

# The ComPWA project

Facilitating and automating amplitude analysis  
with modern Python tools  
and transparent, interactive documentation

Remco de Boer  
Ruhr University Bochum

13 December 2021  
PANDA Seminar

# What is the ComPWA project?



[github.com/ComPWA](https://github.com/ComPWA)

- “Common Partial Wave Analysis”
- Open-source GitHub organization
- Originally a C++ framework ([ComPWA](#))
- Now maintains a collection of Python tools and documentation for amplitude analysis
- Developed by PhD students and postdocs at RUB and JGU Mainz in the context of BESIII and PANDA analyses

# What does ComPWA aim for?

- **Academic continuity:**  
open source, collaboration-independent,  
maintainable, well-documented software
- Provide an **easy starting point**  
for researchers new to amplitude analysis
- Build up a modern, interlinked, and  
**interactive PWA knowledge-base**
- Offer libraries that can be used as  
**building blocks** in specialized applications

The screenshot shows a portion of the ComPWA API documentation. At the top right, there are navigation icons: a magnifying glass for search, a back arrow, and download symbols. Below the search bar is a sidebar with links to various sections: Installation, Usage, Formulae, amplitude model, Modify amplitude model, Inspect model interactively, Helicity versus canonical, Dynamics, Bibliography, and API. Under the API section, there are sub-links for dynamics, builder, kmatrix, helicity, sympy, kinematics, Changelog, Upcoming features, and Help developing. Further down, under RELATED PROJECTS, are links for QRules, TensorWaves, and PWA Pages. Under COMPWA ORGANIZATION, there are links for Website, GitHub Repositories, and About. On the right side, the main content area displays the code definition for the `EnergyDependentWidth` class. The code is as follows:

```
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)
```

Below the code, it says "Bases: ampform.sympy.UnevaluatedExpression". A note follows: "Mass-dependent width, coupled to the pole position of the resonance." Another note refers to PDG2020, §Resonances, p.6 and [11], equation (6). It states that the default value for `phsp_factor` is `PhaseSpaceFactor()`. A detailed note explains that the `BlattWeisskopfSquared` of AmpForm is normalized such that equal powers of  $z$  appear in the numerator and denominator, while the definition in the PDG (and some other sources) always have 1 in the numerator of the Blatt-Weisskopf. In that case, one needs an additional factor  $(q/q_0)^{2L}$  in the definition for  $\Gamma(m)$ . It concludes with the statement: "With that in mind, the ‘mass-dependent’ width in a `relativistic_breit_wigner_with_ff` becomes:

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

Below this, a note states: "where  $B_L^2$  is defined by (1),  $q$  is defined by (2), and  $\rho$  is (by default) defined by (4)." A pink box highlights the `phsp_factor` parameter with the text: "phsp\_factor: PhaseSpaceFactorProtocol".

At the bottom of the page, another code snippet is shown for the `PhaseSpaceFactor` class:

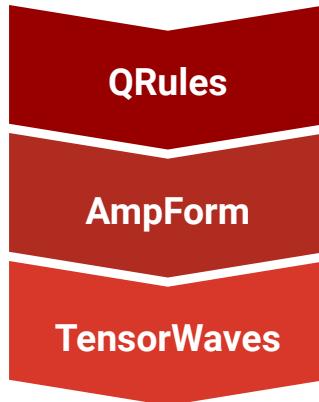
```
class PhaseSpaceFactor(s: Symbol, m_a: Symbol, m_b: Symbol, **hints: Any)
```

A note below it says: "Standard phase-space factor, using `BreakupMomentumSquared()`". Another note refers to PDG2020, §Resonances, p.4, Equation (49.8).

Screenshot from the API of one of ComPWA's packages

# What do we provide?

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



Automated quantum number conservation rules

Formulate symbolic amplitude model

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

# QRules

## Automated quantum number conservation rules

**Core:** ‘search engine’ for quantum numbers

Get particle properties:<sup>\*</sup>

```
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_reactionViolations(  
    initial_state="pi0",  
    final_state=["gamma", "gamma", "gamma"],  
)
```

[Launch binder](#) [Open in Colab](#)

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

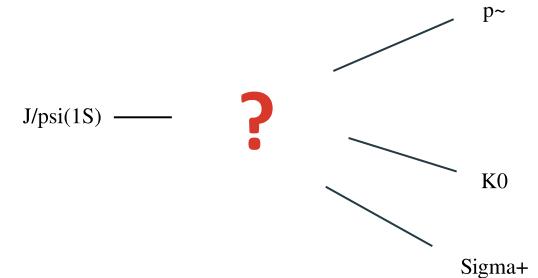
*Also a library of  
conservation rules*

<sup>\*</sup> PDG info computed from the scikit-hep [particle](#) package

# QRules

Automated quantum number conservation rules

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

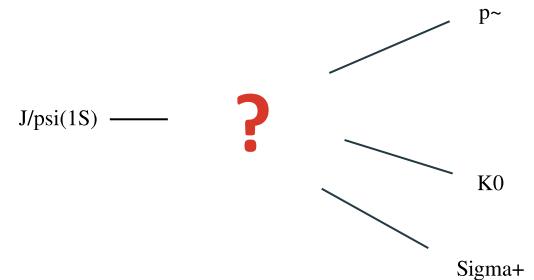


# QRules

Automated 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.)

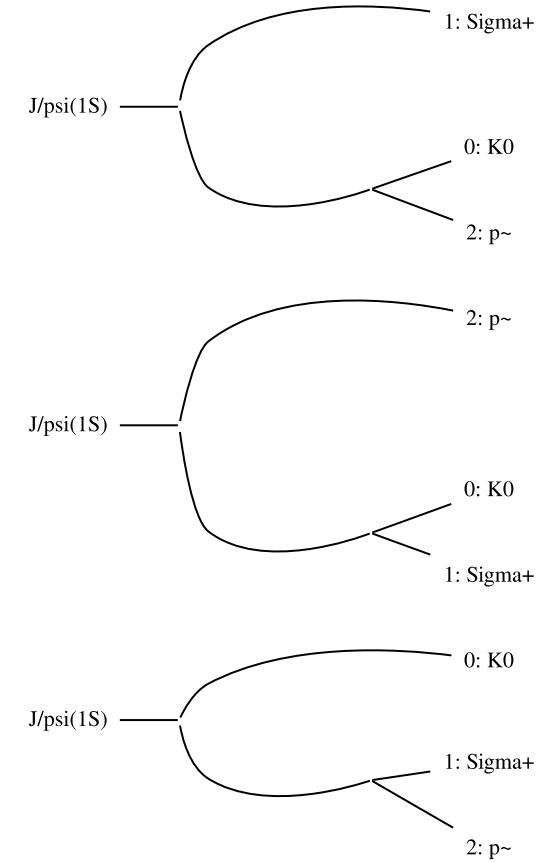


# QRules

## Automated 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:
  - o determines all possible decay topologies,

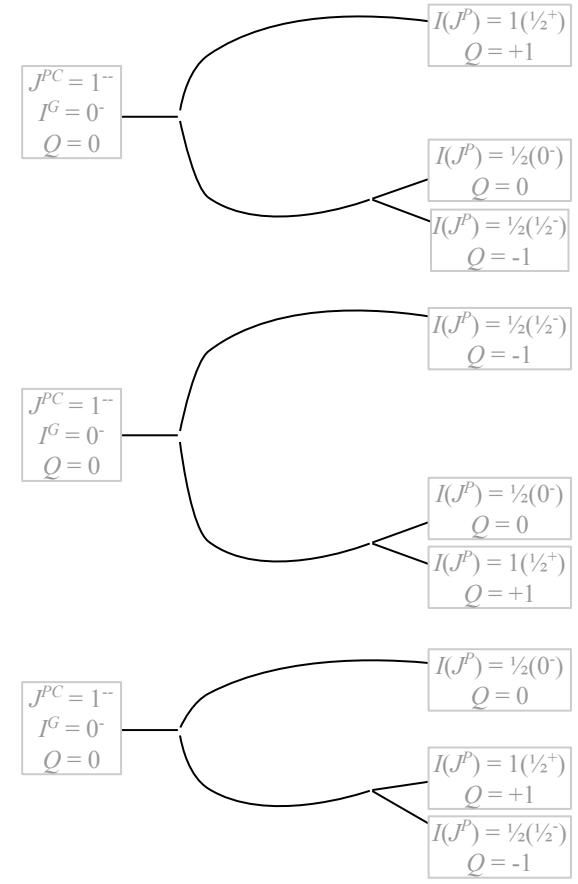


# QRules

## Automated 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:
  - o determines all possible decay topologies,
  - o gets corresponding particle properties from the PDG  
(or any custom definitions),

