# 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**
$\overline{\text{P}}$ANDA Seminar

# What is the ComPWA project?


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

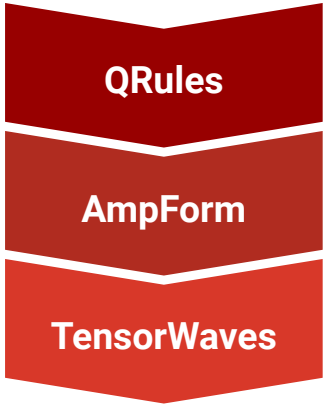
# What does ComPWA aim for?

- **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


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:*

```
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*

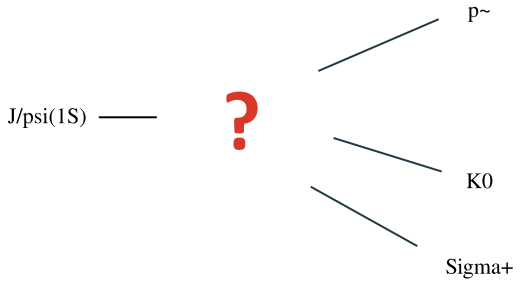
* PDG info computed from the scikit-hep particle package

grules.rtfd.io

# QRules

## Automated quantum number conservation rules

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



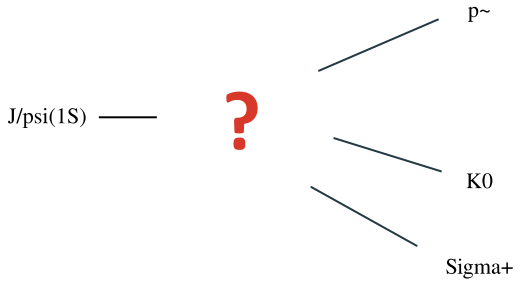
grules.rtfd.io

