longitudinal polarization of $\Lambda_c^+$ provides a test for left-handedness of $b \to c$ current
  [Konig:1993wz, Dutta:2015ueb, Shivashankara:2015cta, Li:2016pdv, Datta:2017aue, Ray:2018hrx, DiSalvo:2018ngq, Penalva:2019rgt, Ferrillo:2019owd]
- BSM searches with measurement of EDM/MDM with charmed mesons
  [Baryshevsky:2016cul, Botella:2016ksl, Fomin:2017ltw]
- Hadron spectroscopy, extending decay chains [backup]
  - ✓ $\Lambda_b^0 \to J/\psi p K$ with $J/\psi \to \mu^+\mu^-$
  - ✓ $\mathcal{B} \to J/\psi \bar{p} \Lambda$ with $\Lambda \to p\pi^-$

  } two-body → scalar asymmetry parameter $\alpha$

  $$\frac{2}{\Gamma}\frac{\mathrm{d}\Gamma}{\mathrm{d}\cos\theta} = 1 + P\alpha\cos\theta$$

  - ? $\mathcal{B}^+ \to \Lambda_c^+ \bar{\Lambda}_c^- K^+$ with $\Lambda_c^+ \to p K^- \pi^+$
  - ? $\Omega_b^- \to \Xi_c^+ \pi^- K^-$ with $\Xi_c^+ \to p K^- \pi^+$

  } three-body → polarimeter vector field $\vec{\alpha}$

# Trick: Dalitz-plot decomposition

Factorization of variables describing dynamics and polarization [JPAC:2019ufm]:

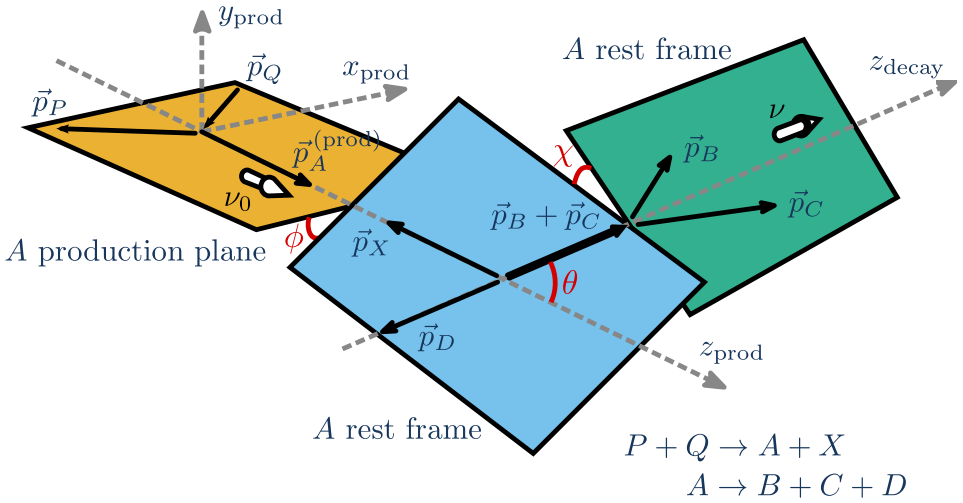
$$T_{\nu_0,\{\lambda\}}(\phi,\theta,\chi;\kappa) = \sum_{\nu} D^{1/2}_{\nu_0,\nu}(\phi,\theta,\chi)\, A_{\nu,\{\lambda\}}(\kappa)$$

**Polarization d.o.f.**

- Euler angles in active ZYZ convention
- rotation of the system as rigid body
- polarization affects angular distribution

**Dynamic d.o.f.**

- Mandelstam variables of the subsystems
- describes resonances in the decay



# Model-agnostic representation of the decay rate

Using the SU(2) → SO(3) homomorphism [backup], we get our polarized **master formula**,

$$|\mathcal{M}(\phi,\theta,\chi,\kappa)|^2 = I_0(\kappa)\left(1+\sum_{i,j=1}^{3} P_i R_{ij}(\phi,\theta,\chi)\alpha_j(\kappa)\right)$$

This study

where

- $I_0(\kappa)$ is the unpolarised intensity
- $R(\phi,\theta,\chi) = R_z(\phi)R_y(\theta)R_z(\chi)$ defines the decay plane orientation.
- $\vec{\alpha}(\kappa)$ is the aligned polarimeter vector field,

$$\vec{\alpha}(\kappa) = \sum_{\nu',\nu,\{\lambda\}} A^*_{\nu',\{\lambda\}} \vec{\sigma}_{\nu',\nu} A_{\nu,\{\lambda\}} \,/\, I_0(\kappa)$$

It is specific for the decay and *does not depend on the production mechanism*.

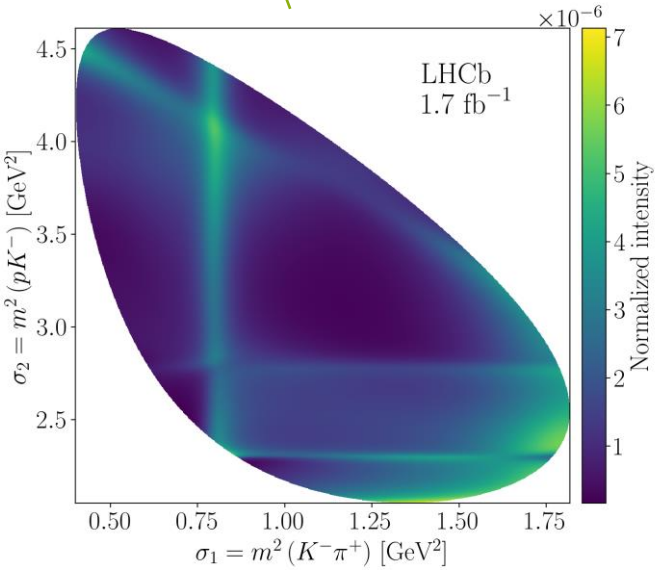
**Determining polarization**

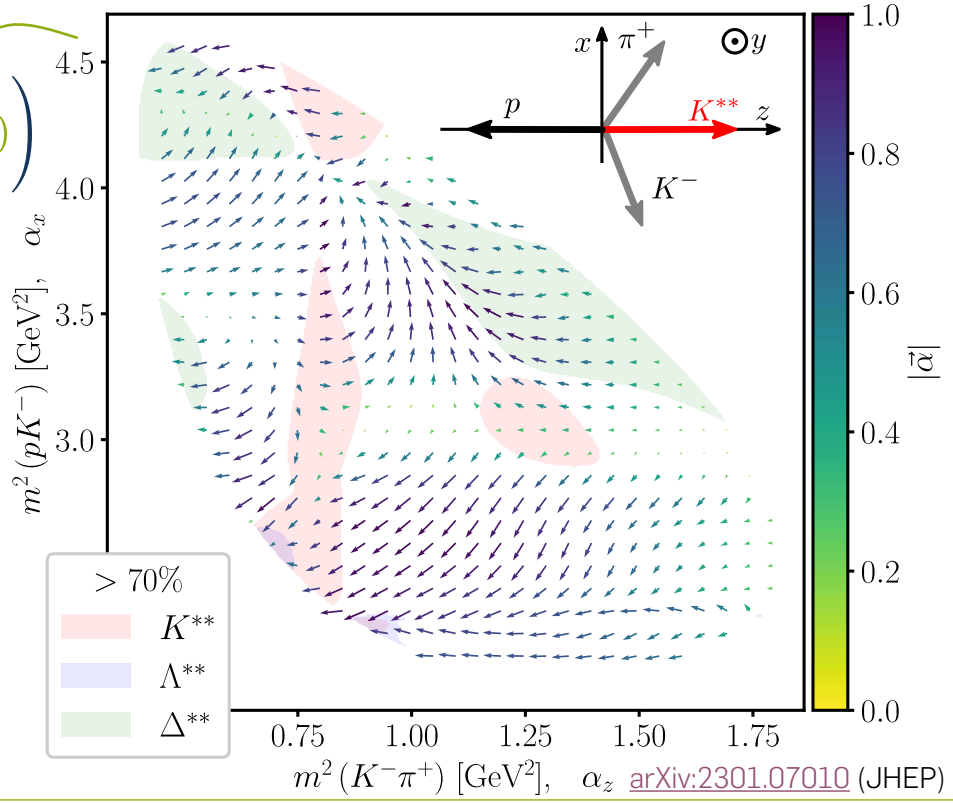
1. provide $I_0(\kappa)$ and $\vec{\alpha}(\kappa)$
2. measure $|\mathcal{M}(\phi,\theta,\chi,\kappa)|^2$
3. get $\vec{P}$ from fit

# Polarimetry study | analysis output for $\Lambda_c^+ \to pK^-\pi^+$

Parameter input: arXiv:2208.03262

$$|\mathcal{M}(\phi,\theta,\chi,\kappa)|^2 = I_0(\kappa)\left(1+\sum_{i,j=1}^{3} P_i R_{ij}(\phi,\theta,\chi)\alpha_j(\kappa)\right)$$



