

RUHR-UNIVERSITÄT BOCHUM

The ComPWA project

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

Self-introduction

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

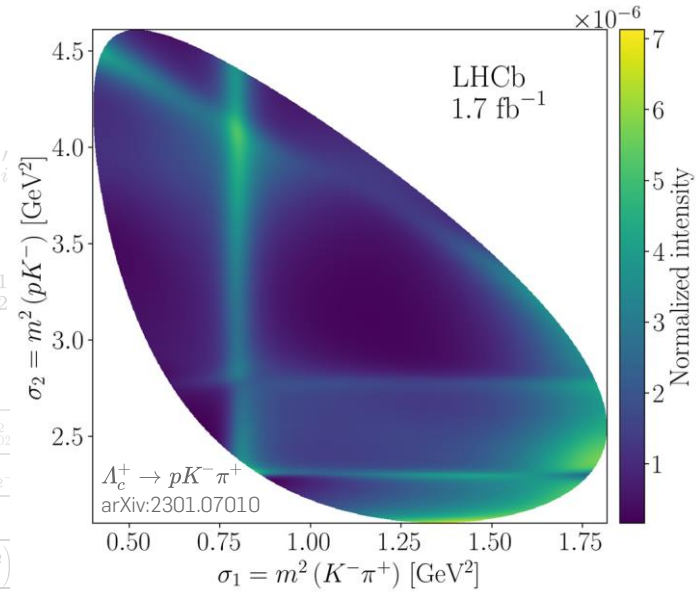
- Research: amplitude analysis (partial wave analysis)
 - Analyzing N^* resonances in J/ψ decays for the BESIII collaboration
 - Also in PANDA collaboration at FAIR, Darmstadt
- Developer for the Common Partial Wave Analysis project (ComPWA)
→ Next slides: symbolic amplitude models + JAX, TF, ...
- Additional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

The logo for BESIII, featuring the letters 'B', 'E', 'S' in blue, red, and green respectively, followed by 'III' in black.The logo for PANDA, with the word 'Panda' in a stylized font where the 'P' is enclosed in a rounded rectangle with a yellow bar at the bottom.The logo for ComPWA, with the letters 'C', 'o', 'm', 'P', 'W', 'A' in blue, where the 'o' and 'A' are stylized with circular and triangular shapes respectively.

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!)



$$\begin{aligned}
 & \frac{1}{\sqrt{2}} \left(\frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} + A_{\lambda_0, \lambda_1}^2 d_{\lambda_0, \lambda_1}^{\frac{1}{2}} (\zeta_{1(1)}^0) + A_{\lambda_0', \lambda_1'}^2 d_{\lambda_0', \lambda_1'}^{\frac{1}{2}} (\zeta_{1(1)}^1) \right) \\
 & \frac{1}{\sqrt{2}} \left(\frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} + \frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} \right) \\
 & \frac{1}{\sqrt{2}} \left(\frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} + \frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} \right) \\
 & \frac{1}{\sqrt{2}} \left(\frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} + \frac{d_{\Sigma(1660)^-}^2 (m_{\Sigma(1660)^-}^2 - (m_0 - m_2)^2) (m_{\Sigma(1660)^-}^2 - (m_0 + m_2)^2)}{4m_{\Sigma(1660)^-}^2} \right)
 \end{aligned}$$

Context: amplitude analysis software

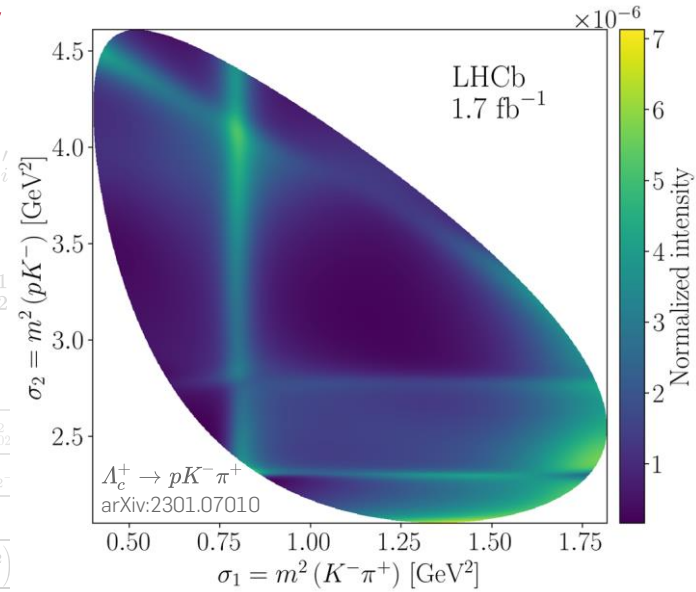
What makes amplitude analysis challenging?