# 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.)



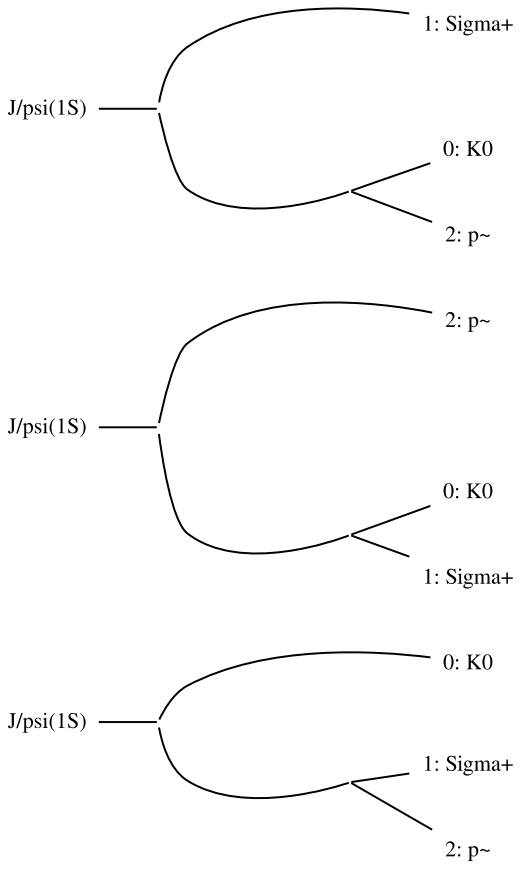
grules.rtfd.io

# 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,



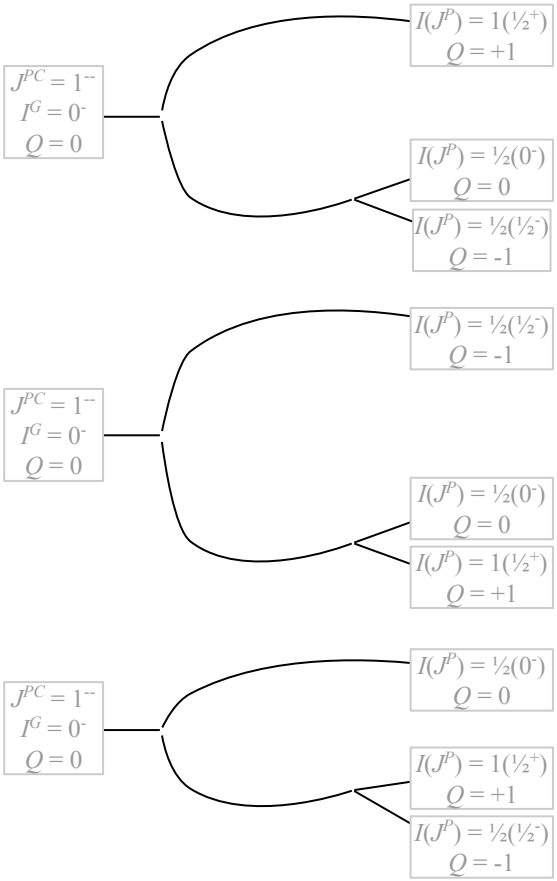
grules.rtfd.io

# 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),



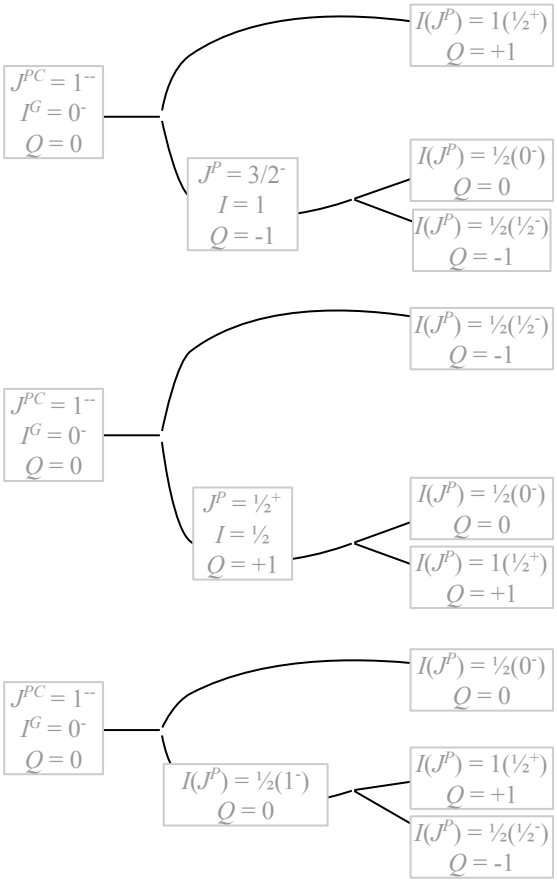
grules.rtfd.io

# 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



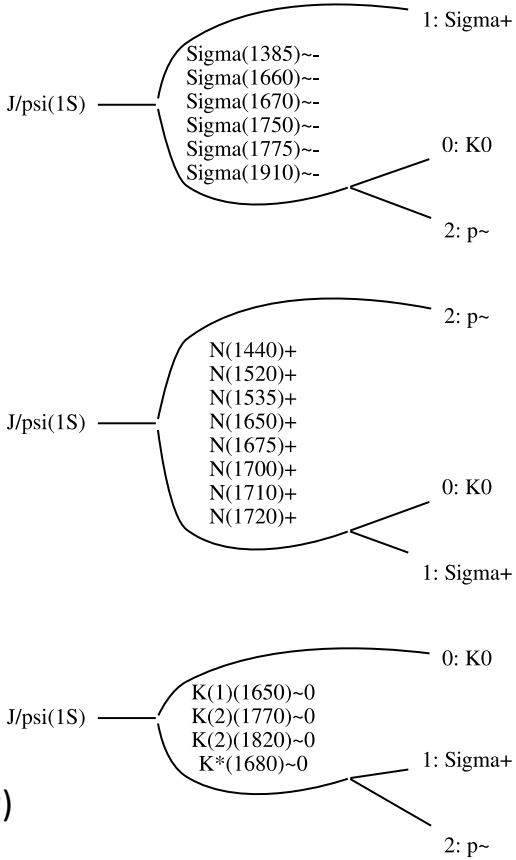
grules.rtfd.io

# 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

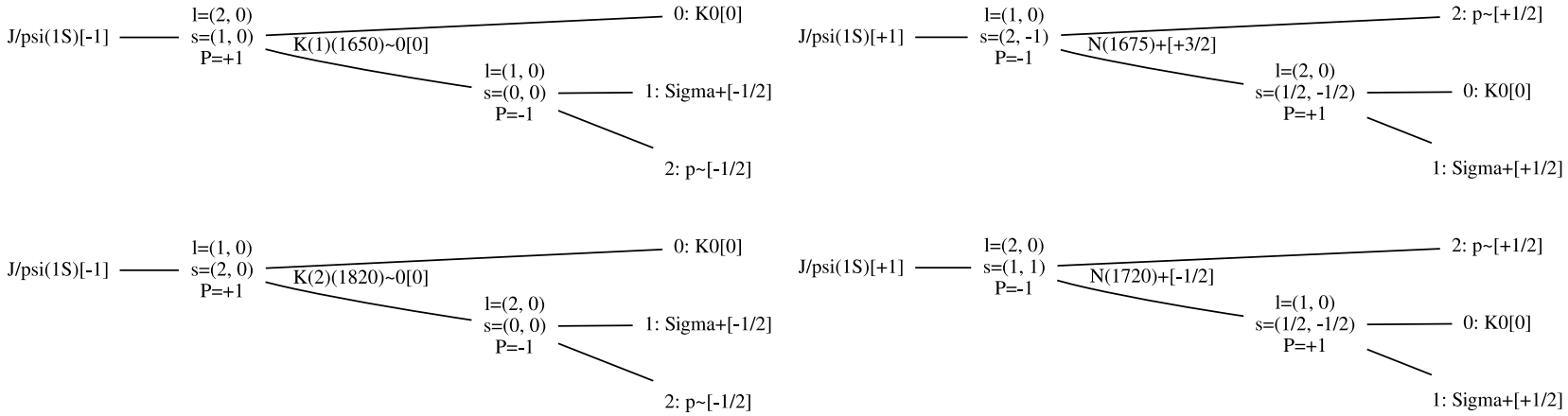
## Automated quantum number conservation rules

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 binder | Open in Colab



ampform.rtfd.io

# 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()
```