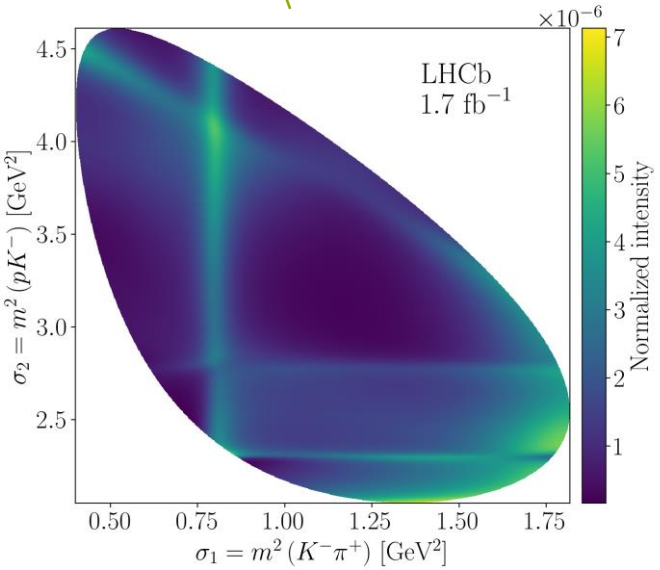



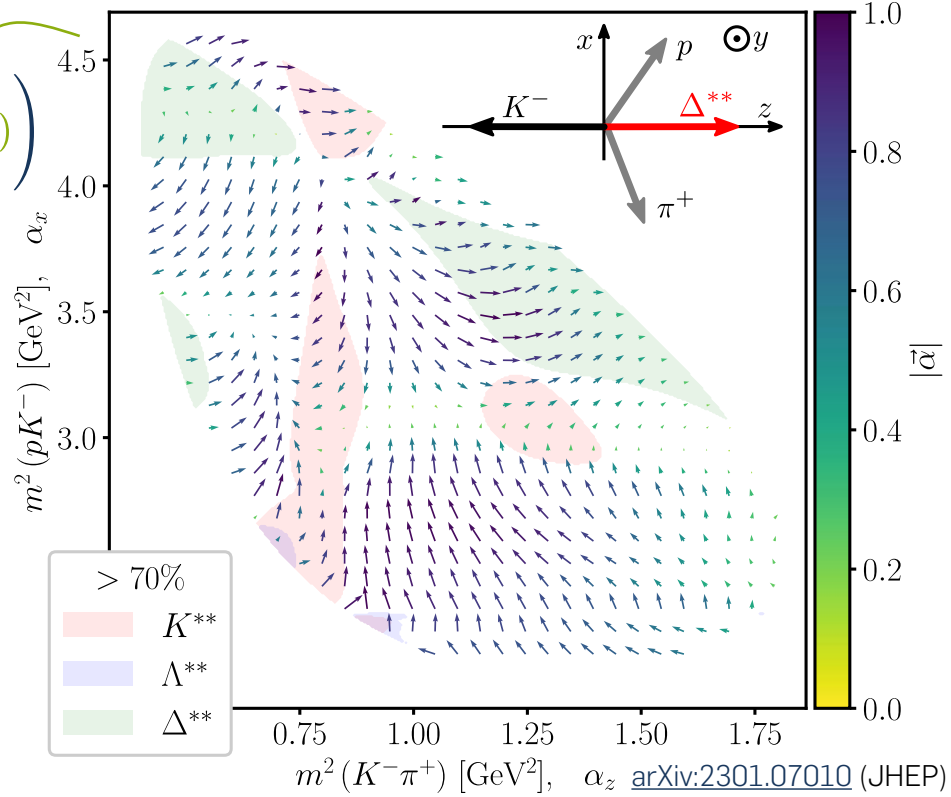

arXiv:2301.07010 (JHEP)

# Polarimetry study | analysis output for $\Lambda_c^+ \to pK^-\pi^+$

Parameter input: arXiv:2208.03262

$$|\mathcal{M}(\phi,\theta,\chi,\kappa)|^2 = I_0(\kappa)\left(1+\sum_{i,j=1}^{3} P_i R_{ij}(\phi,\theta,\chi)\alpha_j(\kappa)\right)$$

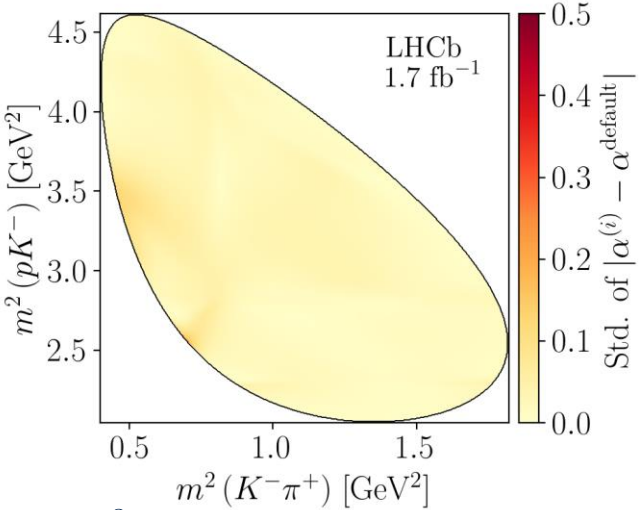


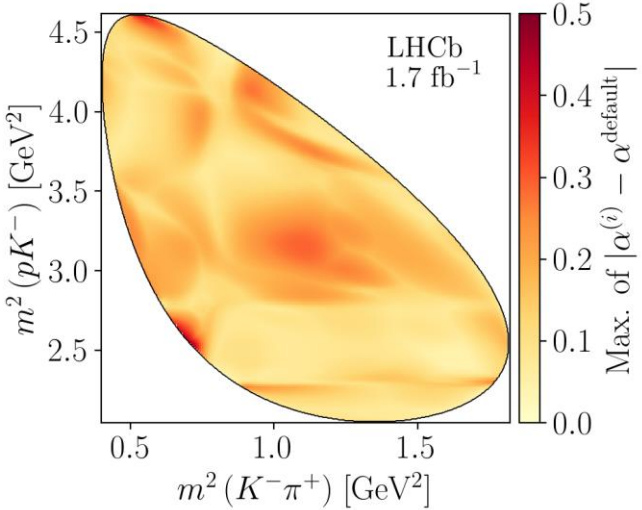




arXiv:2301.07010 (JHEP)

# Polarimetry study | analysis output for $\Lambda_c^+ \to pK^-\pi^+$

- Syst. and stat. uncertainties
- Averaged polarimeter vector
- Details presented elsewhere
- For **reuse in other analyses**:
  - serialised vector fields
  - symbolic transition amplitudes
  - tool to formulate symbolic vector fields for other channels




$$
\begin{aligned}
\overline{\alpha_x} &= \left(-62.6 \pm 4.5^{+8.4}_{-14.8}\right) \times 10^{-3} \\
\alpha_y &= \left(+8.9 \pm 8.9^{+9.1}_{-12.7}\right) \times 10^{-3} \\
\alpha_z &= \left(-278.0 \pm 23.7^{+12.6}_{-40.4}\right) \times 10^{-3} \\
|\alpha| &= \left(669.4 \pm 9.3^{+15.3}_{-10.4}\right) \times 10^{-3}
\end{aligned}
$$

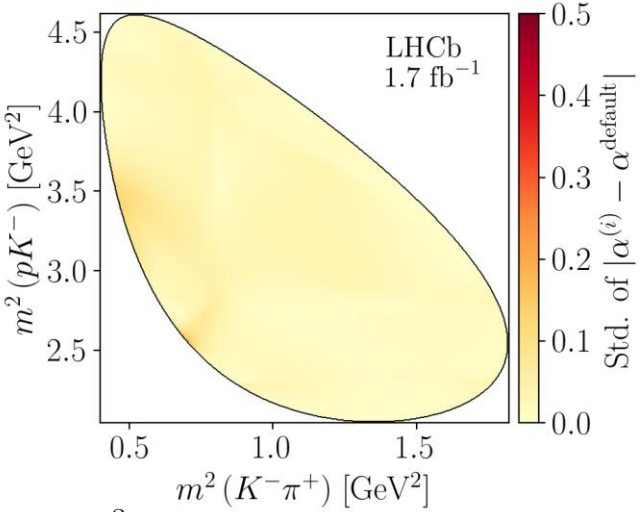
$$
|\mathcal{M}(\phi,\theta,\chi,\kappa)|^2 = I_0(\kappa)\left(1 + \sum_{i,j=1}^{3} P_i R_{ij}(\phi,\theta,\chi)\alpha_j(\kappa)\right)
$$

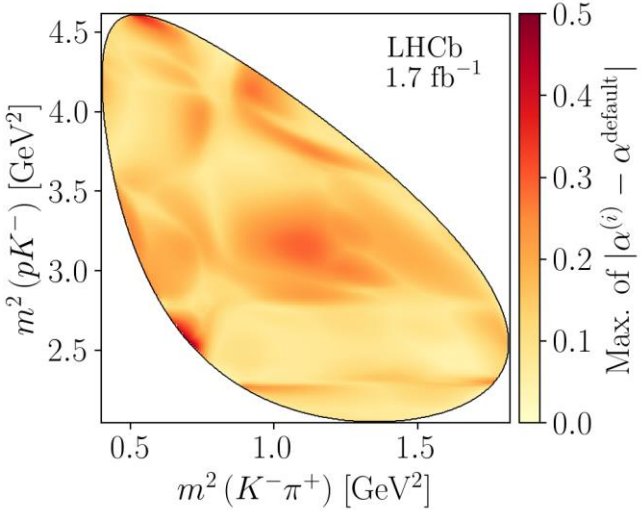
$$
\vec{\alpha}(\kappa) = \sum_{\nu',\nu,\{\lambda\}} A^*_{\nu',\{\lambda\}} \vec{\sigma}_{\nu',\nu} A_{\nu,\{\lambda\}} \,/\, I_0(\kappa)
$$

arXiv:2301.07010 (JHEP)

# Polarimetry study | analysis output for $\Lambda_c^+ \to pK^-\pi^+$

- Syst. and stat. uncertainties
- Averaged polarimeter vector
- Details presented elsewhere
- For **reuse in other analyses**:
  - serialised vector fields
  - symbolic transition amplitudes
  - tool to formulate symbolic vector fields for other channels




$$
\begin{aligned}
\overline{\alpha_x} &= \left(-62.6 \pm 4.5^{+8.4}_{-14.8}\right) \times 10^{-3} \\
\alpha_y &= \left(+8.9 \pm 8.9^{+9.1}_{-12.7}\right) \times 10^{-3} \\
\alpha_z &= \left(-278.0 \pm 23.7^{+12.6}_{-40.4}\right) \times 10^{-3} \\
|\alpha| &= \left(669.4 \pm 9.3^{+15.3}_{-10.4}\right) \times 10^{-3}
\end{aligned}
$$

Main topic of this talk

$$
|\mathcal{M}(\phi,\theta,\chi,\kappa)|^2 = I_0(\kappa)\left(1 + \sum_{i,j=1}^{3} P_i R_{ij}(\phi,\theta,\chi)\alpha_j(\kappa)\right)
$$

$$
\vec{\alpha}(\kappa) = \sum_{\nu',\nu,\{\lambda\}} A^*_{\nu',\{\lambda\}} \vec{\sigma}_{\nu',\nu} A_{\nu,\{\lambda\}} \,/\, I_0(\kappa)
$$

arXiv:2301.07010 (JHEP)

# Amplitude analysis software

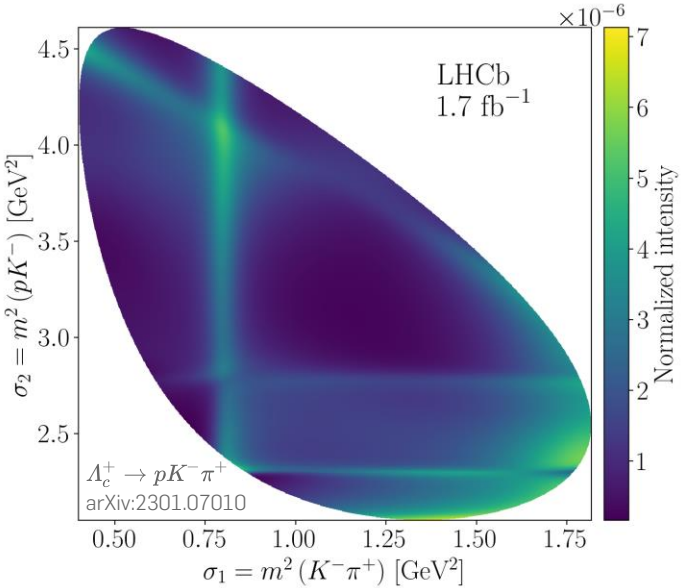
fast computations

Larger perspective: what makes PWA so challenging?