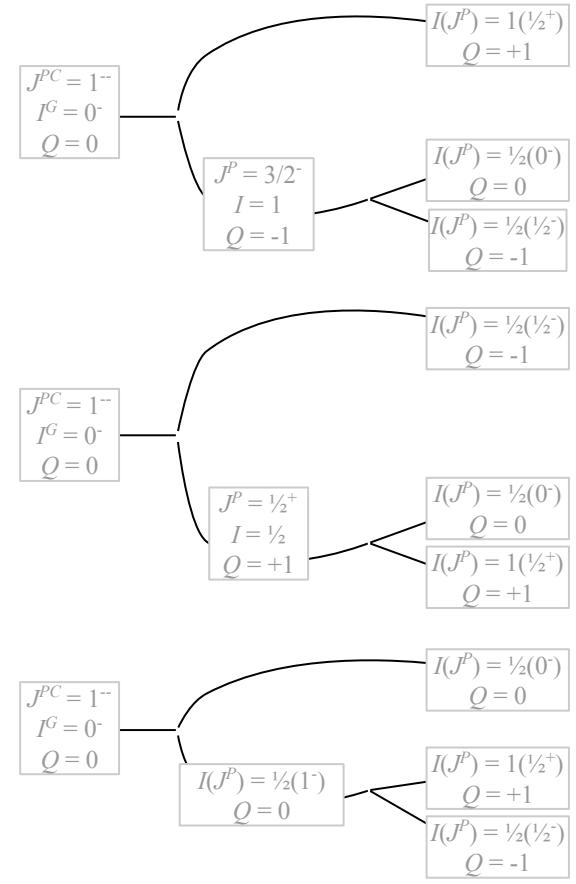


# QRules

## Automated 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



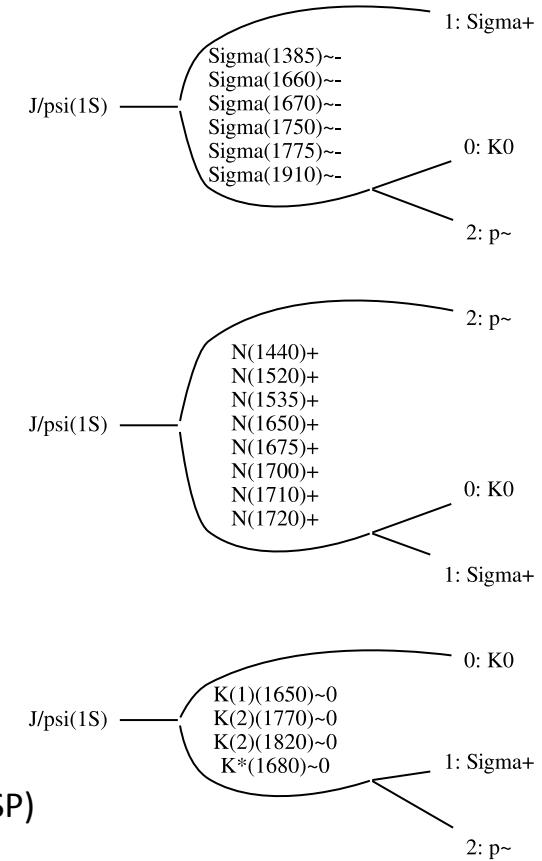
# QRules

## Automated 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

Generalized approach: **the constraints ‘span’ quantum number space (CSP)**



# QRules

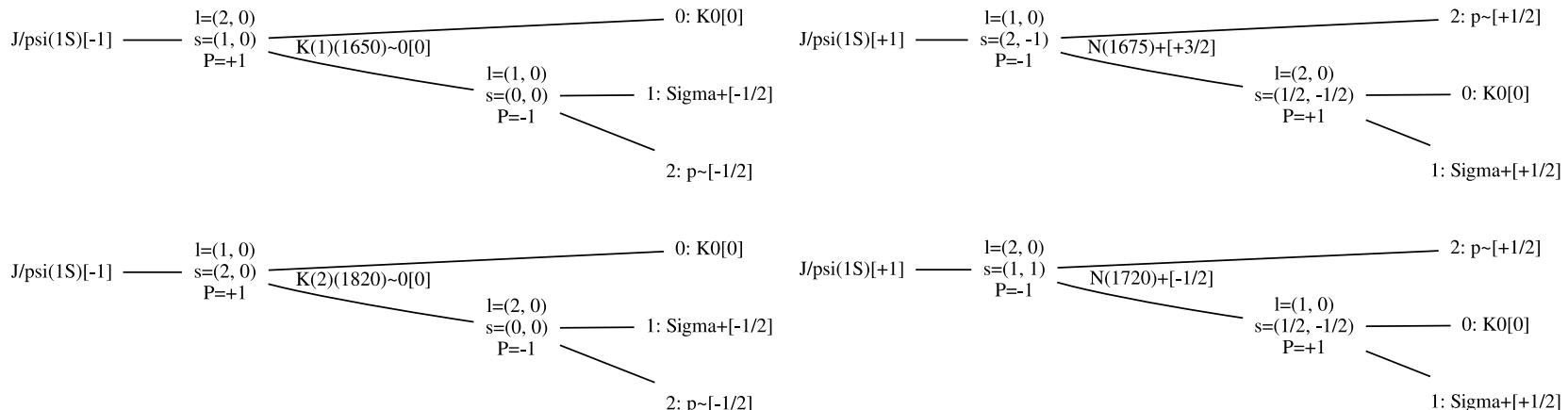
## Automated quantum number conservation rules

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  Open in Colab



# AmpForm

## Symbolic amplitude model formulation

- Library of **spin formalisms** and **dynamics**
- Formulate QRules' state transitions as an amplitude model
- Models are formulated as **algebraic expressions** (SymPy CAS)

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

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

```
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

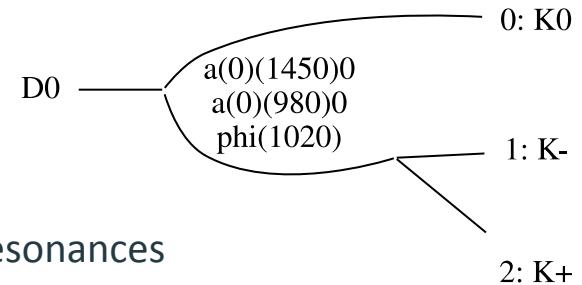
## Symbolic amplitude model formulation

### Example

Building an amplitude model for  $D^0 \rightarrow K^0 K^- K^+$  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



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

1. Select parametrization for each resonance
2. AmpForm takes care of spin (helicity formalism)
3. Resulting amplitude model expressed symbolically

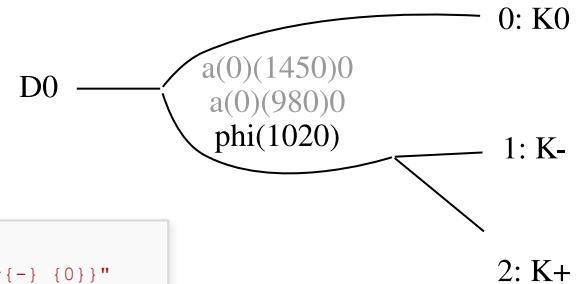
*LaTeX generated  
by the framework!*

# AmpForm

## Symbolic amplitude model formulation

Expression of each amplitude can be further inspected:

```
some_amplitude = model.components[
    R"A_{D^0}{\rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2))} D_{0,0}^0(-\phi_{1+2}, \theta_{1+2}, 0) D_{0,0}^1(-\phi_{1,1+2}, \theta_{1,1+2}, 0)"
]
```



$$\begin{aligned}
 & \frac{C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2))} D_{0,0}^0(-\phi_{1+2}, \theta_{1+2}, 0) D_{0,0}^1(-\phi_{1,1+2}, \theta_{1,1+2}, 0)}{-m_{12}^2 + m_{\phi(1020)}^2 - i m_{\phi(1020)} \Gamma_{1020}(m_{12}^2)} \\
 &= \frac{\sqrt{2} C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{\frac{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)}{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2) + 1}} \cos(\theta_{1,1+2})}{-\frac{i \Gamma_{\phi(1020)} m_{\phi(1020)} B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)) \rho_{122}(m_{12}^2)}{B_1^2(d_{\phi(1020)}^2 q_{10202}^2(m_{\phi(1020)}^2)) \rho_{10202}(m_{\phi(1020)}^2)} - m_{12}^2 + m_{\phi(1020)}^2} \\
 &= \frac{0.0035 \sqrt{m_{12}^2 - 0.97} \cos(\theta_{1,1+2})}{\sqrt{0.33 m_{12}^2 + 1} \left( -m_{12}^2 + 1.0 - \frac{0.27 i (m_{12}^2 - 0.97)^{\frac{3}{2}}}{m_{12} \left( \frac{m_{12}^2}{4} + 0.76 \right)} \right)}
 \end{aligned}$$

*Spin formalism*

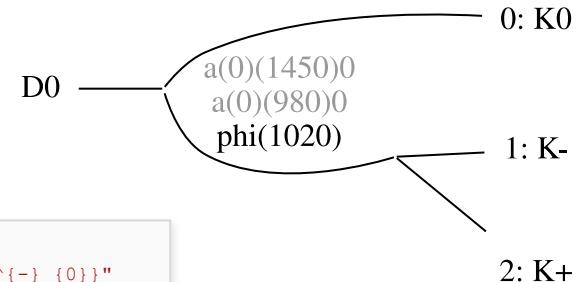
*Dynamics*

# AmpForm

## Symbolic amplitude model formulation

Expression of each amplitude can be further inspected:

```
some_amplitude = model.components[
    R"A_{D^0}{0} \rightarrow K^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^- \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2))} D_{0,0}^0(-\phi_{1+2}, \theta_{1+2}, 0) D_{0,0}^1(-\phi_{1,1+2}, \theta_{1,1+2}, 0)"
]
```



$$\frac{C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2))} D_{0,0}^0(-\phi_{1+2}, \theta_{1+2}, 0) D_{0,0}^1(-\phi_{1,1+2}, \theta_{1,1+2}, 0)}{-m_{12}^2 + m_{\phi(1020)}^2 - i m_{\phi(1020)} \Gamma_{1020}(m_{12}^2)}$$

$$\begin{aligned}
&= \frac{\sqrt{2} C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{\frac{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)}{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2) + 1}} \cos(\theta_{1,1+2})}{-\frac{i \Gamma_{\phi(1020)} m_{\phi(1020)} B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)) \rho_{122}(m_{12}^2)}{B_1^2(d_{\phi(1020)}^2 q_{10202}^2(m_{\phi(1020)}^2)) \rho_{10202}(m_{\phi(1020)}^2)} - m_{12}^2 + m_{\phi(1020)}^2} \\
&= \frac{0.0035 \sqrt{m_{12}^2 - 0.97} \cos(\theta_{1,1+2})}{\sqrt{0.33 m_{12}^2 + 1} \left( -m_{12}^2 + 1.0 - \frac{0.27i(m_{12}^2 - 0.97)^{\frac{3}{2}}}{m_{12} \left( \frac{m_{12}^2}{4} + 0.76 \right)} \right)}
\end{aligned}$$

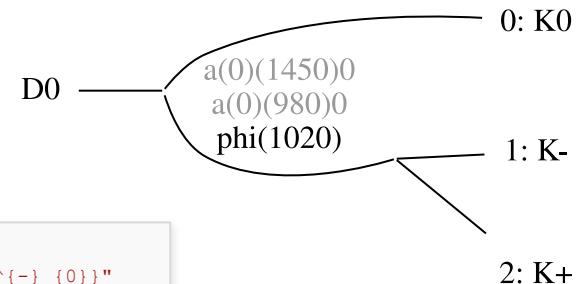
CAS can evaluate  
sub-expressions

# AmpForm

## Symbolic amplitude model formulation

Expression of each amplitude can be further inspected:

```
some_amplitude = model.components[
    R"A_{D^0}{0} \rightarrow K^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^- \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2))} D_{0,0}^0(-\phi_{1+2}, \theta_{1+2}, 0) D_{0,0}^1(-\phi_{1,1+2}, \theta_{1,1+2}, 0)"
]
```



$$\begin{aligned}
 & \frac{C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2))} D_{0,0}^0(-\phi_{1+2}, \theta_{1+2}, 0) D_{0,0}^1(-\phi_{1,1+2}, \theta_{1,1+2}, 0)}{-m_{12}^2 + m_{\phi(1020)}^2 - i m_{\phi(1020)} \Gamma_{1020}(m_{12}^2)} \\
 &= \frac{\sqrt{2} C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^+ K_0^-} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{\frac{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)}{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2) + 1}} \cos(\theta_{1,1+2})}{-\frac{i \Gamma_{\phi(1020)} m_{\phi(1020)} B_1^2(d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)) \rho_{122}(m_{12}^2)}{B_1^2(d_{\phi(1020)}^2 q_{10202}^2(m_{\phi(1020)}^2)) \rho_{10202}(m_{\phi(1020)}^2)} - m_{12}^2 + m_{\phi(1020)}^2} \\
 &= \frac{0.0035 \sqrt{m_{12}^2 - 0.97} \cos(\theta_{1,1+2})}{\sqrt{0.33 m_{12}^2 + 1} \left( -m_{12}^2 + 1.0 - \frac{0.27 i (m_{12}^2 - 0.97)^{\frac{3}{2}}}{m_{12} \left( \frac{m_{12}^2}{4} + 0.76 \right)} \right)}
 \end{aligned}$$

AmpForm provides  
 suggested parameter values

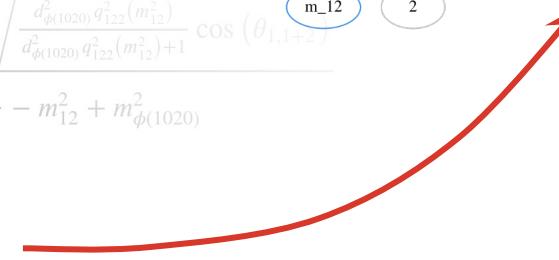
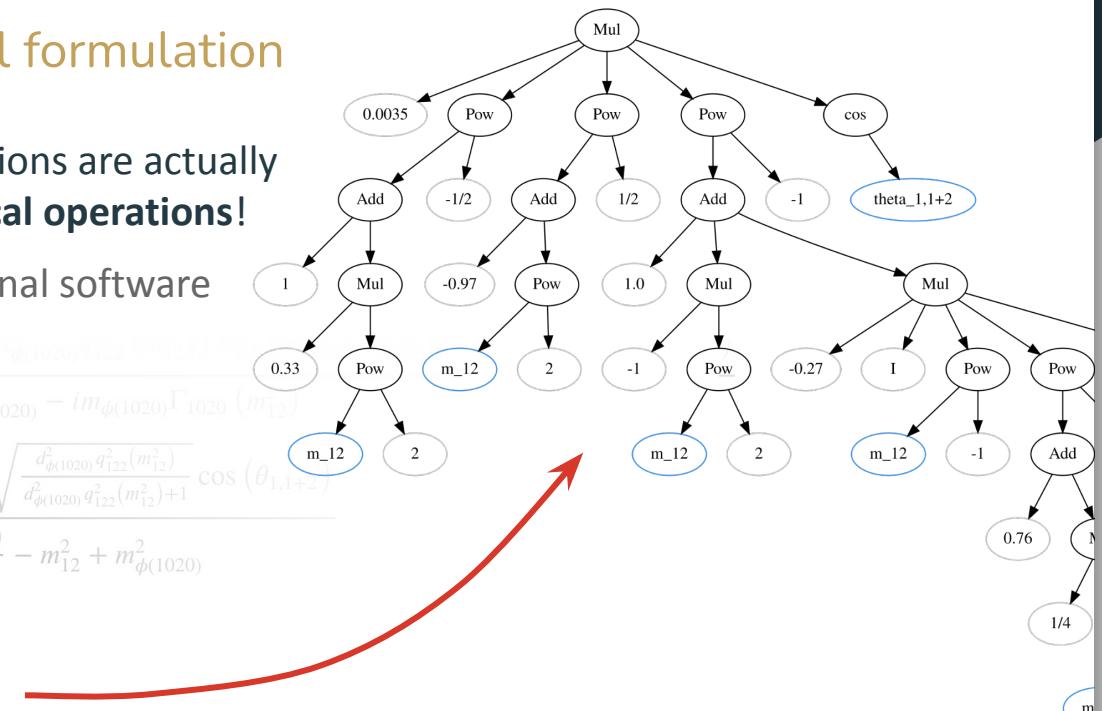
# AmpForm

## Symbolic amplitude model formulation

Original motivation: these expressions are actually  
**trees of fundamental mathematical operations!**

⇒ Template for faster computational software

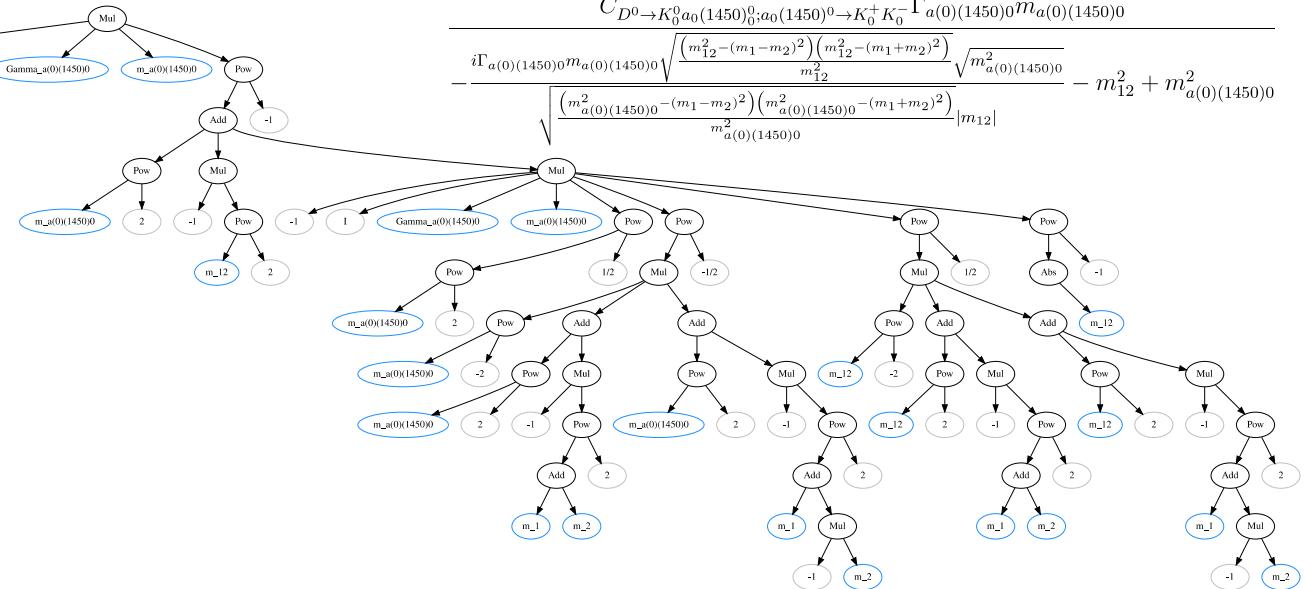
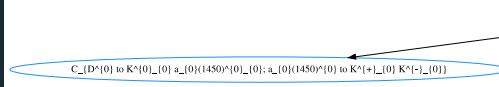
$$\begin{aligned}
 & \frac{\sqrt{2} C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^- K_0^+} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{\frac{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)}{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2) + 1}} \cos(\theta_{1,1+2})}{-m_{12}^2 + m_{\phi(1020)}^2 - i m_{\phi(1020)} \Gamma_{1020}(m_{12})} \\
 & = \frac{\sqrt{2} C_{D^0 \rightarrow K_0^0 \phi(1020)_0; \phi(1020) \rightarrow K_0^- K_0^+} \Gamma_{\phi(1020)} m_{\phi(1020)} \sqrt{\frac{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)}{d_{\phi(1020)}^2 q_{122}^2(m_{12}^2) + 1}} \cos(\theta_{1,1+2})}{\frac{i \Gamma_{\phi(1020)} m_{\phi(1020)} B_1^2 (d_{\phi(1020)}^2 q_{122}^2(m_{12}^2)) \rho_{122}(m_{12}^2)}{B_1^2 (d_{\phi(1020)}^2 q_{10202}^2(m_{\phi(1020)}^2)) \rho_{10202}(m_{\phi(1020)}^2)} - m_{12}^2 + m_{\phi(1020)}^2} \\
 & = \frac{0.0035 \sqrt{m_{12}^2 - 0.97} \cos(\theta_{1,1+2})}{\sqrt{0.33 m_{12}^2 + 1} \left( -m_{12}^2 + 1.0 - \frac{0.27 i (m_{12}^2 - 0.97)^{\frac{3}{2}}}{m_{12} \left( \frac{m_{12}^2}{4} + 0.76 \right)} \right)}
 \end{aligned}$$



# AmpForm

## Symbolic amplitude model formulation

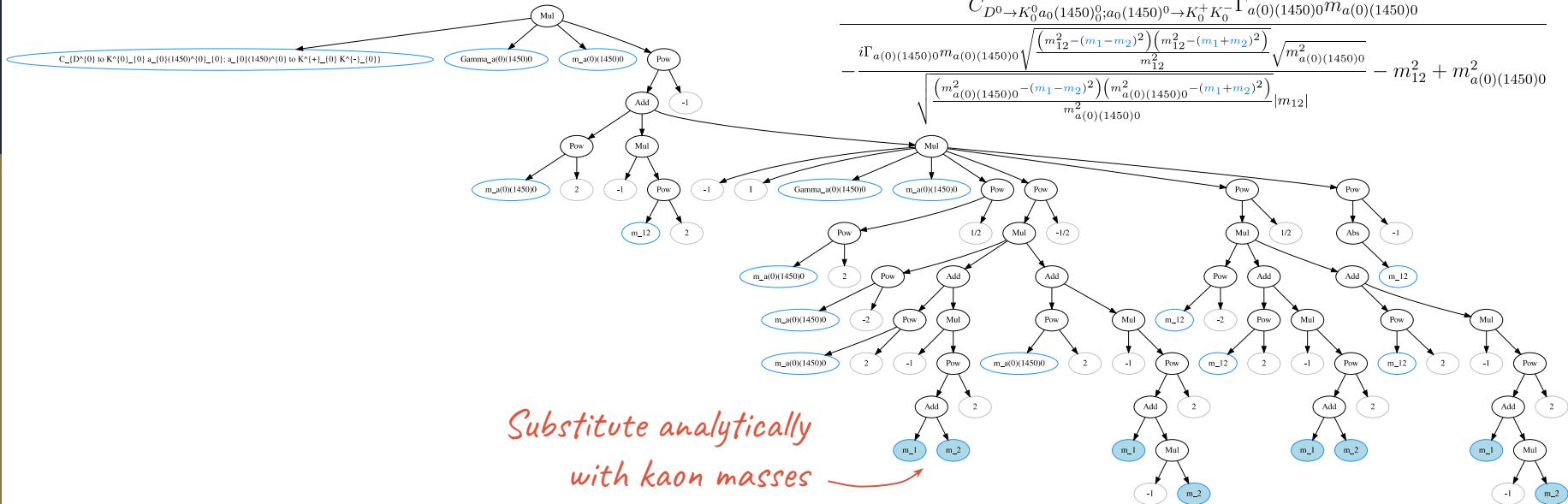
## Expression tree modification example: substitute stable masses



# AmpForm

## Symbolic amplitude model formulation

Expression tree modification example: substitute stable masses

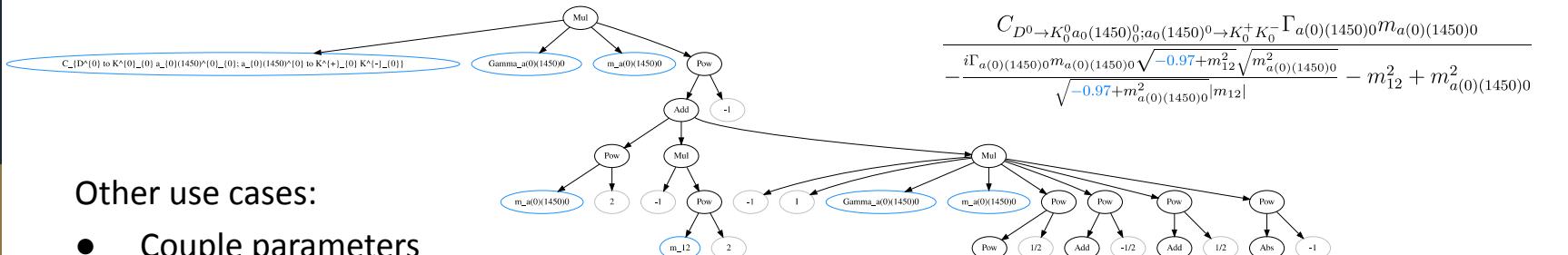


# AmpForm

## Symbolic amplitude model formulation

**Expression tree modification example:** substitute stable masses

$$C_{D^0 \rightarrow K^0 a_0(1450)^0; a_0(1450)^0 \rightarrow K^+ K^-} = a_0(1450)^0 \cdot a_0(1450)^0 \cdot \Gamma_{a_0(1450)} \cdot m_{a_0(1450)}$$



Other use cases:

- Couple parameters
- Insert background expressions
- ...
- Substitute fixed parameters
- Collapse constant sub-trees (caching)

*CAS collapses sub-trees*

⇒ Transparency, flexibility and optimized computations

# AmpForm

## Symbolic amplitude model formulation

**New feature:** helicity angles computed symbolically from four-momenta