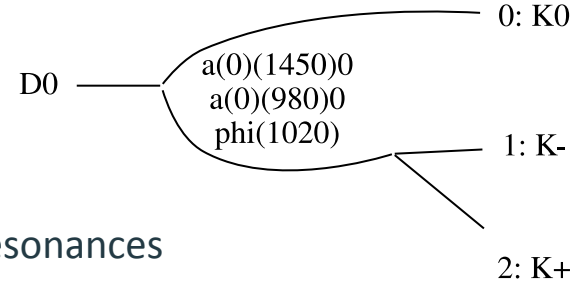
launch binder Open in Colab

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

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




### Example

Building an 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

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

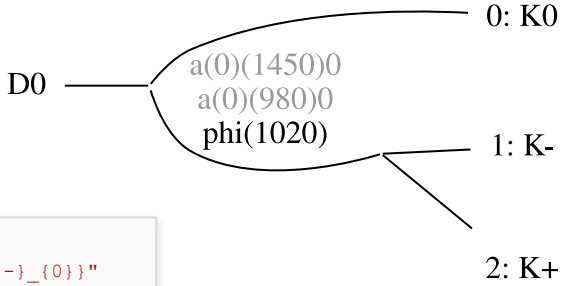
1. Select parametrization for each resonance
2. AmpForm takes care of spin (helicity formalism)
3. Resulting amplitude model expressed symbolically

LaTeX generated by the framework!

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




Expression of each amplitude can be further inspected:

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

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

Spin formalism

Dynamics

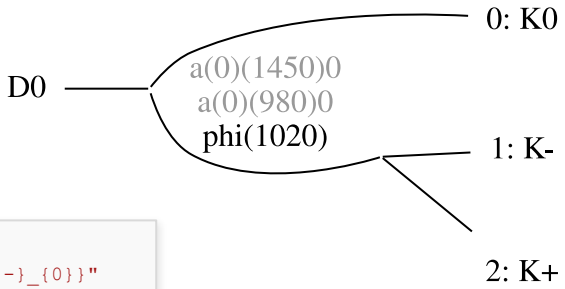
$$=\frac{\sqrt{2}C_{D^0\to K_0^0\phi(1020)_0;\phi(1020)\to K_0^+K_0^-}\Gamma_{\phi(1020)}m_{\phi(1020)}\sqrt{\frac{d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)}{d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)+1}}\cos\left(\theta_{1,1+2}\right)}{-\frac{i\Gamma_{\phi(1020)}m_{\phi(1020)}B_1^2\left(d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)\right)\rho_{122}\left(m^2_{12}\right)}{B_1^2\left(d^2_{\phi(1020)}q^2_{10202}\left(m^2_{\phi(1020)}\right)\right)\rho_{10202}\left(m^2_{\phi(1020)}\right)}-m^2_{12}+m^2_{\phi(1020)}}$$