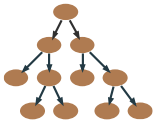
- Unbinned, multidimensional problem set
- Complicated parametrizations and estimators
  - need to quickly try out different parameterizations
  - fits can take several weeks
- Theory is hard to get into
- Relatively small community (but growing interest!)

flexibility

documentation



# Mission: bring code closer to theory



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


Flexibility through a **Computer Algebra System**

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, ...

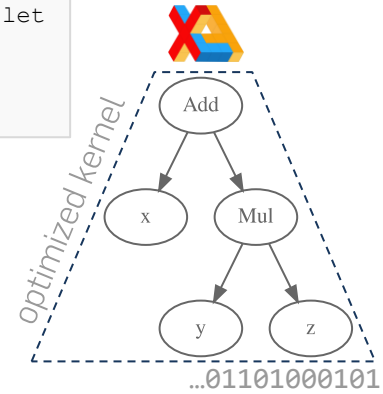
```
@tf.function(jit_compile=True)
def my_expression(x, y, z):
    return x + y * z
```

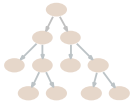
Converted to device-agnostic XLA code

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


```
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





# 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, ...


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

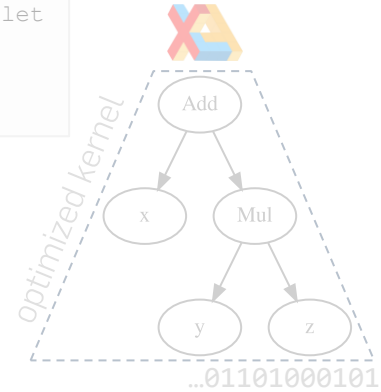
Heavy lifting by optimized backend

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

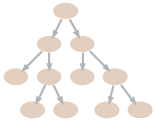
Converted to device-agnostic XLA code

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,) }
```



# Mission: bring code closer to theory



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


Flexibility through a **Computer Algebra System**

Academic continuity through **living documentation**


# 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)
```


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

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

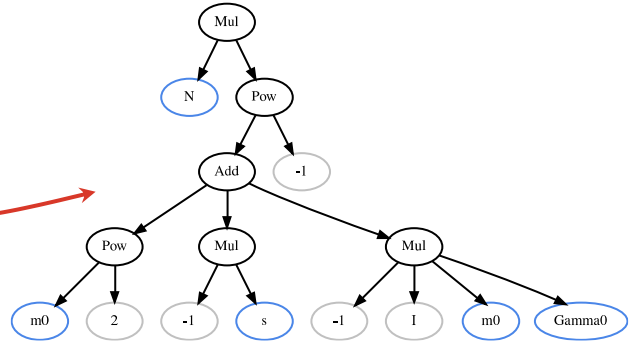
- Transparency: inspect the math as you formulate the model
- Flexibility: modify the model with analytic substitutions
- Performance: simplify expressions algebraically
- Code generation: symbolic model as template to computational back-ends (SSoT)

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

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

SymPy

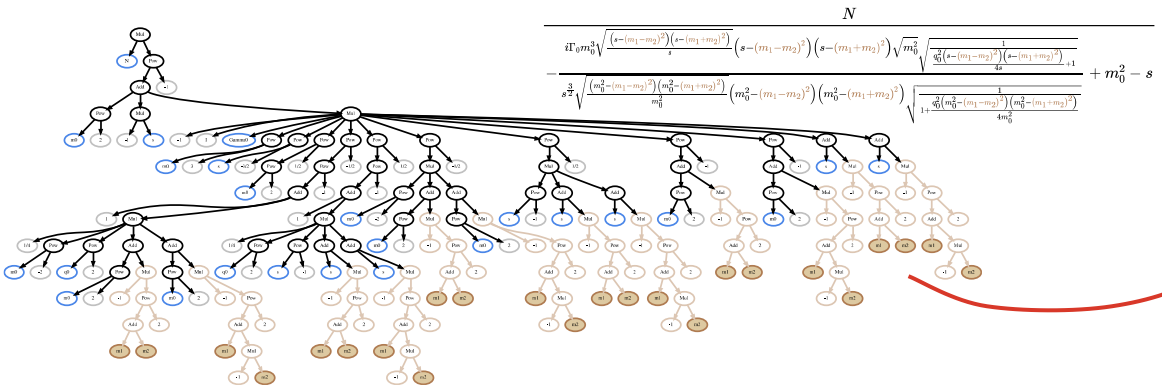
CAS represents expression as a tree

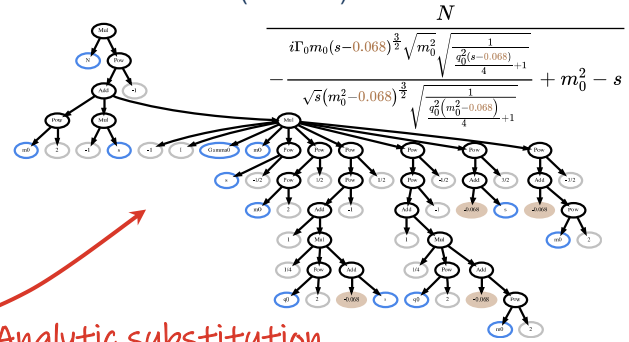


# Symbolic amplitude models

**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)





Analytic substitution and simplification

# 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)

```
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
```

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)

```
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

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

# Symbolic amplitude models

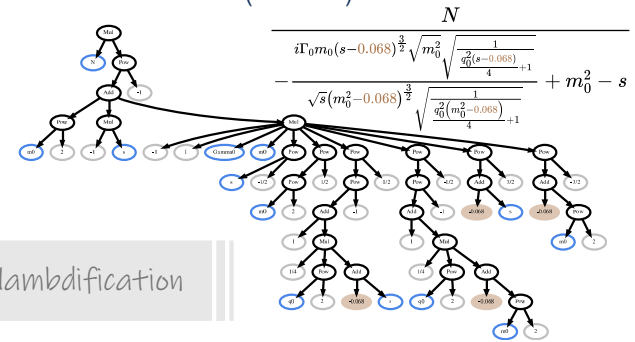
**A new technique**: formulate your amplitude model with a Computer Algebra System

- 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)

```
// 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);
```

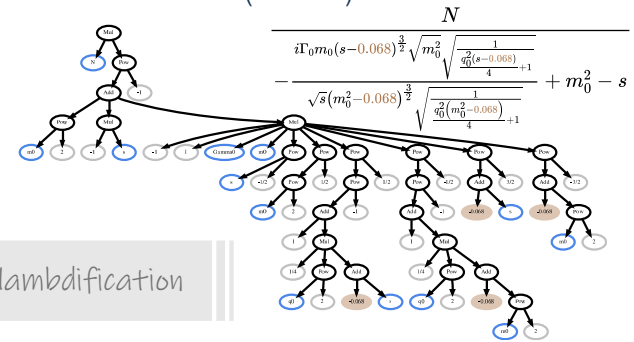


# Symbolic amplitude models

**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
    )
```



# Symbolic amplitude models

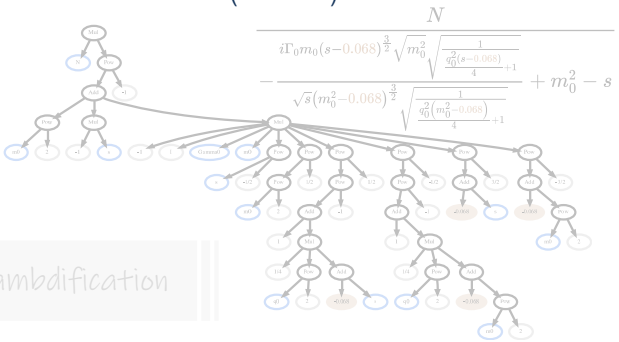
**A new technique**: formulate your amplitude model with a Computer Algebra System

- 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)

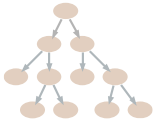
```
@jax.jit
def _lambdifygenerated(Gamma0, N, m0, s):
    return N / (
        -1j
        * Gamma0
        * m0
        * ((1 / 4) * m0**2 + 0.9831)
        * (s - 0.0676) ** (3 / 2)
        * sqrt(m0**2)
        / (sqrt(s) * (m0**2 - 0.0676) ** (3 / 2) * ((1 / 4) * s + 0.9831))
        + m0**2
  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

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

SymPy lambdification



# Mission: bring code closer to theory



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


Flexibility through a **Computer Algebra System**

Academic continuity through **living documentation**

lc2pkpi-polarimetry.docs.cern.ch

# Self-documenting workflow in action


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

Search the docs ...

TABLE OF CONTENTS

1. Nominal amplitude model
2. Cross-check with LHCb data
3. Intensity distribution
4. Polarimeter vector field
5. Uncertainties
6. Average polarimeter per resonance
7. Appendix
8. Bibliography
9. API

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

**$\Lambda_c^+$ 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 $\Lambda_c^+$ 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 $\Lambda_c^+$.