```
model.kinematic_variables[theta_1_12].doit()
```

$$\theta \left( \mathbf{B}_z \left( \frac{|\vec{p_{12}}|}{E(p_{12})} \right) \mathbf{R}_y (-\theta(p_{12})) \mathbf{R}_z (-\phi(p_{12})) p_1 \right) \longrightarrow \text{Easier to implement spin alignment etc.}$$

```
model.kinematic_variables[theta_1_12].doit()
```


[Launch binder](#)

[Open in Colab](#)

$$\text{acos} \left( \frac{\mathbf{B}_z \left( \frac{\sqrt{\sum_{\text{axis1}} (p_{12})[:,1:]^2}}{(p_{12})[:,0]} \right) \mathbf{R}_y \left( -\text{acos} \left( \frac{(p_{12})[:,3]}{\sqrt{\sum_{\text{axis1}} (p_{12})[:,1:]^2}} \right) \right) \mathbf{R}_z (-\text{atan2}((p_{12})[:,2], (p_{12})[:,1])) p_1[:,3]}{\sqrt{\sum_{\text{axis1}} \mathbf{B}_z \left( \frac{\sqrt{\sum_{\text{axis1}} (p_{12})[:,1:]^2}}{(p_{12})[:,0]} \right) \mathbf{R}_y \left( -\text{acos} \left( \frac{(p_{12})[:,3]}{\sqrt{\sum_{\text{axis1}} (p_{12})[:,1:]^2}} \right) \right) \mathbf{R}_z (-\text{atan2}((p_{12})[:,2], (p_{12})[:,1])) p_1[:,1:]^2}} \right)$$

# TensorWaves

Fit and generate data with multiple computational back-ends

Python is slow, but optimized backend libraries offer:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation

# TensorWaves

Fit and generate data with multiple computational back-ends

Python is slow, but optimized backend libraries offer:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation

$$\mathbf{c} = \mathbf{a} \mathbf{b} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}$$

```
c = []
for i in range(len(a)):
    c.append(a[i]*b[i])
```



```
c = a * b
```



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

*Data arrays*

*Heavy lifting by  
optimized backend*

# TensorWaves

Fit and generate data with multiple computational back-ends

Python is slow, but optimized backend libraries offer:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation

Data arrays

```
def my_expression(x, y, z):  
    return x + y * z
```



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

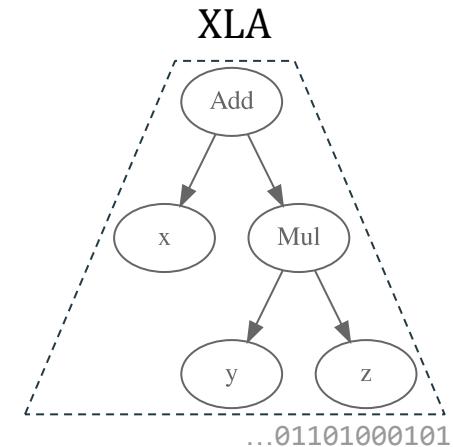
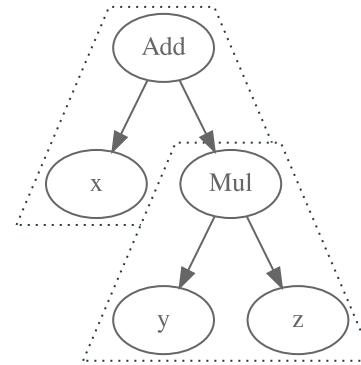


# TensorWaves

Fit and generate data with multiple computational back-ends

Python is slow, but optimized backend libraries offer:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation



```
def my_expression(x, y, z):  
    return x + y * z
```

Data arrays



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

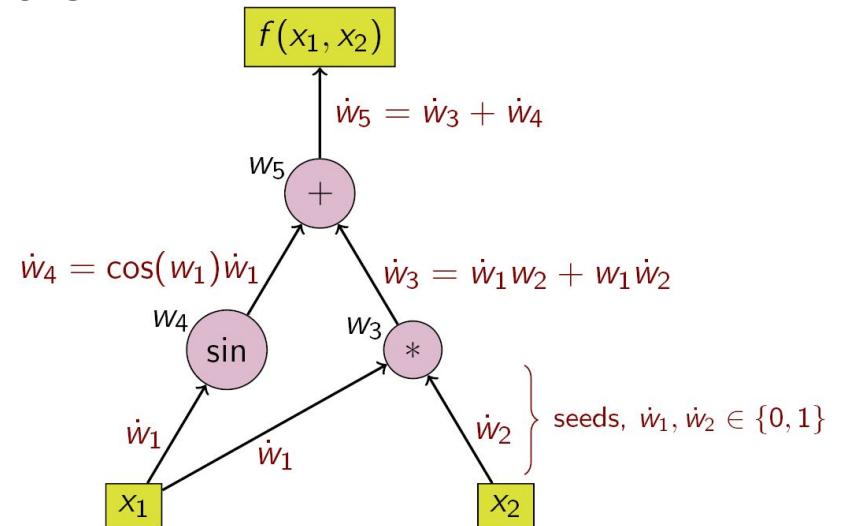


# TensorWaves

Fit and generate data with multiple computational back-ends

Python is slow, but optimized backend libraries offer:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation



$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

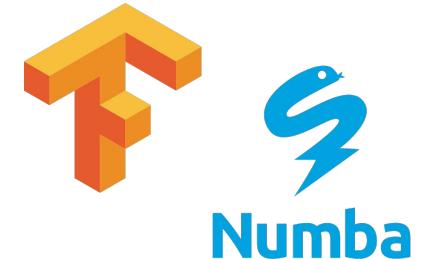
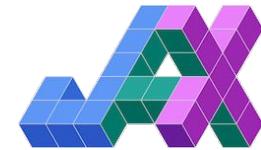
# TensorWaves

Fit and generate data with multiple computational back-ends

Python is slow, but optimized backend libraries offer:

- Vectorization
- Just-in-time compilation
- XLA (Accelerated Linear Algebra)
- Automatic differentiation

→ Since a few years, several optimized packages  
from the ML and data science communities



# TensorWaves

Fit and generate data with multiple 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")
minuit2 = Minuit2(callback=CSVSummary("fit_traceback.csv"))
fit_result = minuit2.optimize(estimator, initial_parameters)
```

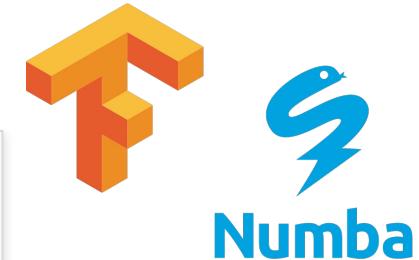
 Open in Colab

# TensorWaves

Fit and generate data with multiple computational back-ends

## Why is this nice?

- Outsource heavy computations to specialized packages from the machine learning and data science
- Get support for GPUs, multithreading, etc. for free
- Separate physics from the ‘number crunching’
- Easily integrate other upcoming back-ends



Any symbolic input

```
function = create_parametrized_function(expression, parameter_defaults, backend="jax")
estimator = UnbinnedNLL(function, data, phsp, backend="jax")
minuit2 = Minuit2(callback=CSVSummary("fit_traceback.csv"))
fit_result = minuit2.optimize(estimator, initial_parameters)
```