- 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!)

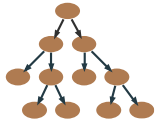
fast computations

flexibility

documentation



Mission: bring code closer to theory



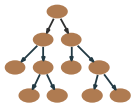
High performance through **computational back-ends**
from ML and data science



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, ...



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

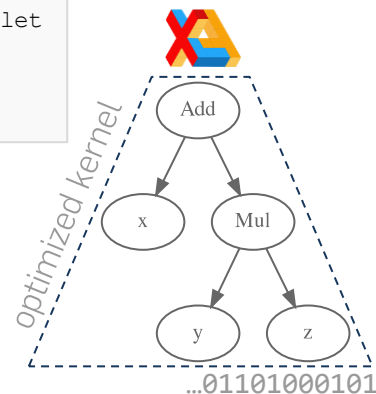
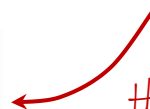
```
@tf.function(jit_compile=True)  
def my_expression(x, y, z):  
    return x + y * z
```

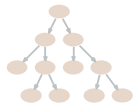
Converted to device-agnostic XLA code



```
{ 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





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];  
  }  
}
```

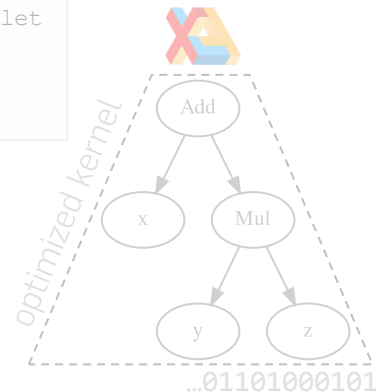
```
@tf.function(jit_compile=True)  
def my_expression(x, y, z):  
    return x + y * z
```

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,) }
```

Heavy lifting by optimized backend





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)
```



$$\frac{N}{m_0^2 - im_0\Gamma_0 - s}$$

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

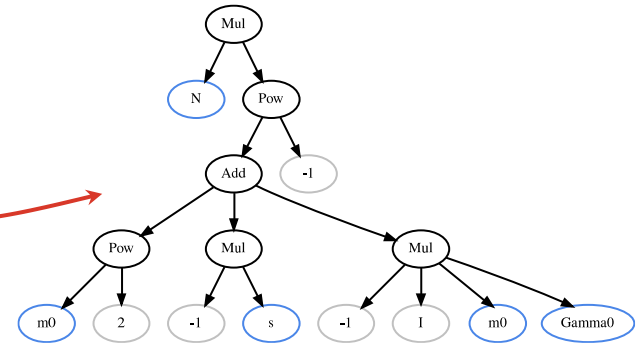
- **Transparency:** inspect the math as you formulate the model
- **Flexibility:** modify the model with analytic substitutions
- **Performance:** simplify expressions algebraically
- **Code generation:** symbolic model as template to computational back-ends (SSoT)