$$=\frac{0.0035\sqrt{m^2_{12}-0.97}\cos\left(\theta_{1,1+2}\right)}{\sqrt{0.33m^2_{12}+1}\left(-m^2_{12}+1.0-\frac{0.27i\left(m^2_{12}-0.97\right)^{\frac{3}{2}}}{m_{12}\left(\frac{m^2_{12}}{4}+0.76\right)}\right)}$$

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




Expression of each amplitude can be further inspected:

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

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

$$=\frac{\sqrt{2}C_{D^0\to K_0^0\phi(1020)_0;\phi(1020)\to K_0^+K_0^-}\Gamma_{\phi(1020)}m_{\phi(1020)}\sqrt{\frac{d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)}{d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)+1}}\cos\left(\theta_{1,1+2}\right)}{-\frac{i\Gamma_{\phi(1020)}m_{\phi(1020)}B_1^2\left(d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)\right)\rho_{122}\left(m^2_{12}\right)}{B_1^2\left(d^2_{\phi(1020)}q^2_{10202}\left(m^2_{\phi(1020)}\right)\right)\rho_{10202}\left(m^2_{\phi(1020)}\right)}-m^2_{12}+m^2_{\phi(1020)}}$$

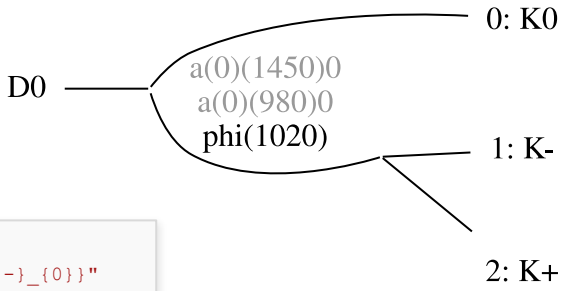
CAS can evaluate sub-expressions

$$=\frac{0.0035\sqrt{m^2_{12}-0.97}\cos\left(\theta_{1,1+2}\right)}{\sqrt{0.33m^2_{12}+1}\left(-m^2_{12}+1.0-\frac{0.27i\left(m^2_{12}-0.97\right)^{\frac{3}{2}}}{m_{12}\left(\frac{m^2_{12}}{4}+0.76\right)}\right)}$$

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




Expression of each amplitude can be further inspected:

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

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

$$=\frac{\sqrt{2}C_{D^0\to K_0^0\phi(1020)_0;\phi(1020)\to K_0^+K_0^-}\Gamma_{\phi(1020)}m_{\phi(1020)}\sqrt{\frac{d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)}{d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)+1}}\cos\left(\theta_{1,1+2}\right)}{-\frac{i\Gamma_{\phi(1020)}m_{\phi(1020)}B_1^2\left(d^2_{\phi(1020)}q^2_{122}\left(m^2_{12}\right)\right)\rho_{122}\left(m^2_{12}\right)}{B_1^2\left(d^2_{\phi(1020)}q^2_{10202}\left(m^2_{\phi(1020)}\right)\right)\rho_{10202}\left(m^2_{\phi(1020)}\right)}-m^2_{12}+m^2_{\phi(1020)}}$$

$$=\frac{0.0035\sqrt{m^2_{12}-0.97}\cos\left(\theta_{1,1+2}\right)}{\sqrt{0.33m^2_{12}+1}\left(-m^2_{12}+1.0-\frac{0.27i\left(m^2_{12}-0.97\right)^{\frac{3}{2}}}{m_{12}\left(\frac{m^2_{12}}{4}+0.76\right)}\right)}$$