**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**

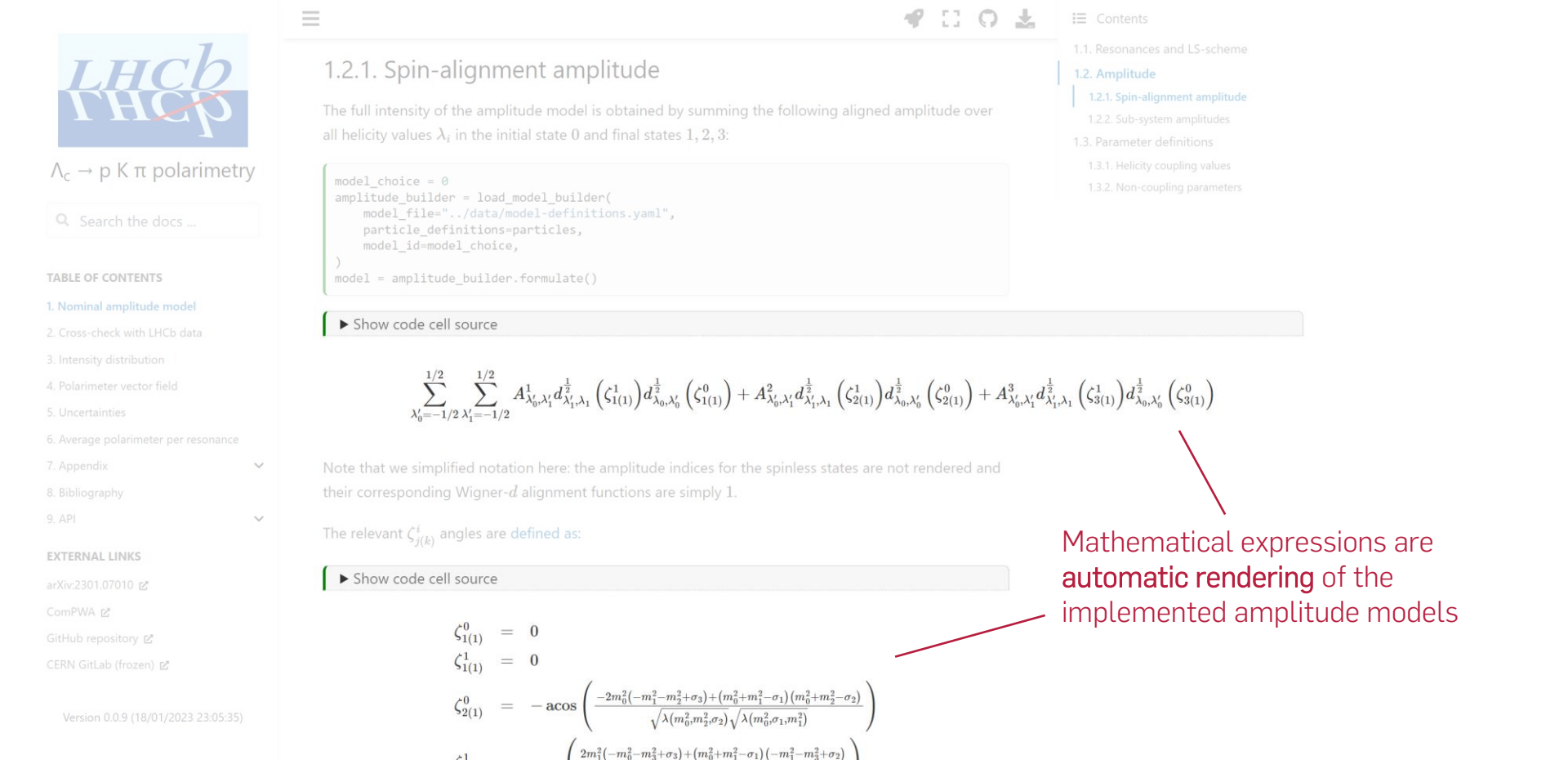
This website shows all analysis results that led to the publication of LHCb-PAPER-2022-044. More information on this publication can be found on the following pages:

Polarimetry study brings it all together:

- 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

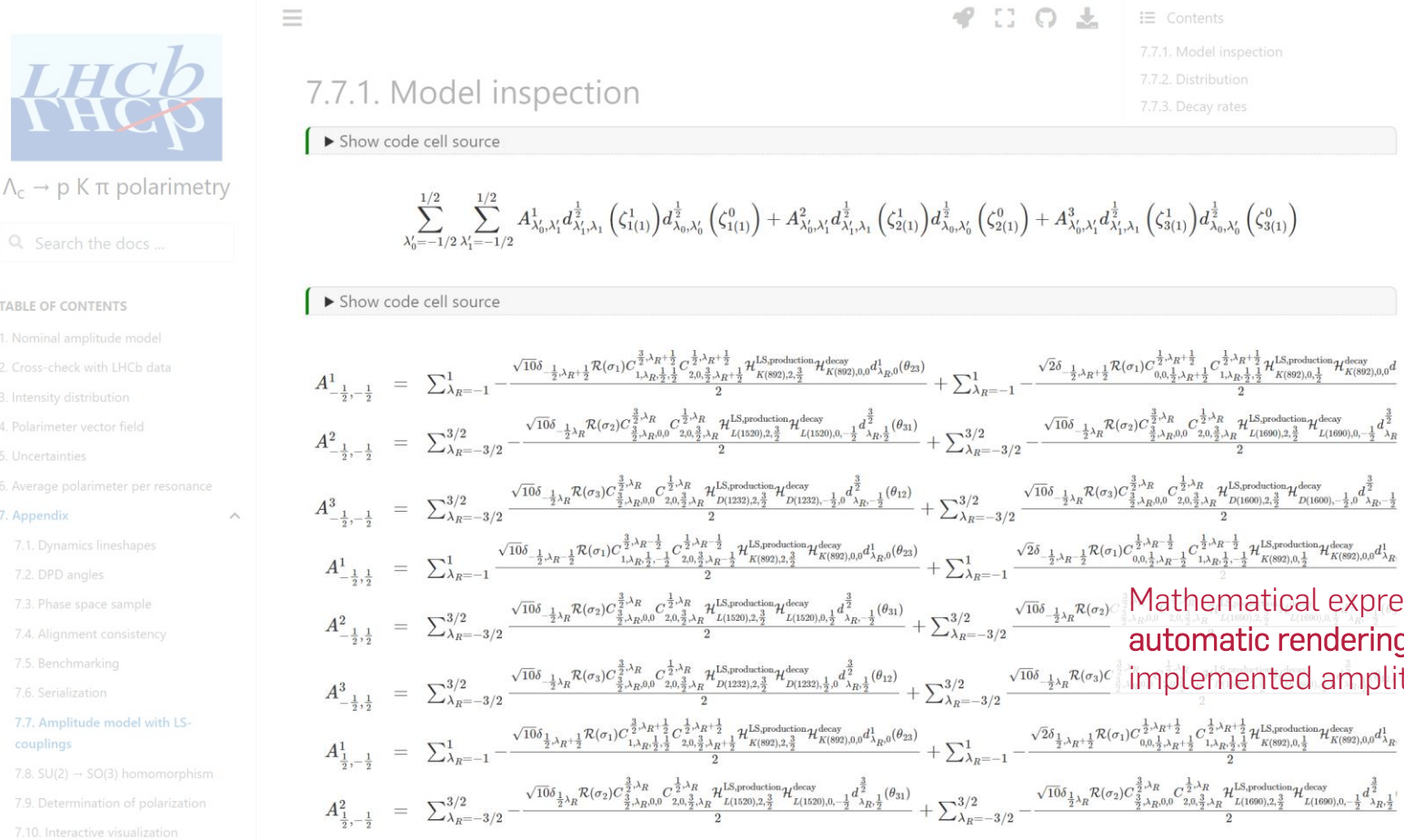
lc2pkpi-polarimetry.docs.cern.ch

# Self-documenting workflow in action




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

# Self-documenting workflow in action




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

# Self-documenting workflow in action









## 7.1.1. Relativistic Breit-Wigner

▶ Show code cell source

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

## 7.1.2. Bugg Breit-Wigner

▶ Show code cell source

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

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

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

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

▶ Show code cell source

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

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

# Self-documenting workflow in action



$$\cos\theta_i = \frac{\vec{\alpha}_i \cdot \vec{\alpha}_0}{|\alpha_i||\alpha_0|}.$$

The solid angle can then be computed as:

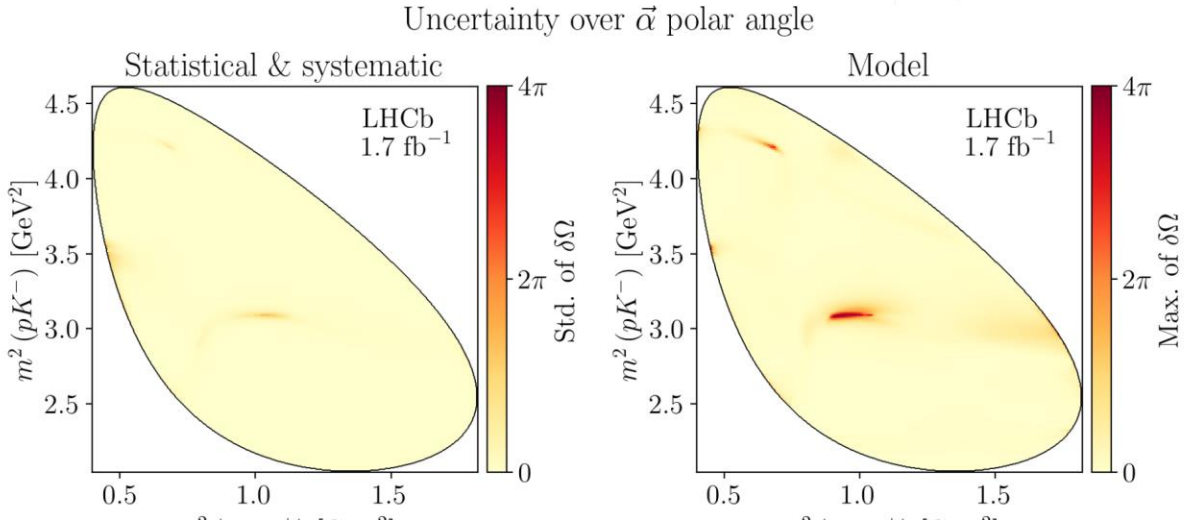
$$\delta\Omega = \int_0^{2\pi}\int_0^{\theta} \mathrm{d}\phi\,\mathrm{d}\cos\theta = 2\pi\,(1-\cos\theta).$$

The statistical uncertainty is given by taking the standard deviation on the $\delta\Omega$ distribution and the systematic uncertainty is given by taking finding $\theta_{\max} = \max\theta_i$ and computing $\delta\Omega_{\max}$ from that.