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



$$\frac{N}{m_0^2 - im_0\Gamma_0 - s}$$

CAS represents expression as a tree

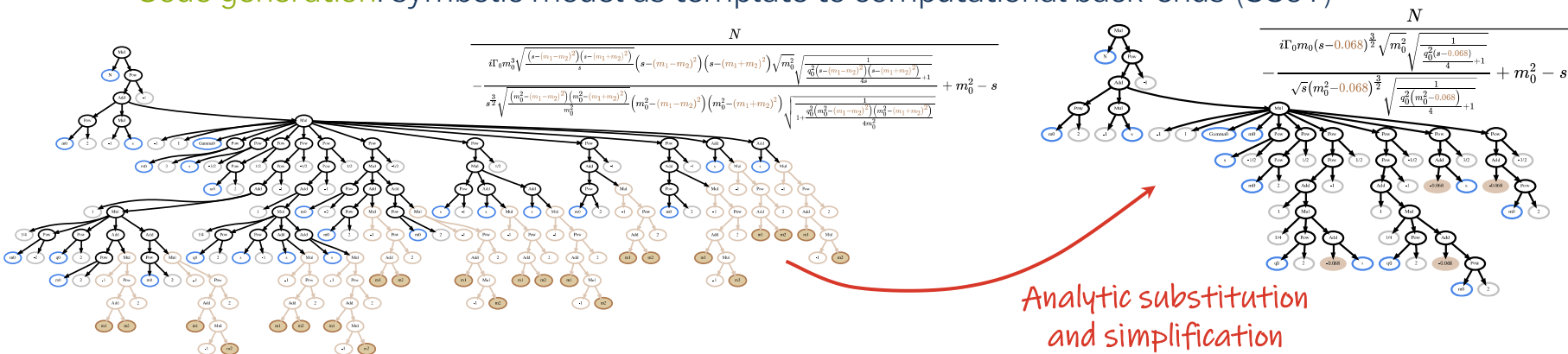




Symbolic amplitude models

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

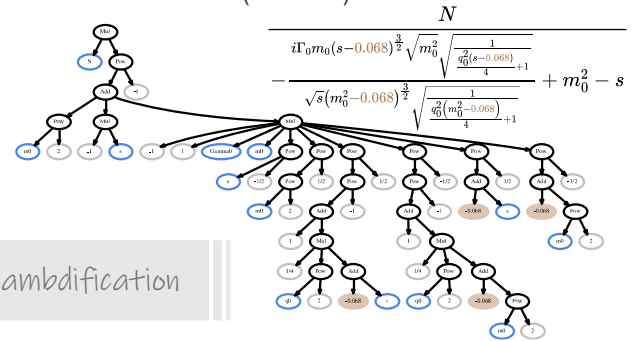
A new technique: formulate your amplitude model with a Computer Algebra System

- **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)

```
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

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)

```

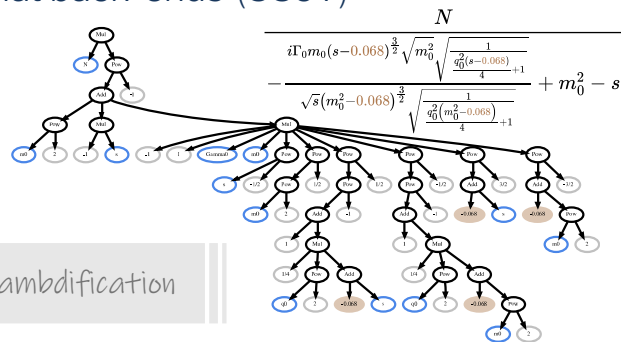
REAL*8 function my_expr(Gamma0, N, m0, s)
implicit none
REAL*8, intent(in) :: Gamma0
REAL*8, intent(in) :: N
REAL*8, intent(in) :: m0
REAL*8, intent(in) :: s

```

```

my_expr = N/(-cplx(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

```





Symbolic amplitude models

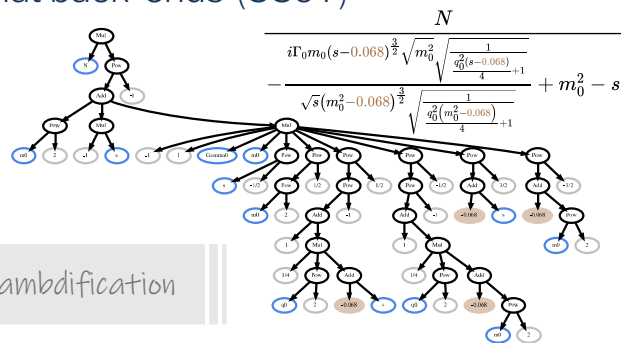
A new technique: formulate your amplitude model with a Computer Algebra System

- **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);
```