 Open in Colab

# TensorWaves

Fit and generate data with multiple computational back-ends

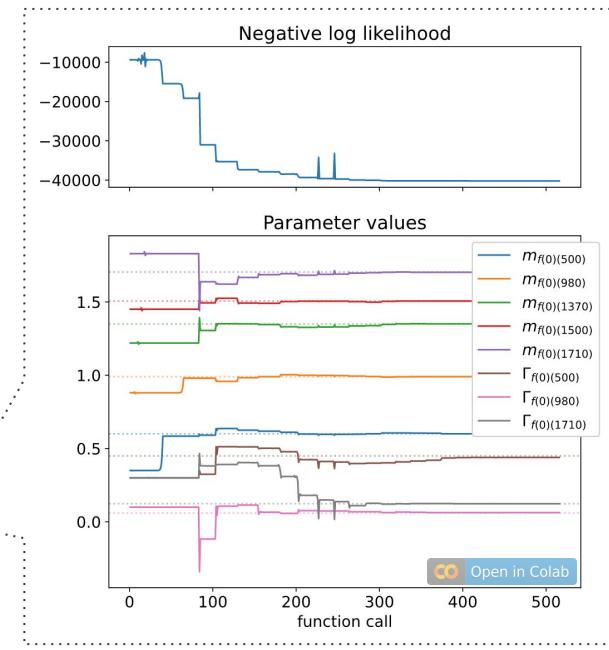
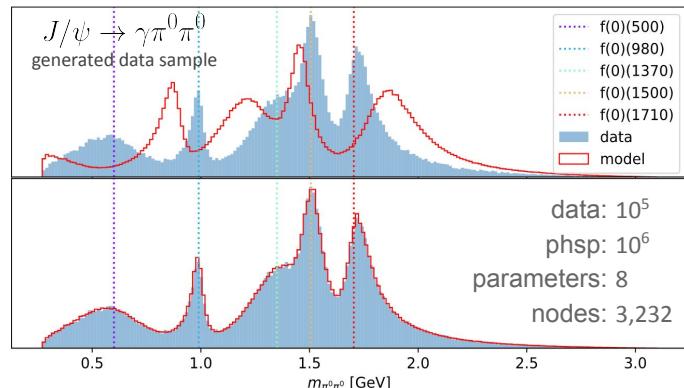
**Does it work?** JAX+Minuit2 example benchmark:

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**



# Documentation

Narrow the gap between code and theory

All packages come with well-maintained websites:

- Extensive explanations of implemented physics
- Thoroughly integrated with codebase
- Continuously tested: links and code examples won't break
- Run demos directly from the browser
- Kind of an **interactive book**  
(see [Executable Book Project](#))

Search the docs ...     

Contents 

Physics

- Partial wave expansion
- Transition operator
- Ensuring unitarity
- Lorentz-invariance

Production processes

- Pole parametrization
- Implementation
- Interactive visualization

Installation

Usage

Formulate amplitude model

Inspect model interactively

Helicity versus canonical

Custom dynamics

Analytic continuation

K-matrix

API

Changelog

Coming features

Help developing

RELATED PROJECTS

QRules

TensorWaves

PWA Pages

COMPWA ORGANIZATION

Website

GitHub Repositories

About

**Launch interactive examples**

^ Pole parametrization

After all these matrix definitions, the final challenge is to choose a correct parametrization for the elements of  $\mathbf{K}$  and  $\mathbf{P}$  that accurately describes the resonances we observe.<sup>[3]</sup> There are several choices, but a common one is the following summation over the poles  $R$ :<sup>[4]</sup>

$$K_{ij} = \sum_R \frac{g_{R,i} g_{R,j}}{m_R^2 - s} + c_{ij} \quad (14)$$
$$\hat{K}_{ij} = \sum_R \frac{g_{R,i}(s) g_{R,j}(s)}{(m_R^2 - s) \sqrt{\rho_i \rho_j}} + \hat{c}_{ij}$$

With  $\rho_i$  some optional background characterization and  $g_{R,i}$  the **residue functions**. The residue functions are often further expressed as:

$$g_{R,i} = \gamma_{R,i} \sqrt{m_R \Gamma_{R,i}^0} \quad (15)$$
$$g_{R,i}(s) = \gamma_{R,i} \sqrt{m_R \Gamma_{R,i}(s)}$$

with  $\gamma_{R,i}$  some **real** constants and  $\Gamma_{R,i}^0$  the **partial width** of each pole. In the Lorentz-invariant form, the fixed width  $\Gamma^0$  is replaced by an "energy dependent" CoupledWidth  $\Gamma(s)$ .<sup>[5]</sup> The **width** for each pole can be computed as  $\Gamma_R^0 = \sum_i \Gamma_{R,i}^0$ .

The production vector  $\mathbf{P}$  is commonly parameterized

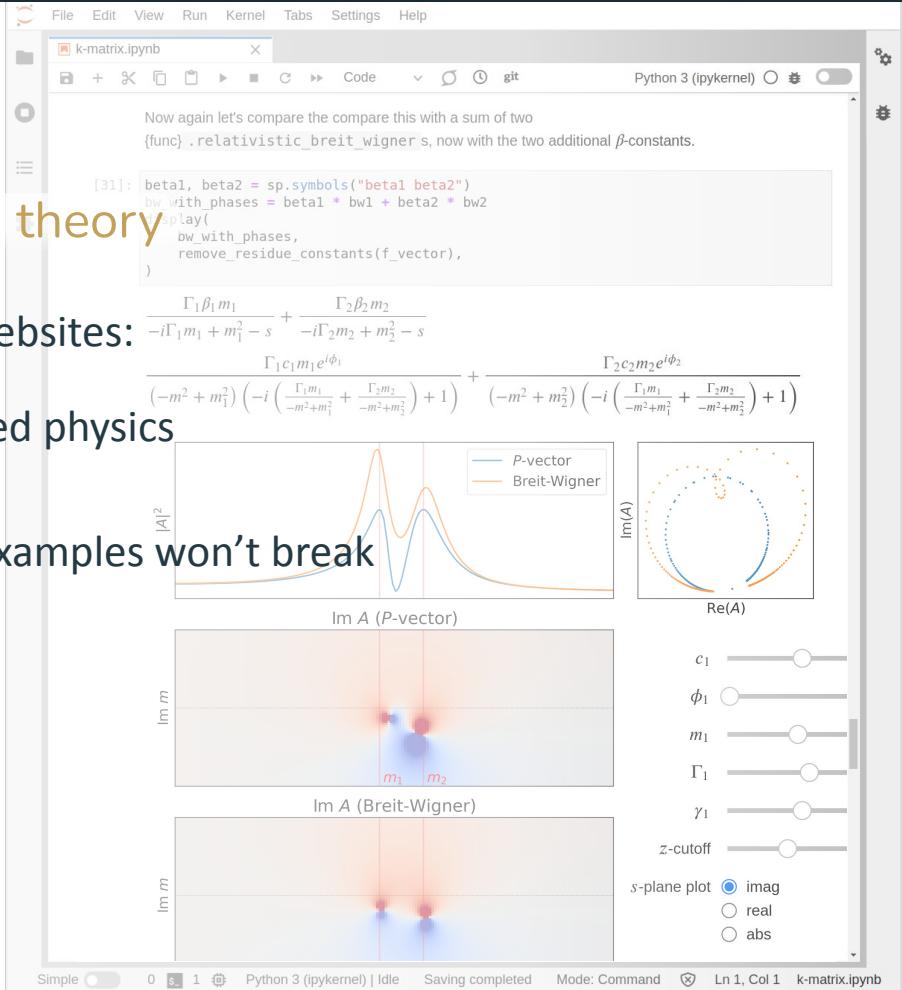
[4] Eqs. (75-78)

# Documentation

## Narrow the gap between code and theory

All packages come with well-maintained websites:

- Extensive explanations of implemented physics
- Thoroughly integrated with codebase
- Continuously tested: links and code examples won't break
- Run demos directly from the browser
- Kind of an **interactive book**  
(see [Executable Book Project](#))



# PWA Software Pages

Interactive knowledge-base for PWA theory and software

⇒ Spin-off project: PWA Pages ([pwa.rtfd.io](https://pwa.rtfd.io))

- Intended as a guide through the main ingredients of Partial Wave Analysis
- Easily navigate to literature and existing PWA software\*
- Currently skeletal: infrastructure is there and easy contribute to
- No need to know HTML, CSS, etc.

Welcome to include or reference your PWA project!

\*See also [HSF DAWG amplitude analysis benchmark meetings](#)

Partial Wave Analysis



## Amplitude analysis projects

The following frameworks or projects are specifically designed to do amplitude analysis. The table is built from [this YAML file](#) and can be updated [here](#).