▶ Show code cell source

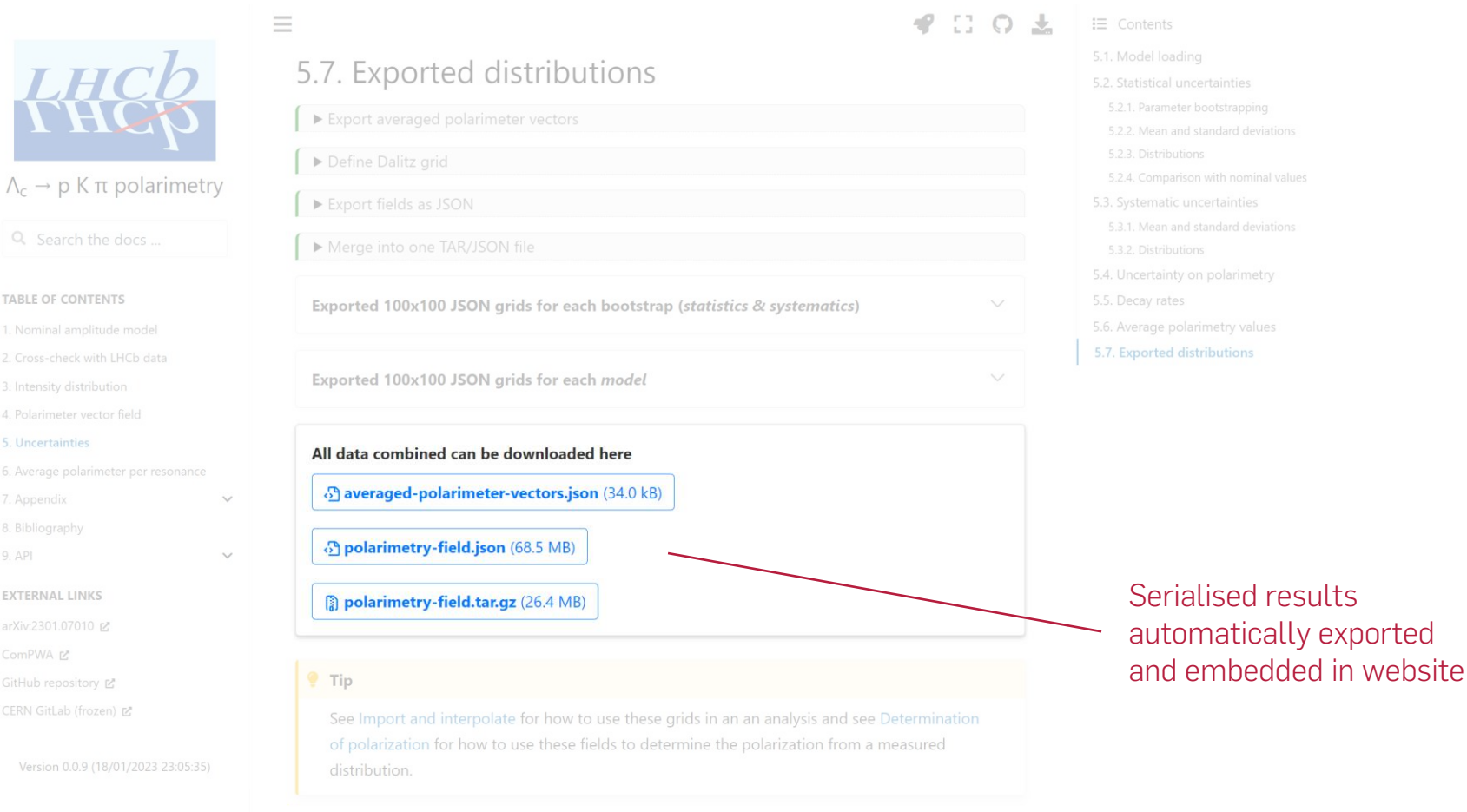


High performance
Output from resource-intensive computations is rendered alongside the mathematical models



lc2pkpi-polarimetry.docs.cern.ch

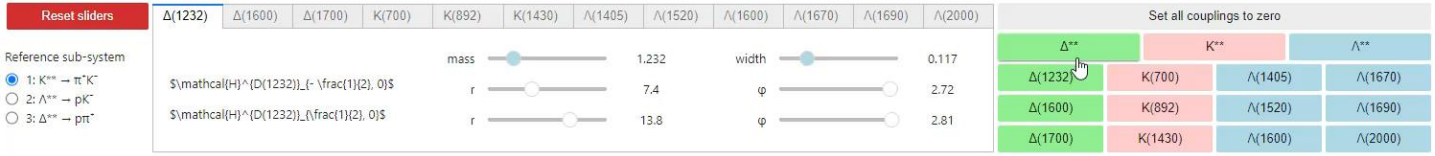
# Self-documenting workflow in action

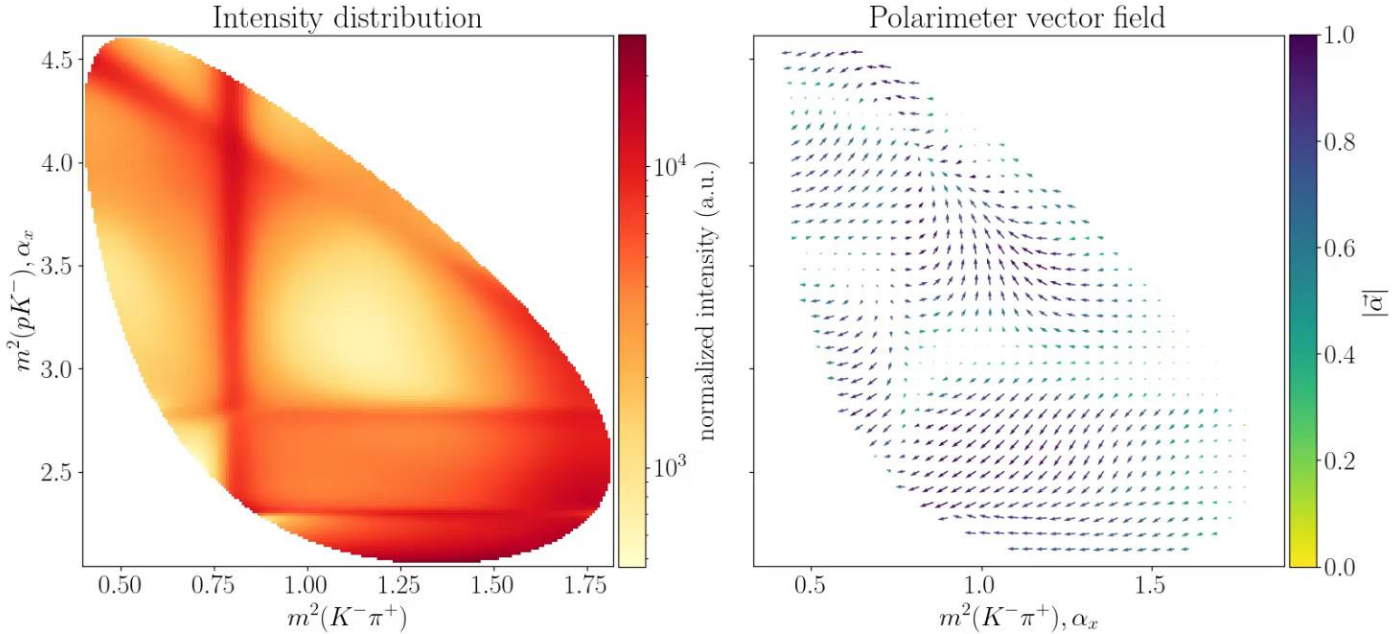



Serialised results automatically exported and embedded in website

lc2pkpi-polarimetry.docs.cern.ch

# Performance demo | interactive widget



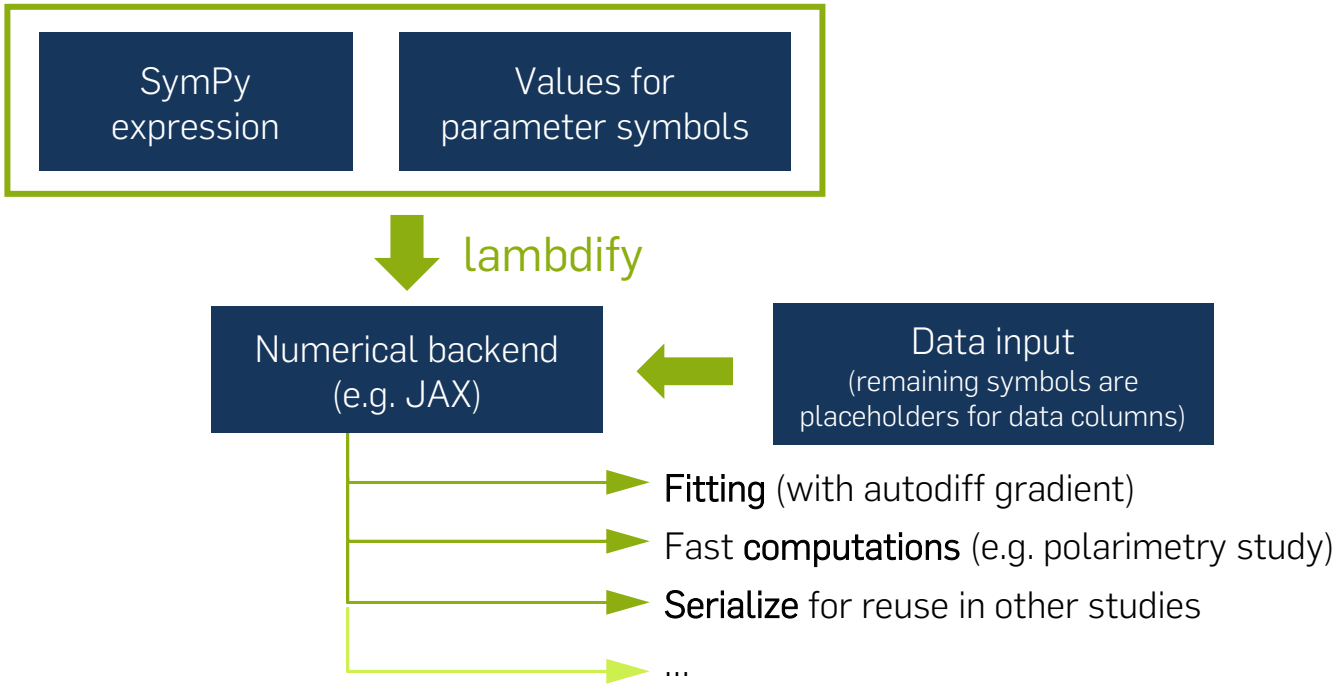




[link to video]

High performance
Intensity and vector field are computed upon each modification to the sliders (recorded on laptop)

# Physics separated from the ‘number crunching’

## Communicate the model



# Physics separated from the ‘number crunching’

## Communicate the model


Focus our efforts into formulating physics models...

lambdify

Numerical backend (e.g. JAX)

Data input (remaining symbols are placeholders for data columns)

- **Fitting** (with autodiff gradient)
- Fast **computations** (e.g. polarimetry study)
- **Serialize** for reuse in other studies
- ...