Sympy lambdification

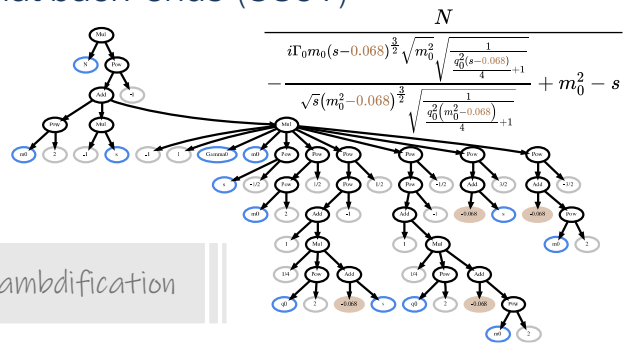


Symbolic amplitude models

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
    )
```



← Sympy lambdification



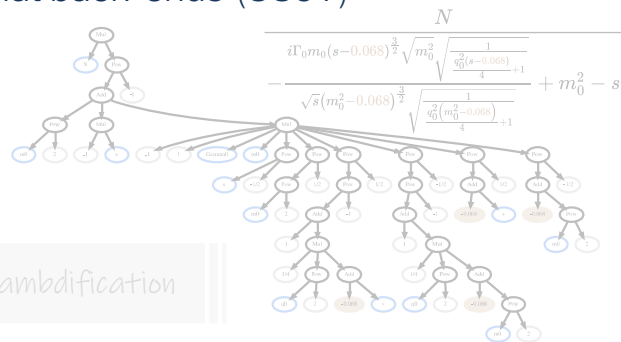
Symbolic amplitude models

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))
    )
```

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



← Sympy lambdification



Symbolic amplitude models

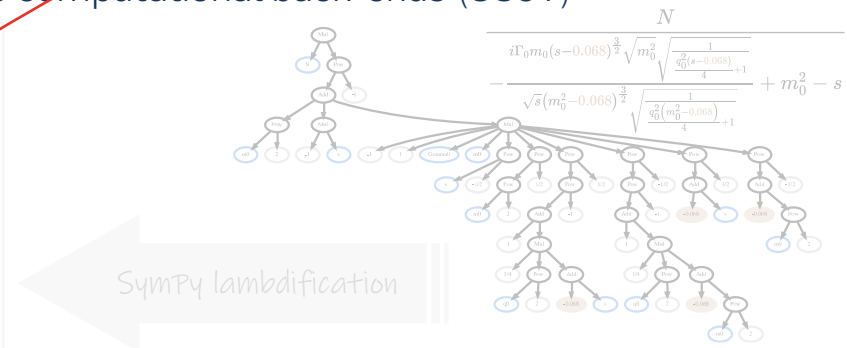
A new technique: formulate your amplitude model with a Computer Algebra System

- **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!

```
@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))
    )
```

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



Self-documenting workflow



$\Lambda_c \rightarrow p K \pi$ 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 $\Lambda_c^+ \rightarrow p K^- \pi^+$

DOI [10.48550/arXiv.2301.07010](https://doi.org/10.48550/arXiv.2301.07010) DOI [10.5281/zenodo.7544989](https://doi.org/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 $\Lambda_c^+ \rightarrow p K^- \pi^+$ 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 $\Lambda_c^+ \rightarrow p K^- \pi^+$ decay mode in hadronic amplitude analyses.

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

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

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:

Self-documenting workflow



$\Lambda_c \rightarrow p K \pi$ 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)



Contents

1.1. Resonances and LS-scheme

1.2. Amplitude

1.2.1. Spin-alignment amplitude

1.2.2. Sub-system amplitudes

1.3. Parameter definitions

1.3.1. Helicity coupling values

1.3.2. Non-coupling parameters

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

$$\sum_{\lambda_0=-1/2}^{1/2} \sum_{\lambda_1=-1/2}^{1/2} A_{\lambda_0, \lambda_1}^1 d_{\lambda_1, \lambda_1}^{1/2}(\zeta_{1(1)}^1) d_{\lambda_0, \lambda_0}^{1/2}(\zeta_{1(1)}^0) + A_{\lambda_0, \lambda_1}^2 d_{\lambda_1, \lambda_1}^{1/2}(\zeta_{2(1)}^1) d_{\lambda_0, \lambda_0}^{1/2}(\zeta_{2(1)}^0) + A_{\lambda_0, \lambda_1}^3 d_{\lambda_1, \lambda_1}^{1/2}(\zeta_{3(1)}^1) d_{\lambda_0, \lambda_0}^{1/2}(\zeta_{3(1)}^0)$$