Project	Since	Lastest commit	C++	Python	Cuda
AmpGen	CLEO / LHCb	2018	11/2021	✓	
AmpTools		2011	11/2021	✓	✓
ComPWA (C++)		2012	06/2020	✓	
ComPWA project		2020	12/2021		✓
• QRules					
• AmpForm					
• Tensor Waves					
cFit		05/2018	✓		
SETUPWA	BESIII	2011		✓	
PWA Frameworks					
HAMMER		2016	10/2021	✓	✓
Ipanema		2017	03/2019	✓	✓
Laura++	LHCb	2013		✓	
Mint2		2016	01/2020	✓	
Pawian	BESIII / PANDA	2010		✓	
PyPWA	JLab	2014	07/2021		✓
ROOTPWA	COMPASS	2009	06/2020	✓	✓
TensorFlowAnalysis	LHCb	2016	11/2021		✓
• AmpliTF					
• TFA2					
TF-PWA	BESIII / LHCb	2019	11/2021		✓

# Code quality and Developer Experience

## Long-term maintainability:

- Get involved with just a few commands
- Automated coding conventions
- Reproducibility in each commit:  
pinned dependencies and permanent links
- Continuous benchmark monitoring
- Self-documenting code (typed libraries)
- Documentation as integration tests

Anyone is welcome to contribute!

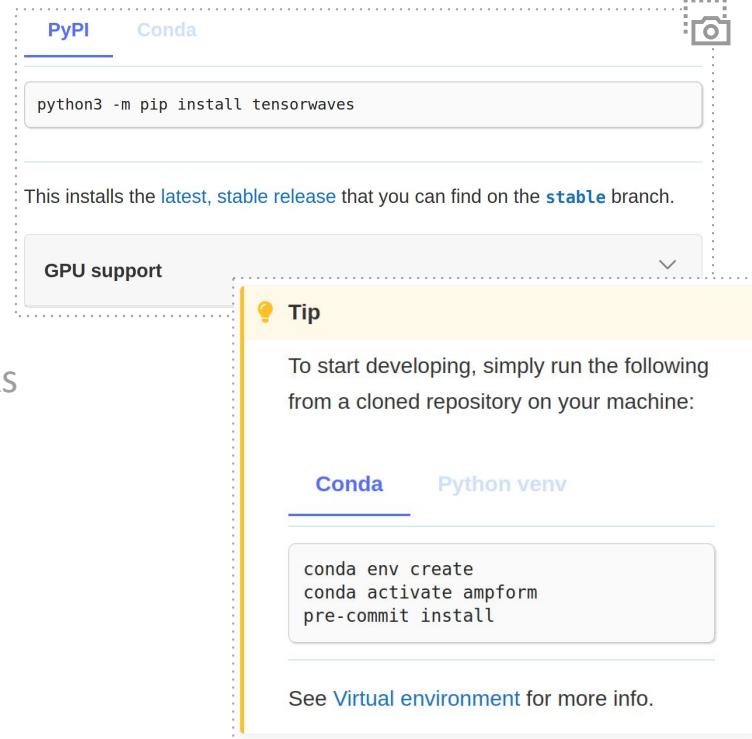
[compwa-org.rtfdf.io/en/stable/develop.html](https://compwa-org.rtfdf.io/en/stable/develop.html)

# Code quality and Developer Experience

## Long-term maintainability:

- Get involved with just a few commands
- Automated coding conventions
- Reproducibility in each commit:  
pinned dependencies and permanent links
- Continuous benchmark monitoring
- Self-documenting code (typed libraries)
- Documentation as integration tests

Anyone is welcome to contribute!  
[compwa-org.rtfdf.io/en/stable/develop.html](https://compwa-org.rtfdf.io/en/stable/develop.html)



The screenshot shows a section of the ComPWA documentation. It includes two tabs at the top: "PyPI" (selected) and "Conda". Below the tabs is a command-line interface (CLI) box containing the command: `python3 -m pip install tensorwaves`. A note below the command states: "This installs the latest, stable release that you can find on the [stable](#) branch." To the right of the CLI box is a "GPU support" section with a dropdown arrow. Below the GPU support section is a yellow "Tip" box with a lightbulb icon. The tip text reads: "To start developing, simply run the following from a cloned repository on your machine:" followed by the command: `conda env create  
conda activate ampform  
pre-commit install`. At the bottom of the tip box is the text: "See [Virtual environment](#) for more info." There is also a camera icon in the top right corner of the main content area.

# Code quality and Developer Experience

## Long-term maintainability:

- Get involved with just a few commands
- Automated coding conventions
- Reproducibility in each commit:  
pinned dependencies and permanent links
- Continuous benchmark monitoring
- Self-documenting code (typed libraries)
- Documentation as integration tests

Anyone is welcome to contribute!

[compwa-org.rtfid.io/en/stable/develop.html](https://compwa-org.rtfid.io/en/stable/develop.html)

\$ pre-commit run -a	
Check hooks apply to the repository.....	Passed
Check for useless excludes.....	Passed
Check python ast.....	Passed
Check for case conflicts.....	Passed
Check JSON.....	Passed
Check for merge conflicts.....	Passed
Check Toml.....	Passed
Check vcs permalinks.....	Passed
Check Yaml.....	Passed
Debug Statements (Python).....	Passed
Fix End of Files.....	Passed
Mixed line ending.....	Passed
Trim Trailing Whitespace.....	Passed
Check developer config files in the repository.....	Passed
Set nbformat minor version to 4 and remove cell IDs.....	Passed
Format setup.cfg.....	Passed
Check whether notebook contains a pip install line.....	Passed
black.....	Passed
blacken-docs.....	Passed
cspell.....	Passed
Check .editorconfig rules.....	Passed
isort.....	Passed
markdownlint.....	Passed
nbqa-black.....	Passed
nbqa-flake8.....	Passed
nbqa-isort.....	Passed
nbqa-pyupgrade.....	Passed
nbstripout.....	Passed
prettier.....	Passed
pyright.....	Passed
pyupgrade.....	Passed
flake8.....	Passed
pylint.....	Passed

# Code quality and Developer Experience

## Long-term maintainability:

- Get involved with just a few commands
- Automated coding conventions
- Reproducibility in each commit:  
pinned dependencies and permanent links
- Continuous benchmark monitoring
- Self-documenting code (typed libraries)
- Documentation as integration tests

Anyone is welcome to contribute!

[compwa-org.rtfid.io/en/stable/develop.html](https://compwa-org.rtfid.io/en/stable/develop.html)

☰ py3.8.txt (f05903f) ↔ py3.8.txt (8135732) .../constraints

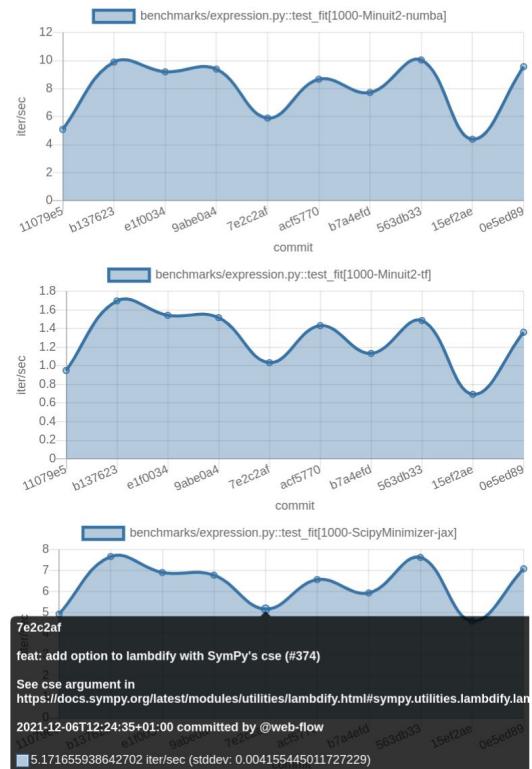
```
nbformat==5.1.3          nbformat==5.1.3
nest-asyncio==1.5.         nest-asyncio==1.5.1
nodeenv==1.6.0            nodeenv==1.6.0
notebook==6.4.5           notebook==6.4.5
- numpy==1.20.3           + numpy==1.21.4
packaging==21.2            packaging==21.2
pandocfilters==1.5        pandocfilters==1.5.0
parso==0.8.2              parso==0.8.2
particle==0.16.2          particle==0.16.2
pathspec==0.9.0            pathspec==0.9.0
pep517==0.12.0             pep517==0.12.0
pep8-naming==0.12.         pep8-naming==0.12.1
pexpect==4.8.0              pexpect==4.8.0
pickleshare==0.7.5         pickleshare==0.7.5
pillow==8.4.0               pillow==8.4.0
pip-tools==6.4.0            pip-tools==6.4.0
platformdirs==2.4.          platformdirs==2.4.0
pluggy==1.0.0                pluggy==1.0.0
pre-commit==2.15.0          pre-commit==2.15.0
prometheus-client=           prometheus-client==0.12.0
prompt-toolkit==3.           prompt-toolkit==3.0.22
ptyprocess==0.7.0            ptyprocess==0.7.0
py==1.11.0                  py==1.11.0
pybtex==0.24.0               pybtex==0.24.0
pybtex_docutils==1.0.1       pybtex_docutils==1.0.1
```