Recent examples [WIP]

- Matrix elements with autodiff
- arXiv:2302.07665

...and preserving them automatically

# Layered software development

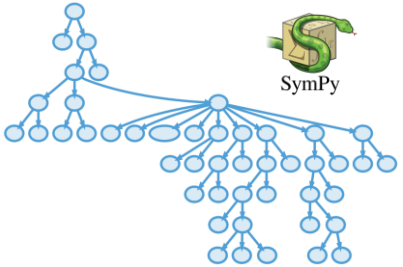
- CAS allows us to **separate physics from number crunching**
- Symbolic expressions become a **Single Source of Truth** for physics implementations
- Model building through layers of configurability and generalization
  1. Build up symbolic models directly in a script
  2. Generalize model building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

Polarimetry project can also formulate models for other channels

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

2.
```
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:

Physics expressions collected here

Computations

**QRules** — Automated quantum number conservation rules

**AmpForm** — Formulate symbolic amplitude models

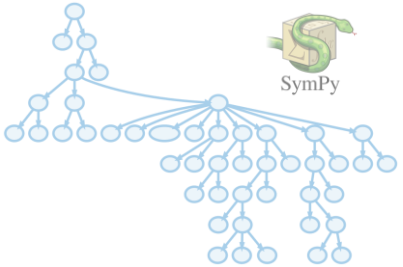
**TensorWaves** — Fit models to data and generate data samples with multiple computational back-ends

TensorWaves

Demo in backup slides

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

# Proof of concept | the ComPWA project




Common Partial Wave Analysis

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

Physics expressions collected here

Computations

**QRules**

Automated quantum number conservation rules

**AmpForm**

Formulate symbolic amplitude models

**TensorWaves**

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

TensorWaves

Demo in Google Colab

All are designed as **libraries**, so they can be used by other packages

by installing through pip or Conda

Thank you for your attention!

# Summary

- New results: polarimeter vector field
  - Increased sensitivity for polarisation studies
  - Extend amplitude models
  - Serialised fields and model building available for reuse
- New technique: separate physics from number crunching
  - High performance with computational backends from the ML community
  - Flexibility and transparency with a CAS
- Polarimetry project proves that symbolic expressions
  - result in a self-documenting workflow
  - allow building up physics models through layered software development
- Proof-of-concept project: ComPWA

# Application: extending amplitude models

Normally, without polarization taken into account:

$$I\left(m^2_{\Lambda_c^+ K^+}, m^2_{\bar{\Lambda}_c^- K^+}\right) = \sum_{\nu_0,\bar{\nu}_0} \left| O^B_{\nu_0,\bar{\nu}_0}(m^2_{\Lambda_c^+ K^+}, m^2_{\bar{\Lambda}_c^- K^+}) \right|^2$$

Polarimeter provides **10 additional degrees of freedom**:

$$I\left(m^2_{\Lambda_c^+ K^+}, m^2_{\bar{\Lambda}_c^- K^+}; \phi, \theta, \chi, m^2_{pK^-}, m^2_{K^-\pi^+}; \bar{\phi}, \bar{\theta}, \bar{\chi}, m^2_{\bar{p}K^+}, m^2_{K^+\pi^-}\right) =$$

$$\sum_{\nu_0,\bar{\nu}_0,\nu,\bar{\nu}} \sum_{\nu_0',\bar{\nu}_0',\nu',\bar{\nu}'} O^{B*}_{\nu_0',\bar{\nu}_0'}(m^2_{\Lambda_c^+ K^+}, m^2_{\bar{\Lambda}_c^- K^+}) O^B_{\nu_0,\bar{\nu}_0}(m^2_{\Lambda_c^+ K^+}, m^2_{\bar{\Lambda}_c^- K^+})$$

$$\times D^{1/2*}_{\nu_0',\nu'}(\phi,\theta,\chi) D^{1/2}_{\nu_0,\nu}(\phi,\theta,\chi) X^{\Lambda_c^+}_{\nu',\nu}(m^2_{pK^-}, m^2_{K^-\pi^+})$$

$$\times D^{1/2*}_{\bar{\nu}_0',\bar{\nu}'}(\bar{\phi},\bar{\theta},\bar{\chi}) D^{1/2}_{\bar{\nu}_0,\bar{\nu}}(\bar{\phi},\bar{\theta},\bar{\chi}) X^{\bar{\Lambda}_c^-}_{\bar{\nu}',\bar{\nu}}(m^2_{\bar{p}K^+}, m^2_{K^+\pi^-})$$

**Physics motivation**

- exotic structures in $\Lambda_c^+\bar{\Lambda}_{\bar{c}}$
- studies of $\Sigma_c^{**}$ in $\Lambda_c^+ K$

[BaBar:2007xtc, Belle:2017jrt, Belle:2018yob, LHCb:2022vns]

No need to fit these d.o.f.

$$X_{\nu',\nu}(\kappa) = \frac{I_0(\kappa)}{2}\left(1 + \vec{\alpha}(\kappa)\cdot\vec{\sigma}^P\right)_{\nu',\nu}$$

# SU(2) → SO(3) homomorphism

The **polarized decay rate** is given by:

$$|\mathcal{M}|^2 = \sum_{\nu_0,\nu_0',\{\lambda\}} \mathfrak{R}_{\nu_0',\nu_0} T^*_{\nu_0',\{\lambda\}} T_{\nu_0,\{\lambda\}}, \qquad \mathfrak{R}_{\nu_0',\nu_0} = \frac{1}{2}\left(1 + \vec{P}\cdot\vec{\sigma}^P\right)_{\nu_0',\nu_0}$$

$$T_{\nu_0,\{\lambda\}}(\phi,\theta,\chi;\kappa) = \sum_{\nu} D^{1/2}_{\nu_0,\nu}(\phi,\theta,\chi)\, A_{\nu,\{\lambda\}}(\kappa)$$

Trick for factoring out polarization [Cornwell:1997ke]

SU(2) is double cover of SO(3) — spin $1/2$, $4\pi$ rotation

Explicit homomorphism with the non-trivial centre:

$$\phi : \mathrm{SU}(2) \to \mathrm{SO}(3), \qquad \phi(d) = R\,,$$

$$\phi(d) = \frac{1}{2}\mathrm{Tr}\left[D^{1/2*}(\phi,\theta,\chi)\sigma_i^P D^{1/2}(\phi,\theta,\chi)\sigma_j^P\right] = R_{ij}(\phi,\theta,\chi)$$

# Back-up

## The main ComPWA packages

```
pip install grules
pip install ampform
pip install tensorwaves
```

# QRules

## Quantum number conservation rules

**Core**: 'search engine' for quantum numbers

Get particle properties:*

```
PDG = qrules.load_pdg()
PDG.find("a(2)(1320)0")
```

```
Particle(
  name='a(2)(1320)0',
  pid=115,
  latex='a_{2}(1320)^{0}',
  spin=2.0,
  mass=1.3182,
  width=0.107,
  isospin=Spin(1, 0),
  parity=+1,
  c_parity=+1,
  g_parity=-1,
)
```

Find particles by quantum number:

```
selection = PDG.filter(
    lambda p: p.mass > 2.8
    and p.spin > 0
    and p.charge
    and p.charmness
    and p.parity == +1
)
selection.names
```

```
['Lambda(c)(2880)+', 'Xi(c)(2815)~-']
```

Check which conservation rules are violated:

```
qrules.check_reaction_violations(
    initial_state="pi0",
    final_state=["gamma", "gamma", "gamma"],
)
```

launch binder Open in Colab