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 $\zeta_{j(k)}^i$ angles are defined as:

Show code cell source

$$\begin{aligned} \zeta_{1(1)}^0 &= 0 \\ \zeta_{1(1)}^1 &= 0 \\ \zeta_{2(1)}^0 &= -\arccos\left(\frac{-2m_0^2(-m_1^2-m_2^2+\sigma_3)+(m_0^2+m_1^2-\sigma_1)(m_0^2+m_2^2-\sigma_2)}{\sqrt{\lambda(m_0^2, m_2^2, \sigma_2)}\sqrt{\lambda(m_0^2, \sigma_1, m_1^2)}}\right) \\ \zeta_{3(1)}^0 &= \arccos\left(\frac{2m_1^2(-m_0^2-m_2^2+\sigma_3)+(m_0^2+m_1^2-\sigma_1)(-m_1^2-m_2^2+\sigma_2)}{\sqrt{\lambda(m_0^2, m_2^2, \sigma_2)}\sqrt{\lambda(m_0^2, \sigma_1, m_1^2)}}\right) \end{aligned}$$

Mathematical expressions are automatic rendering of the implemented amplitude models

Self-documenting workflow



$\Lambda_c \rightarrow p K \pi$ 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
 - 7.1. Dynamics lineshapes
 - 7.2. DPD angles
 - 7.3. Phase space sample
 - 7.4. Alignment consistency
 - 7.5. Benchmarking
 - 7.6. Serialization
 - 7.7. Amplitude model with LS-couplings
 - 7.8. SU(2) → SO(3) homomorphism
 - 7.9. Determination of polarization
 - 7.10. Interactive visualization



Contents

- 7.7.1. Model inspection
- 7.7.2. Distribution
- 7.7.3. Decay rates

7.7.1. Model inspection

Show code cell source

$$\sum_{\lambda_0=-1/2}^{1/2} \sum_{\lambda_1=-1/2}^{1/2} A_{\lambda_0, \lambda_1}^1 d_{\lambda_1, \lambda_1}^{\frac{1}{2}}(\zeta_{1(1)}) d_{\lambda_0, \lambda_0}^{\frac{1}{2}}(\zeta_{1(1)}) + A_{\lambda_0, \lambda_1}^2 d_{\lambda_1, \lambda_1}^{\frac{1}{2}}(\zeta_{2(1)}) d_{\lambda_0, \lambda_0}^{\frac{1}{2}}(\zeta_{2(1)}) + A_{\lambda_0, \lambda_1}^3 d_{\lambda_1, \lambda_1}^{\frac{1}{2}}(\zeta_{3(1)}) d_{\lambda_0, \lambda_0}^{\frac{1}{2}}(\zeta_{3(1)})$$

Show code cell source

$$A_{-\frac{1}{2}, -\frac{1}{2}}^1 = \sum_{\lambda_R=-1}^1 \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \frac{1}{2} \mathcal{R}(\sigma_1) C_{1, \lambda_R, \frac{1}{2}, \frac{1}{2}}^{\frac{3}{2}, \lambda_R, \frac{1}{2}} C_{2, 0, \frac{3}{2}, \lambda_R, \frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} \mathcal{H}_{K(892), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{K(892), 0, 0}^{\text{decay}} d_{\lambda_R, 0}^1(\theta_{23})}{2} + \sum_{\lambda_R=-1}^1 \frac{\sqrt{2\delta} \frac{1}{2} \lambda_R \frac{1}{2} \mathcal{R}(\sigma_1) C_{0, 0, \frac{3}{2}, \lambda_R, \frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} C_{1, \lambda_R, \frac{1}{2}, \frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} \mathcal{H}_{K(892), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{K(892), 0, 0}^{\text{decay}} d_{\lambda_R, 0}^1(\theta_{23})}{2}$$

$$A_{-\frac{1}{2}, -\frac{1}{2}}^2 = \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_2) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{L(1520), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{L(1520), 0, -\frac{1}{2}}^{\text{decay}} d_{\lambda_R, \frac{1}{2}}^{\frac{3}{2}}(\theta_{31})}{2} + \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_2) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{L(1690), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{L(1690), 0, -\frac{1}{2}}^{\text{decay}} d_{\lambda_R, \frac{1}{2}}^{\frac{3}{2}}(\theta_{31})}{2}$$