AmpForm provides suggested parameter values

ampform.rtfd.io

# AmpForm

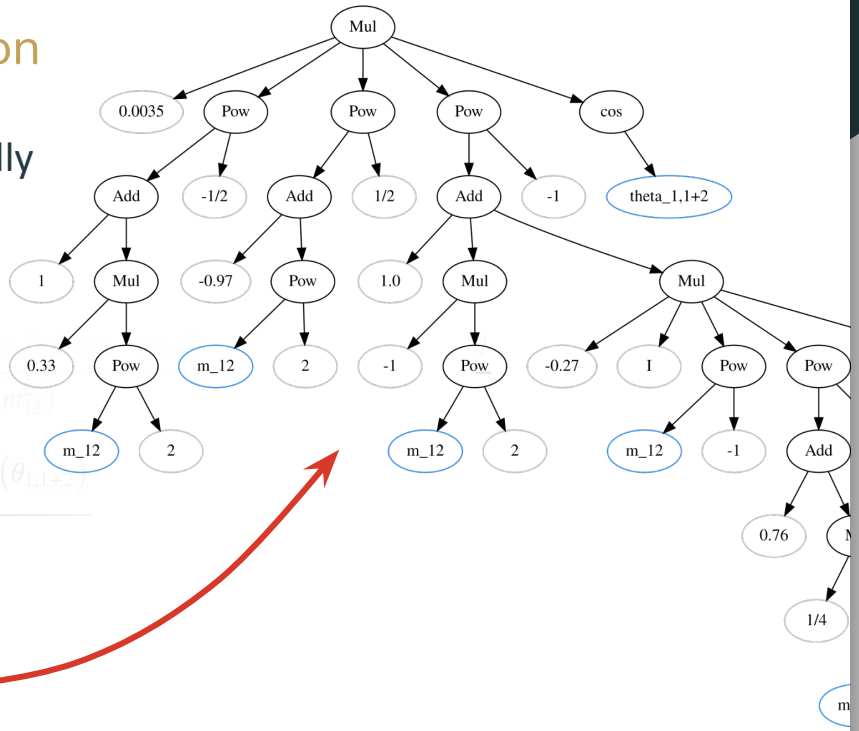
## Symbolic amplitude model formulation

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

⇒ Template for faster computational software

$$= \frac{\sqrt{2}C_{D^0\to K_0^0\phi(1020)_0:\phi(1020)\to K_0^+K_0^-}\Gamma_{\phi(1020)}m_{\phi(1020)}\sqrt{\frac{d^2_{\phi(1020)}q^2_{122}(m^2_{12})}{d^2_{\phi(1020)}q^2_{122}(m^2_{12})+1}}\cos\left(\theta_{1,1+2}\right)}{-\frac{i\Gamma_{\phi(1020)}m_{\phi(1020)}B_1^2\left(d^2_{\phi(1020)}q^2_{122}(m^2_{12})\right)\rho_{122}(m^2_{12})}{B_1^2\left(d^2_{\phi(1020)}q^2_{10202}(m^2_{\phi(1020)})\right)\rho_{10202}(m^2_{\phi(1020)})}-m^2_{12}+m^2_{\phi(1020)}}$$

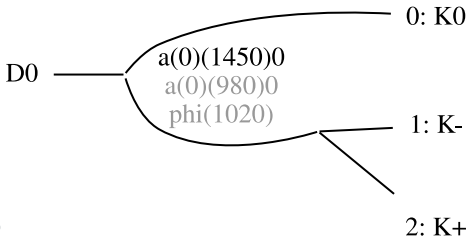
$$= \frac{0.0035\sqrt{m_{12}^2-0.97}\cos\left(\theta_{1,1+2}\right)}{\sqrt{0.33m_{12}^2+1}\left(-m_{12}^2+1.0-\frac{0.27i\left(m_{12}^2-0.97\right)^{\frac{3}{2}}}{m_{12}\left(\frac{m_{12}^2}{4}+0.76\right)}\right)}$$