```
{frozenset({'c_parity_conservation'})}
```

Also a library of conservation rules

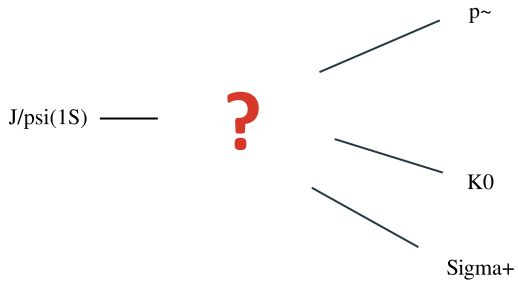
*PDG info computed from the scikit-hep particle package

# QRules

## Quantum number conservation rules

**PWA use case**: compute which particle reactions are allowed between a given initial and final state

1. User specifies some boundary conditions
   (particle names, allowed interactions, isobar model, etc.)

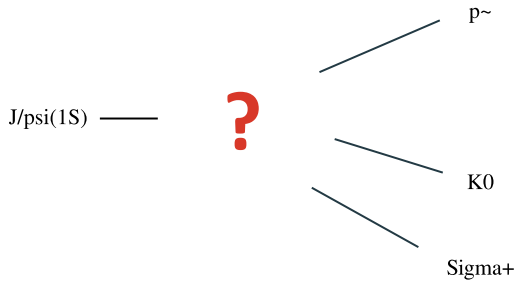


# QRules

## Quantum number conservation rules

**PWA use case**: compute which particle reactions are allowed between a given initial and final state

1. User specifies some boundary conditions
   (particle names, allowed interactions, isobar model, etc.)
2. QRules then:
   - determines all possible decay topologies,

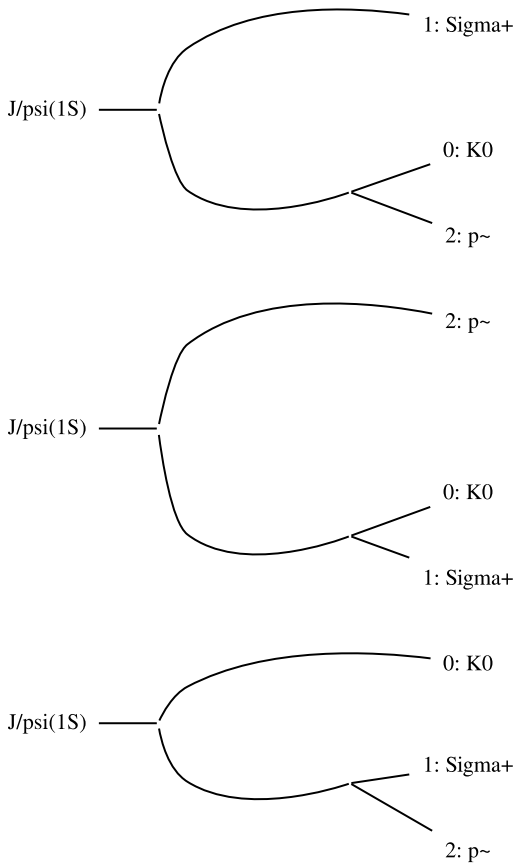


# QRules

## Quantum number conservation rules

**PWA use case**: compute which particle reactions are allowed between a given initial and final state

1. User specifies some boundary conditions
   (particle names, allowed interactions, isobar model, etc.)
2. QRules then:
   - determines all possible decay topologies,
   - gets corresponding particle properties from the PDG
     (or any custom definitions),

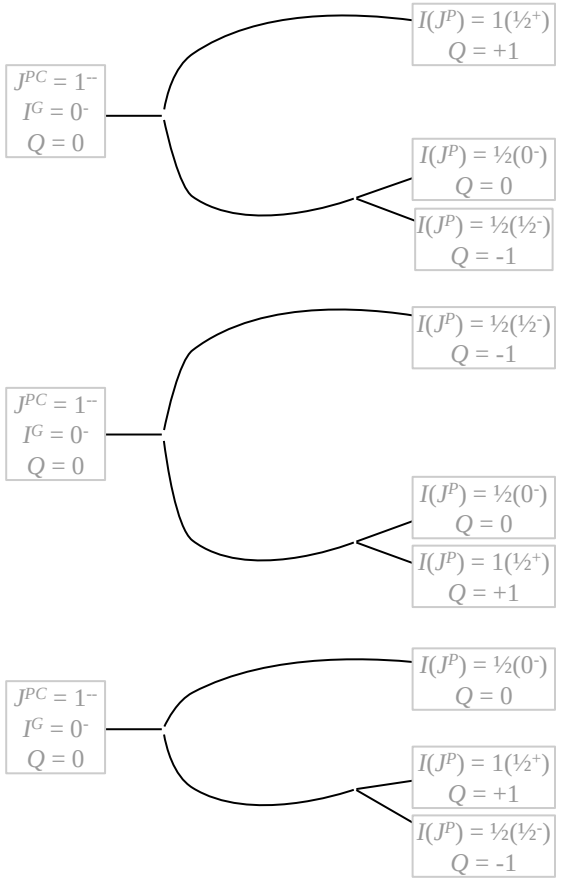


# QRules

## Quantum number conservation rules

**PWA use case**: compute which particle reactions are allowed between a given initial and final state

1. User specifies some boundary conditions
   (particle names, allowed interactions, isobar model, etc.)
2. QRules then:
   - determines all possible decay topologies,
   - gets corresponding particle properties from the PDG
     (or any custom definitions),

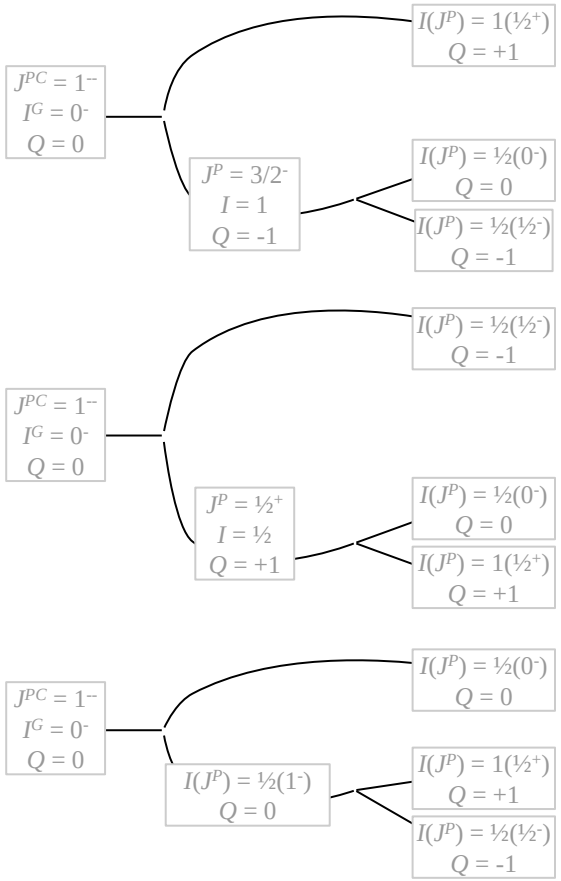


# QRules

## Quantum number conservation rules

**PWA use case**: compute which particle reactions are allowed between a given initial and final state

1. User specifies some boundary conditions
   (particle names, allowed interactions, isobar model, etc.)
2. QRules then:
   - determines all possible decay topologies,
   - gets corresponding particle properties from the PDG
     (or any custom definitions),
   - propagates quantum numbers through intermediate edges,
   - and selects all allowed transitions with its conservation laws

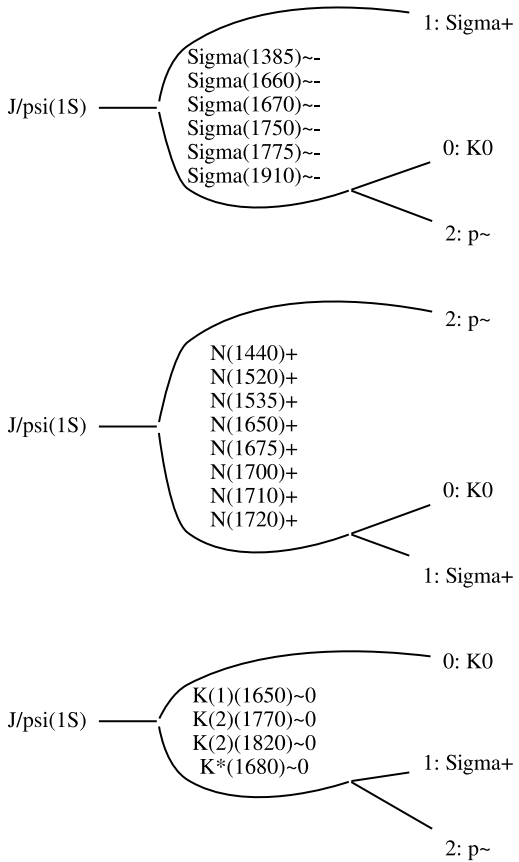


# QRules

## Quantum number conservation rules

**PWA use case**: compute which particle reactions are allowed between a given initial and final state

1. User specifies some boundary conditions
   (particle names, allowed interactions, isobar model, etc.)
2. QRules then:
   - determines all possible decay topologies,
   - gets corresponding particle properties from the PDG
     (or any custom definitions),
   - propagates quantum numbers through intermediate edges,
   - and selects all allowed transitions with its conservation laws



# QRules

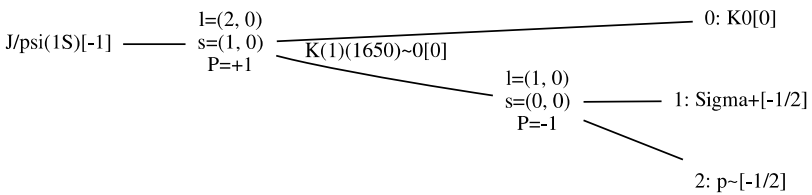
## Quantum number conservation rules

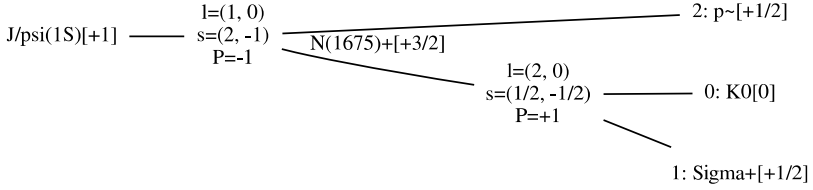
The returned object contains **all information to formulate an amplitude model**!

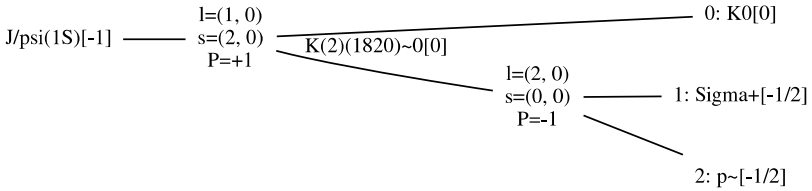
```
reaction = qrules.generate_transitions(
   initial_state="J/psi(1S)",
   final_state=["K0", "Sigma+", "p~"],
   allowed_interaction_types=["strong"],
)
```

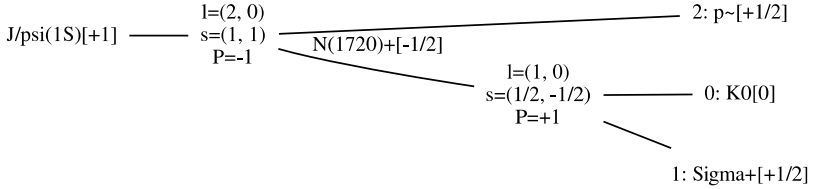
Selects conservation rules

launch binder | Open in Colab



# AmpForm

## Symbolic amplitude model formulation

- Library of **spin formalisms** and **dynamics**
- Formulate QRules' state transitions as an amplitude model
- Formulated as **algebraic expressions** (SymPy)
- Serves as **template to a computational back-end** for fitting and generating data distributions

```
n = sp.Symbol("n_R")
matrix = RelativisticKMatrix.formulate(
    n_channels=1,
    n_poles=n,
)
matrix[0, 0]
```

$$\frac{\rho\left(s\right)\sum_{R=1}^{n_R}\frac{\Gamma(s)\gamma_{R,0}^2 m_R}{-s+m_R^2}}{-i\rho\left(s\right)\sum_{R=1}^{n_R}\frac{\Gamma(s)\gamma_{R,0}^2 m_R}{-s+m_R^2}+1}$$

```
matrix = NonRelativisticKMatrix.formulate(
    n_channels=2,
    n_poles=1,
).doit()
matrix[0, 0].simplify()
```

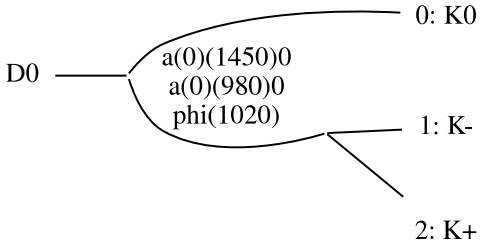
launch binder Open in Colab

$$-\frac{\Gamma_{1,0}\gamma_{1,0}^2 m_1}{s+i\Gamma_{1,0}\gamma_{1,0}^2 m_1+i\Gamma_{1,1}\gamma_{1,1}^2 m_1-m_1^2}$$

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




**Example**: amplitude model for $D^0 \to K^0\, K^-\, K^+$ with 3 resonances

```
builder = ampform.get_builder(reaction)
resonances = reaction.get_intermediate_particles()
for p in resonances:
   builder.set_dynamics(p.name, create_relativistic_breit_wigner_with_ff)
builder.set_dynamics("a(0)(980)0", create_analytic_breit_wigner)
model = builder.formulate()
```

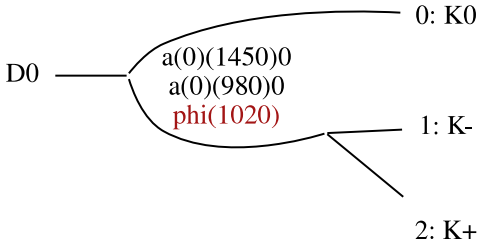
launch binder | Open in Colab

$$\left| A_{D_0^0 \to K_0^0 \phi(1020)_0;\phi(1020)_0 \to K_0^+ K_0^-} + A_{D_0^0 \to K_0^0 a_0(1450)_0^0;a_0(1450)_0^0 \to K_0^+ K_0^-} + A_{D_0^0 \to K_0^0 a_0(980)_0^0;a_0(980)_0^0 \to K_0^+ K_0^-} \right|^2$$

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




**Example**: amplitude model for $D^0 \to K^0\, K^-\, K^+$ with 3 resonances