$$A_{-\frac{1}{2}, -\frac{1}{2}}^3 = \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_3) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{D(1232), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{D(1232), -\frac{1}{2}, 0}^{\text{decay}} d_{\lambda_R, -\frac{1}{2}}^{\frac{3}{2}}(\theta_{12})}{2} + \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_3) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{D(1600), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{D(1600), -\frac{1}{2}, 0}^{\text{decay}} d_{\lambda_R, -\frac{1}{2}}^{\frac{3}{2}}(\theta_{12})}{2}$$

$$A_{-\frac{1}{2}, \frac{1}{2}}^1 = \sum_{\lambda_R=-1}^1 \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \frac{1}{2} \mathcal{R}(\sigma_1) C_{1, \lambda_R, \frac{1}{2}, -\frac{1}{2}}^{\frac{3}{2}, \lambda_R, \frac{1}{2}} C_{2, 0, \frac{3}{2}, \lambda_R, -\frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} \mathcal{H}_{K(892), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{K(892), 0, 0}^{\text{decay}} d_{\lambda_R, 0}^1(\theta_{23})}{2} + \sum_{\lambda_R=-1}^1 \frac{\sqrt{2\delta} \frac{1}{2} \lambda_R \frac{1}{2} \mathcal{R}(\sigma_1) C_{0, 0, \frac{3}{2}, \lambda_R, -\frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} C_{1, \lambda_R, \frac{1}{2}, -\frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} \mathcal{H}_{K(892), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{K(892), 0, 0}^{\text{decay}} d_{\lambda_R, 0}^1(\theta_{23})}{2}$$

$$A_{-\frac{1}{2}, \frac{1}{2}}^2 = \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_2) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{L(1520), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{L(1520), 0, \frac{1}{2}}^{\text{decay}} d_{\lambda_R, -\frac{1}{2}}^{\frac{3}{2}}(\theta_{31})}{2} + \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_2) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{L(1690), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{L(1690), 0, \frac{1}{2}}^{\text{decay}} d_{\lambda_R, -\frac{1}{2}}^{\frac{3}{2}}(\theta_{31})}{2}$$

$$A_{-\frac{1}{2}, \frac{1}{2}}^3 = \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_3) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{D(1232), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{D(1232), \frac{1}{2}, 0}^{\text{decay}} d_{\lambda_R, \frac{1}{2}}^{\frac{3}{2}}(\theta_{12})}{2} + \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_3) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{D(1600), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{D(1600), \frac{1}{2}, 0}^{\text{decay}} d_{\lambda_R, \frac{1}{2}}^{\frac{3}{2}}(\theta_{12})}{2}$$

$$A_{\frac{1}{2}, -\frac{1}{2}}^1 = \sum_{\lambda_R=-1}^1 \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \frac{1}{2} \mathcal{R}(\sigma_1) C_{1, \lambda_R, \frac{1}{2}, \frac{1}{2}}^{\frac{3}{2}, \lambda_R, \frac{1}{2}} C_{2, 0, \frac{3}{2}, \lambda_R, \frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} \mathcal{H}_{K(892), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{K(892), 0, 0}^{\text{decay}} d_{\lambda_R, 0}^1(\theta_{23})}{2} + \sum_{\lambda_R=-1}^1 \frac{\sqrt{2\delta} \frac{1}{2} \lambda_R \frac{1}{2} \mathcal{R}(\sigma_1) C_{0, 0, \frac{3}{2}, \lambda_R, \frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} C_{1, \lambda_R, \frac{1}{2}, \frac{1}{2}}^{\frac{1}{2}, \lambda_R, \frac{1}{2}} \mathcal{H}_{K(892), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{K(892), 0, 0}^{\text{decay}} d_{\lambda_R, 0}^1(\theta_{23})}{2}$$