# Code quality and Developer Experience

## Long-term maintainability:

- Get involved with just a few commands
- Automated coding conventions
- Reproducibility in each commit:  
pinned dependencies and permanent links
- Continuous benchmark monitoring
- Self-documenting code (typed libraries)
- Documentation as integration tests

Anyone is welcome to contribute!  
[compwa-org.rtfid.io/en/stable/develop.html](https://compwa-org.rtfid.io/en/stable/develop.html)



# Code quality and Developer Experience

## Long-term maintainability:

- Get involved with just a few commands
- Automated coding conventions
- Reproducibility in each commit:  
pinned dependencies and permanent links
- Continuous benchmark monitoring
- Self-documenting code (typed libraries)
- Documentation as integration tests

Anyone is welcome to contribute!  
[compwa-org.rtfd.io/en/stable/develop.html](https://compwa-org.rtfd.io/en/stable/develop.html)

The screenshot shows a detailed view of a Python documentation page for the `AmpForm` library. The left sidebar contains a navigation tree with sections like Installation, Usage, Modify amplitude model, Inspect model interactively, Helicity versus canonical, Dynamics, Bibliography, API, dynamics, builder, kmatrix, helicity, sympy, kinematics, Changelog, Upcoming features, Help developing, RELATED PROJECTS, QRules, TensorWaves, PWA Pages, COMPWA ORGANIZATION, Website, GitHub Repositories, and About. The main content area displays code snippets and their explanations. One snippet is for the `EnergyDependentWidth` class, which calculates a mass-dependent width based on pole position and phase space factor. Another snippet is for the `PhaseSpaceFactor` class, which provides a standard phase-space factor using `BreakupMomentumSquared`. Both snippets include source links.

```
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, evaluate_pole: [str, float] = None, evaluate: bool = False) → Expression
    [source]
```

Bases: `ampform.sympy.UnevaluatedExpression`

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

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:

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

where  $B_L^2$  is defined by (1),  $q$  is defined by (2), and  $\rho$  is (by default) defined by (4).

`phsp_factor: PhaseSpaceFactorProtocol`

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

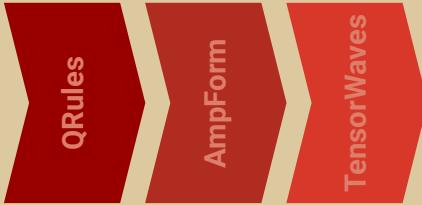
Standard phase-space factor, using `BreakupMomentumSquared()`.

See PDG2020, §Resonances, p.4, Equation (49.8).

# Summary — Key ComPWA features

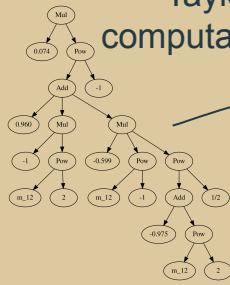
## Modularity

Separate libraries that facilitate common procedures in amplitude analysis



## Scalability

Taylor to the growing number of computational frameworks from industry



## Documentation

Narrow the gap between theory and code with well-referenced, interactive documentation

```
sympy.partial_derivative(f, vector_1r[0], sp.Sum)

$$\frac{\Gamma_{1,0}\beta_1\beta_2m_1}{(1-\frac{v_1^2m_1}{s+m_1^2})(-s+m_1^2)}$$

```

And when we recompute it with `rel_f`:

```
rel_f = f.vector().lr + kmatrix.NeRelativisticVector().mulat(
    n_poles), n_channels=2
rel_f.vector().lr = symvector.PartialDoitrel().execute(rel_f, s)
```

$$\frac{\Gamma_{1,0}\gamma_{1,0}\beta_1m_1\sqrt{B_1^2(q_0^0(s))}}{v_1^2m_1\Gamma_{1,0}(s)-m_1^2+s}$$

Note that the F-vector approach introduces additional  $\beta$ -coefficients. These can constants can be complex and can introduce phase differences from the production process.

```
f_vector_2r = kmatrix.NeRelativisticFVector.formulate(n_poles=2, n_channels=2)
f_vector = f.vector().lr[0].doit()
```



## Code quality

Academic continuity through careful design, automated QA, and self-documenting code



```
7: 'phi_344_2v3d4'
One labels explained:
code: phi_1j2 : *edge 0** on the 'left' topology, because for this topology, phi_1j2 is the leftmost edge.
code: phi_2v3d4 : *edge 0** +right*, because for this topology, phi_2v3d4 is the rightmost edge.
math: p_0p_2p_3p_4 : *edge 0** +right*, because it decays from the left.
math: p_0p_2p_3p_4 : *edge 4** +right*, because it decays from the right.
- code: phi_4j4v4 : *edge 4** +right*, because it decays from edge 4.
As noted, the top-most parameter 'initial_state' is not listed in the label.
assert_isobar_topology(topology)

def recursive_label(topology: Topology, state_id: int) -> str:
    edge = topology.get_edge(state_id)
    if edge.outgoing_node_id is None:
        label = f"({state_id})"
    else:
        attached_final_state_ids = determine_attached_final_state(
            topology, state_id
        )
        label = "+".join((str(id) for id in attached_final_state_ids))
    if edge.incoming_node_id is None:
        incoming_state_ids = topology.get_edge_ids.ingoing_to_node(
            edge.outgoing_node_id
        )
        state_id = next(iter(incoming_state_ids))
    if state_id not in topology.incoming_edge_ids:
        label += f"+{state_id}"
    return label

label = recursive_label(topology, state_id)
return f"phi_{label}", f"theta_{label}"
```

\*\*Generate an invariant mass label for a state node on a topology.

# Summary — Key ComPWA features

## Modularity

Separate libraries that facilitate common procedures in amplitude analysis



## Documentation

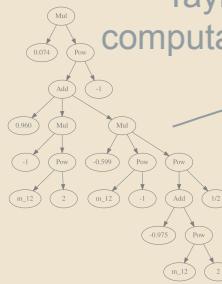
# Narrow the gap between theory and code with well-referenced, interactive documentation



**Thank you for your attention!**

## Scalability

Taylor to the growing number of computational frameworks from industry



## Code quality

# Academic continuity through careful design, automated QA, and self-documenting code

```

ty 301-344-2343
Some labels explained:
  code: ph_1=1 - <code 6> on the "left" topology, because for this topology,
    we have math:  $\text{left} = \text{big}_1 \text{ top}_2$ .
  code: ph_2=1+4 - <code 6> "right", because for this topology,
    we have math:  $\text{right} = \text{big}_2 \text{ top}_1$ . The label "right" always from edge
     $\text{big}_2$ , and "left" always from edge  $\text{big}_1$ .
  math: big_1=1 - <math>\text{big}_1</math> is the label for the edge it decays from edge
     $\text{big}_1$ .
  math: big_2=1 - <math>\text{big}_2</math> is the label for the edge it decays from edge
     $\text{big}_2$ .
As noted, the top-most parent (initial state) is not listed in the label.
...
  def _set_label_topology(topology):
    pass

  def recursive_label_topology(topology, topology_id: int) -> str:
    if topology == topology_id:
      return topology
    else:
      attached, final_state_id = None, None
      for label in topology.state_id:
        if label == f"({state_id})":
          attached, final_state_id = determine_attached_final_state(
            topology, state_id)
        else:
          label = f"({label}, {topology.state_id})"
      incoming = topology.incoming_edge_ids_implementing_node(
        edge, originating_node)
      if incoming:
        state_id = f"({incoming}, {topology.state_id})"
      else:
        state_id = f"({topology.state_id})"
      return f"({label}, {state_id})"

  def recursive_label_topologies(state_id: int) -> str:
    """Generate a recursive label for a state on a topology.
    This is useful for generating labels for states on a topology
    that contains other topologies.
    """
    pass

```