ampform.rtfd.io

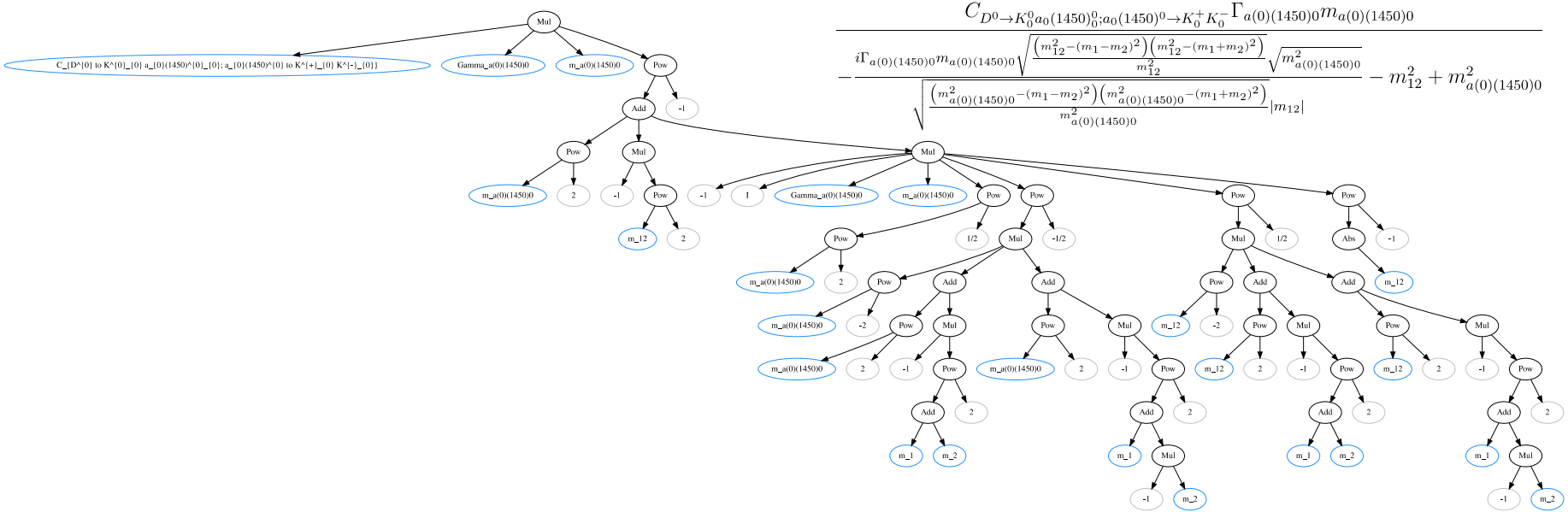
# AmpForm

## Symbolic amplitude model formulation




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

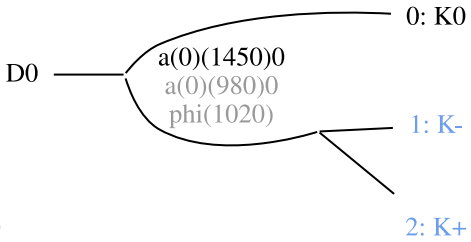
$$\frac{C_{D^0\to K_0^0 a_0(1450)_0^0;a_0(1450)^0\to K_0^+K_0^-}\Gamma_{a(0)(1450)0}m_{a(0)(1450)0}}{-\frac{i\Gamma_{a(0)(1450)0}m_{a(0)(1450)0}\sqrt{\frac{\left(m_{12}^2-(m_1-m_2)^2\right)\left(m_{12}^2-(m_1+m_2)^2\right)}{m_{12}^2}}\sqrt{m_{a(0)(1450)0}^2}}{\sqrt{\frac{\left(m_{a(0)(1450)0}^2-(m_1-m_2)^2\right)\left(m_{a(0)(1450)0}^2-(m_1+m_2)^2\right)}{m_{a(0)(1450)0}^2}}|m_{12}|}-m_{12}^2+m_{a(0)(1450)0}^2}$$