```
builder = ampform.get_builder(reaction)
resonances = reaction.get_intermediate_particles()
for p in resonances:
   builder.set_dynamics(p.name, create_relativistic_breit_wigner_with_ff)
builder.set_dynamics("a(0)(980)0", create_analytic_breit_wigner)
model = builder.formulate()
```

launch binder | Open in Colab

$$\left| A_{D_0^0 \to K_0^0 \phi(1020)_0; \phi(1020)_0 \to K_0^+ K_0^-} + A_{D_0^0 \to K_0^0 a_0(1450)_0^0; a_0(1450)_0^0 \to K_0^+ K_0^-} + A_{D_0^0 \to K_0^0 a_0(980)_0^0; a_0(980)_0^0 \to K_0^+ K_0^-} \right|^2$$

Each amplitude can be further inspected:

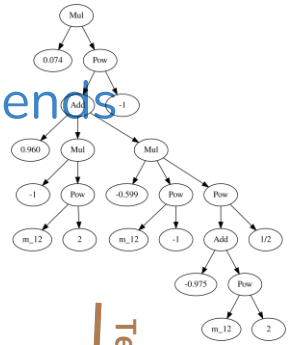
```
model.components[R"A_{D^{0}_{0} \to K^{0}_{0} \phi(1020)_{0}; \phi(1020)_{0} \to K^{+}_{0} K^{-}_{0}}"]
```

$$\frac{C_{D^0 \to K_0^0 \phi(1020)_0; \phi(1020) \to K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2\left(d^2_{\phi(1020)} q^2_{122}\left(m^2_{12}\right)\right)} D^0_{0,0}(-\phi_0, \theta_0, 0) D^1_{0,0}\left(-\phi_1^{12}, \theta_1^{12}, 0\right)}{-m_{12}^2 + m^2_{\phi(1020)} - i m_{\phi(1020)} \Gamma_{1020}\left(m^2_{12}\right)}$$

tensorwaves.rtfd.io

# TensorWaves

## Fit and generate data with computational back-ends




**TensorWaves responsibilities:**

- Express mathematical expressions in a computational back-end
- Generate (deterministic) amplitude-based Monte Carlo samples
- Perform unbinned fits with different back-ends
  (TensorFlow, NumPy, JAX, …)
- Also integrates different optimizers (Minuit2, SciPy, …)

Any symbolic input

```
function = create_parametrized_function(expression, parameter_defaults, backend="jax")
estimator = UnbinnedNLL(function, data, phsp, backend="jax")
optimizer = Minuit2(callback=CSVSummary("fit_traceback.csv"))
fit_result = optimizer.optimize(estimator, initial_parameters)
```

Open in Colab

TensorWaves

NumPy

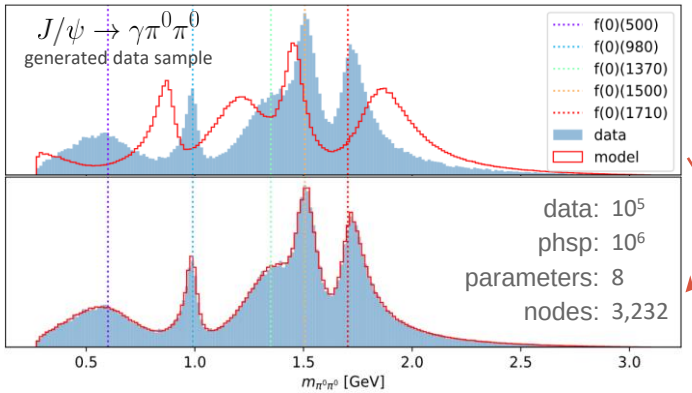
Numba

# TensorWaves

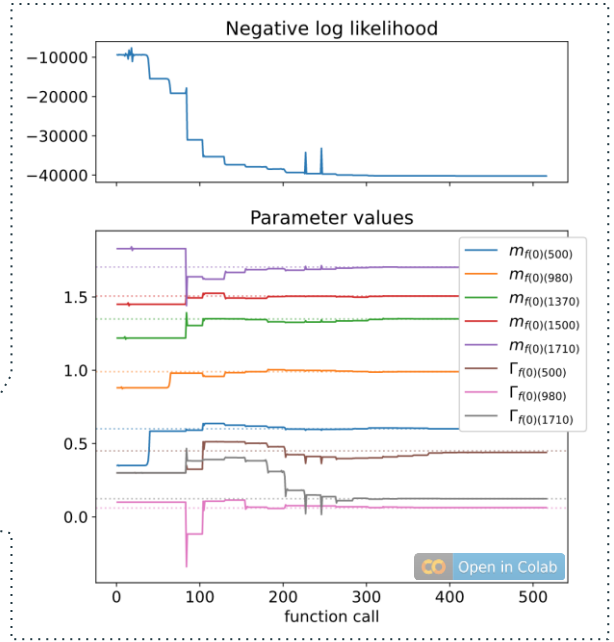
## Fit and generate data with computational back-ends

**Does it work?** JAX+Minuit2 example benchmark:

| | |
|---|---|
| Intel Core i7-8750H CPU @ 2.20GHz 12 cores | **56s** |
| GeForce GTX 1050 Mobile GPU @ 1.35GHz | **47s** |
| Tesla K80 GPU (Colab) | **15s** |
| Intel Xeon CPU @ 2.20GHz 1 core (Colab) | **3m20** |




optimise parameters



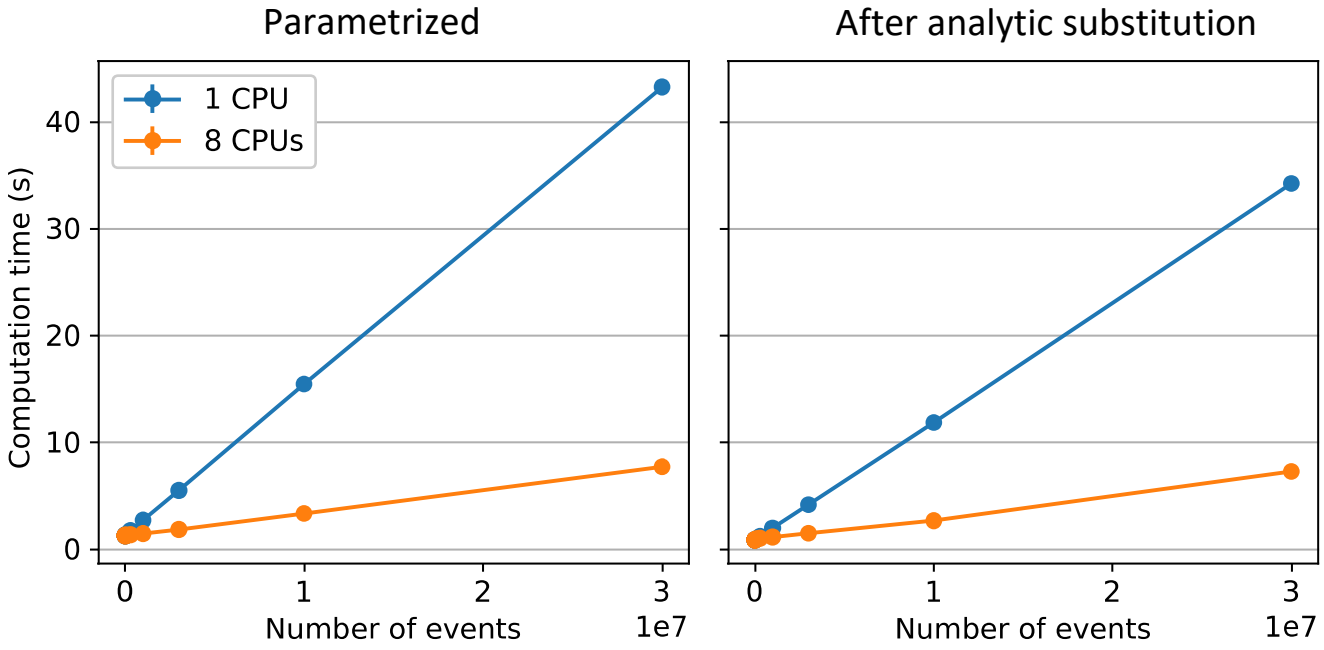
# TensorWaves

## Fit and generate data with computational back-ends

Amplitude model for $\Lambda_c \to p\pi K$
12 resonances, 59 parameters,
DPD alignment for 3 subsystems

Expression tree complexity:
parametrized: 43,198 nodes
substituted: 9,624 nodes

Backend: JAX
CPU: Intel i7-8750H 2.20GHz
→ computation time
**decreases by 25%**



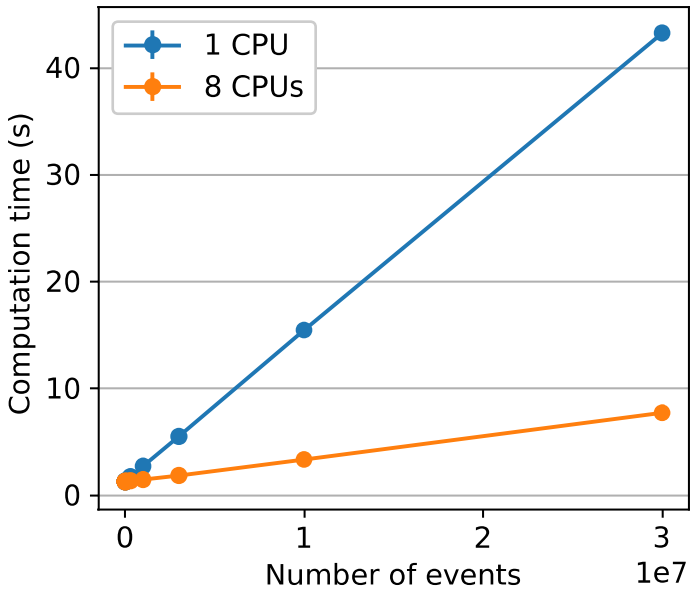
# TensorWaves

## Fit and generate data with computational back-ends

Amplitude model for $\Lambda_c \to p\pi K$
12 resonances, 59 parameters,
DPD alignment for 3 subsystems

Expression tree complexity:
parametrized: 43,198 nodes
substituted: 9,624 nodes

SymPy+JAX (Python)




Julia