$$A_{\frac{1}{2}, -\frac{1}{2}}^2 = \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_2) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{L(1520), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{L(1520), 0, -\frac{1}{2}}^{\text{decay}} d_{\lambda_R, \frac{1}{2}}^{\frac{3}{2}}(\theta_{31})}{2} + \sum_{\lambda_R=-3/2}^{3/2} \frac{\sqrt{10\delta} \frac{1}{2} \lambda_R \mathcal{R}(\sigma_2) C_{\frac{3}{2}, \lambda_R, 0, 0}^{\frac{3}{2}, \lambda_R} C_{2, 0, \frac{3}{2}, \lambda_R}^{\frac{1}{2}, \lambda_R} \mathcal{H}_{L(1690), 2, \frac{3}{2}}^{\text{LS, production}} \mathcal{H}_{L(1690), 0, -\frac{1}{2}}^{\text{decay}} d_{\lambda_R, \frac{1}{2}}^{\frac{3}{2}}(\theta_{31})}{2}$$

Mathematical expressions are automatic rendering of the implemented amplitude models

Self-documenting workflow



$\Lambda_c \rightarrow p K \pi$ 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
 - 7.1. Dynamics lineshapes
 - 7.2. DPD angles
 - 7.3. Phase space sample
 - 7.4. Alignment consistency
 - 7.5. Benchmarking
 - 7.6. Serialization
 - 7.7. Amplitude model with LS-couplings
 - 7.8. $SU(2) \rightarrow SO(3)$ homomorphism
 - 7.9. Determination of polarization
 - 7.10. Interactive visualization



Contents

- 7.1.1. Relativistic Breit-Wigner
- 7.1.2. Bugg Breit-Wigner
- 7.1.3. Flatté for S-waves

7.1.1. Relativistic Breit-Wigner

Show code cell source

$$\mathcal{R}(s) = \frac{F_{1R}(R_{res} p_{m_1, m_2}(s)) F_{1\Lambda_c}(R_{\Lambda_c} q_{m_{top}, m_{spectator}}(s)) \left(\frac{p_{m_1, m_2}(s)}{p_{m_1, m_2}(m^2)} \right)^{l_R} \left(\frac{q_{m_{top}, m_{spectator}}(s)}{q_{m_{top}, m_{spectator}}(m^2)} \right)^{l_{\Lambda_c}}}{F_{1R}(R_{res} p_{m_1, m_2}(m^2)) F_{1\Lambda_c}(R_{\Lambda_c} q_{m_{top}, m_{spectator}}(m^2)) \left(\frac{p_{m_1, m_2}(m^2)}{p_{m_1, m_2}(m^2)} \right) \left(\frac{q_{m_{top}, m_{spectator}}(m^2)}{q_{m_{top}, m_{spectator}}(m^2)} \right)} m^2 - i m \Gamma(s) - s$$

7.1.2. Bugg Breit-Wigner

Show code cell source

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

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

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

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

Show code cell source

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

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

Self-documenting workflow



$\Lambda_c \rightarrow p K \pi$ 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)

vector $\vec{\alpha}_i$ and the one from the nominal model, $\vec{\alpha}_0$:

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

The solid angle can then be computed as:

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

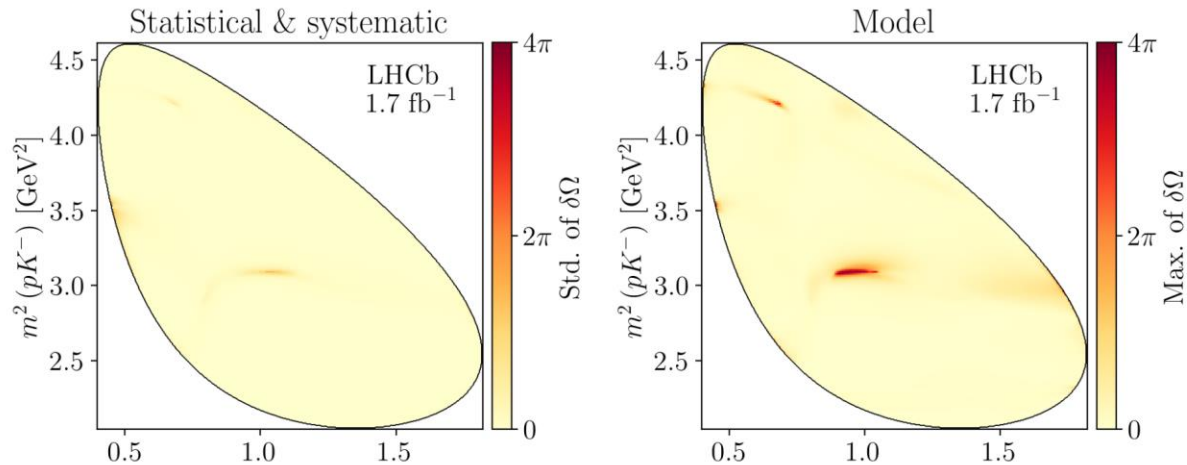
The statistical uncertainty is given by taking the standard deviation on the $\delta\Omega$ distribution and the systematic uncertainty is given by taking finding $\theta_{\max} = \max \theta_i$ and computing $\delta\Omega_{\max}$ from that.

Show code cell source

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

- Contents
- 5.1. Model loading
- 5.2. Statistical uncertainty
- 5.3. Distributions
- 5.4. Uncertainty on polarimetry
- 5.5. Decay rates
- 5.6. Average polarimetry values
- 5.7. Exported distributions

Uncertainty over $\vec{\alpha}$ polar angle



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. 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()
```

```
class EnergyDependentWidth(s: Symbol, mass0: Symbol,
gamma0: Symbol, m_a: Symbol, m_b: Symbol,
angular_momentum: Symbol, meson_radius: Symbol,
phsp_factor: Optional[PhaseSpaceFactorProtocol] =
None, name: Optional[str] = None, evaluate: bool =
False) [source]
```

Bases: `ampform.sympy.UnevaluatedExpression`

Mass-dependent width, coupled to the pole position of the

See PDG2020, [\\$Resonances](#), p.6 and [11], equation (6). Default value for `phsp_factor` is `PhaseSpaceFactor()`.

Note that the `BlattWeisskopfSquared` of `AmpForm` is normalized in the sense that equal powers of z appear in the nominator and the denominator, while the definition in the PDG (as well as some other sources), always have 1 in the nominator of the Blatt-Weisskopf. In that case, one needs an additional factor $(q/q_0)^{2L}$ in the definition for $\Gamma(m)$.

With that in mind, the "mass-dependent" width in a `relativistic_breit_wigner_with_ff` becomes:

$$\Gamma_0(s) = \frac{\Gamma_0 B_L^2(q^2(s)) \rho(s)}{B_L^2(q^2(m_0^2)) \rho(m_0^2)} \quad (3)$$

by (2), and ρ is by

otocol

```
class PhaseSpaceFactor(s: Symbol, m_a: Symbol, m_b:
Symbol, **hints: Any) [source]
```

Standard phase-space factor, using

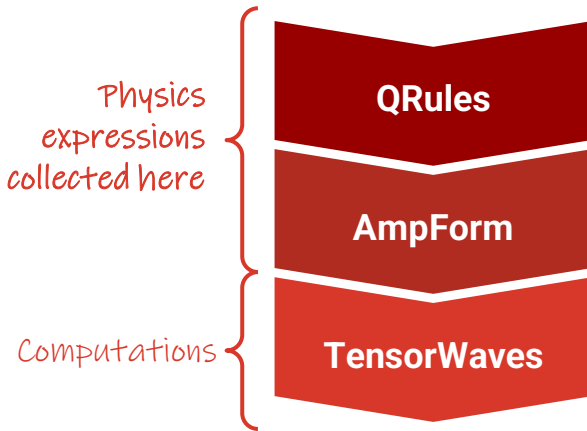
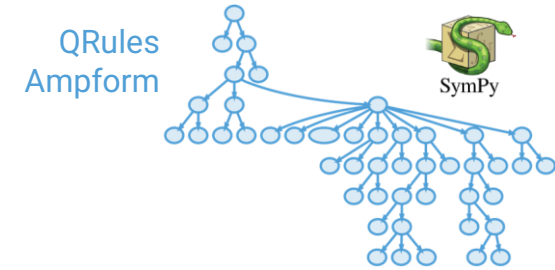
`BreakupMomentumSquared()`.

See PDG2020, [\\$Resonances](#), p.4, Equation (49.8).

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

TensorWaves



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.?)