ampform.rtfd.io

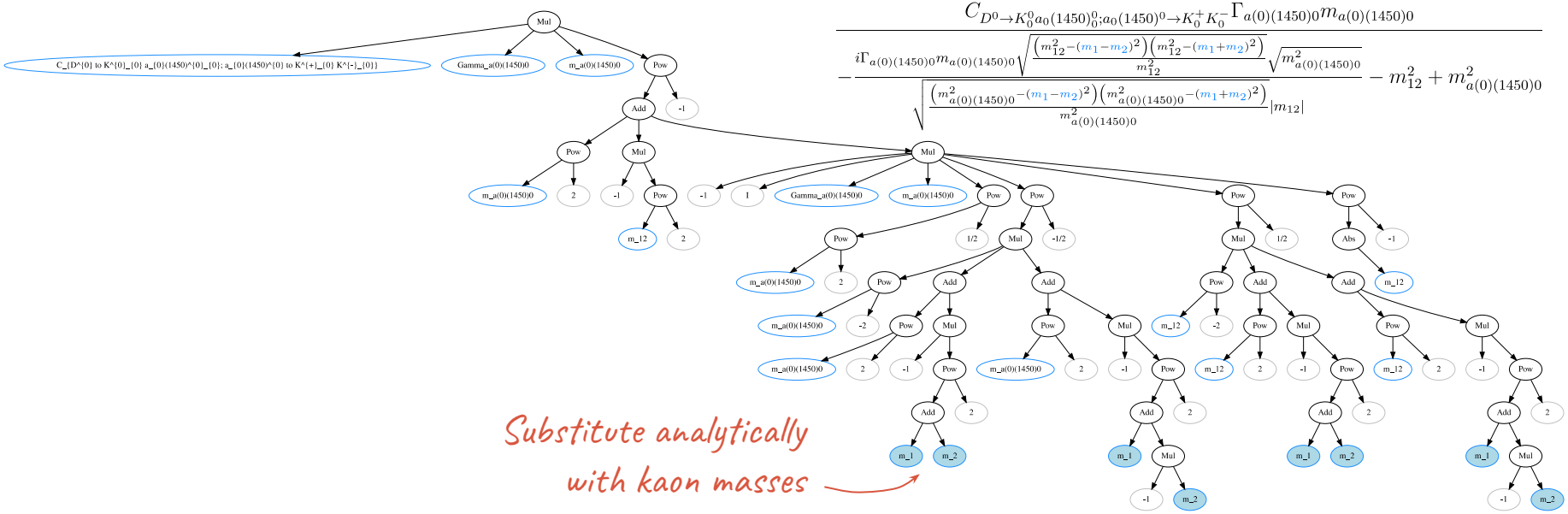
# AmpForm

## Symbolic amplitude model formulation




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

$$\frac{C_{D^0\to K_0^0 a_0(1450)_0^0;a_0(1450)^0\to K_0^+K_0^-}\Gamma_{a(0)(1450)0}m_{a(0)(1450)0}}{-\frac{i\Gamma_{a(0)(1450)0}m_{a(0)(1450)0}\sqrt{\frac{\left(m_{12}^2-(m_1-m_2)^2\right)\left(m_{12}^2-(m_1+m_2)^2\right)}{m_{12}^2}}\sqrt{m_{a(0)(1450)0}^2}}{\sqrt{\frac{\left(m_{a(0)(1450)0}^2-(m_1-m_2)^2\right)\left(m_{a(0)(1450)0}^2-(m_1+m_2)^2\right)}{m_{a(0)(1450)0}^2}}|m_{12}|}-m_{12}^2+m_{a(0)(1450)0}^2}$$

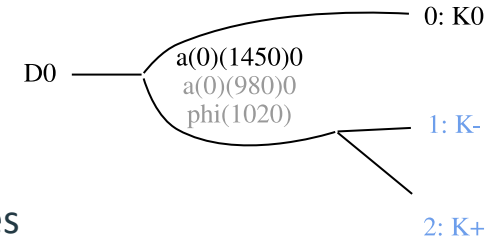



*Substitute analytically with kaon masses*

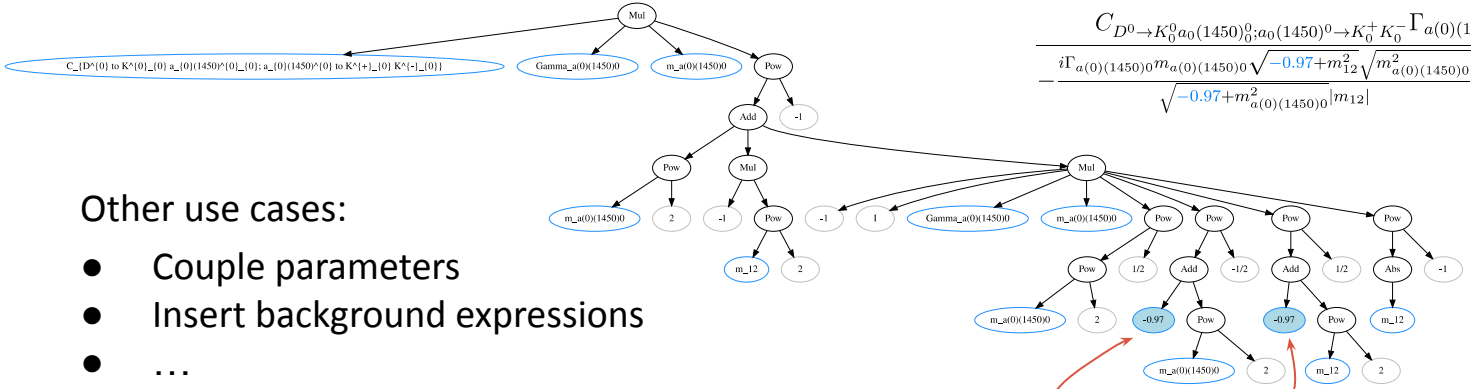
ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation




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




$$\frac{C_{D^0\to K_0^0 a_0(1450)_0^0; a_0(1450)^0\to K_0^+ K_0^-}\Gamma_{a(0)(1450)0} m_{a(0)(1450)0}}{-\frac{i\Gamma_{a(0)(1450)0} m_{a(0)(1450)0}\sqrt{-0.97+m_{12}^2}\sqrt{m_{a(0)(1450)0}^2}}{\sqrt{-0.97+m_{a(0)(1450)0}^2}|m_{12}|} - m_{12}^2 + m_{a(0)(1450)0}^2}$$

CAS collapses sub-trees

Other use cases:

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

⇒ **Transparency**, **flexibility** and **optimized computations**

ampform.rtfd.io

# AmpForm

## Symbolic amplitude model formulation

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

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

$$\theta\left(\boldsymbol{B_z}\left(\frac{|\vec{p_{12}}|}{E\left(p_{12}\right)}\right)\boldsymbol{R_y}\left(-\theta\left(p_{12}\right)\right)\boldsymbol{R_z}\left(-\phi\left(p_{12}\right)\right)p_1\right)$$

Easier to implement spin alignment etc.

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

launch binder Open in Colab

$$\operatorname{acos}\left(\frac{\boldsymbol{B_z}\left(\frac{\sqrt{\sum_{\text{axis1}}\left(p_{12}\right)[:,1:]^2}}{\left(p_{12}\right)[:,0]}\right)\boldsymbol{R_y}\left(-\operatorname{acos}\left(\frac{\left(p_{12}\right)[:,3]}{\sqrt{\sum_{\text{axis1}}\left(p_{12}\right)[:,1:]^2}}\right)\right)\boldsymbol{R_z}\left(-\operatorname{atan_2}\left(\left(p_{12}\right)[:,2],\left(p_{12}\right)[:,1]\right)\right)p_1\left[:,3\right]}{\sqrt{\sum_{\text{axis1}}\boldsymbol{B_z}\left(\frac{\sqrt{\sum_{\text{axis1}}\left(p_{12}\right)[:,1:]^2}}{\left(p_{12}\right)[:,0]}\right)\boldsymbol{R_y}\left(-\operatorname{acos}\left(\frac{\left(p_{12}\right)[:,3]}{\sqrt{\sum_{\text{axis1}}\left(p_{12}\right)[:,1:]^2}}\right)\right)\boldsymbol{R_z}\left(-\operatorname{atan_2}\left(\left(p_{12}\right)[:,2],\left(p_{12}\right)[:,1]\right)\right)p_1\left[:,1:\right]^2}}\right)$$

tensorwaves.rtfd.io

# 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.rtfd.io

# 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.rtfd.io

# 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
```

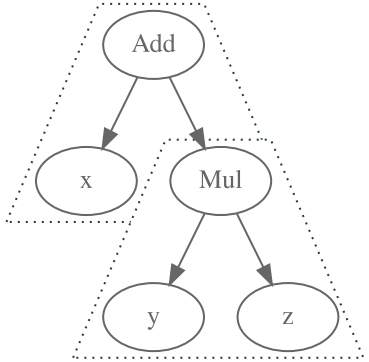
tensorwaves.rtfd.io

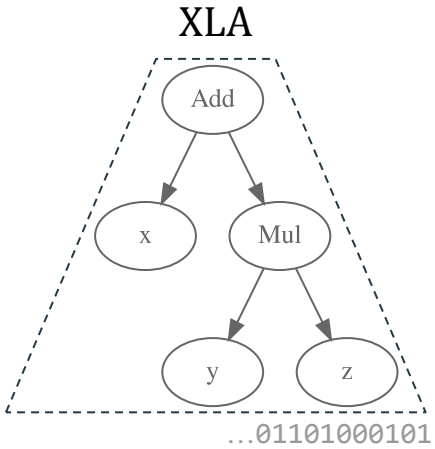
# TensorWaves

## 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
```

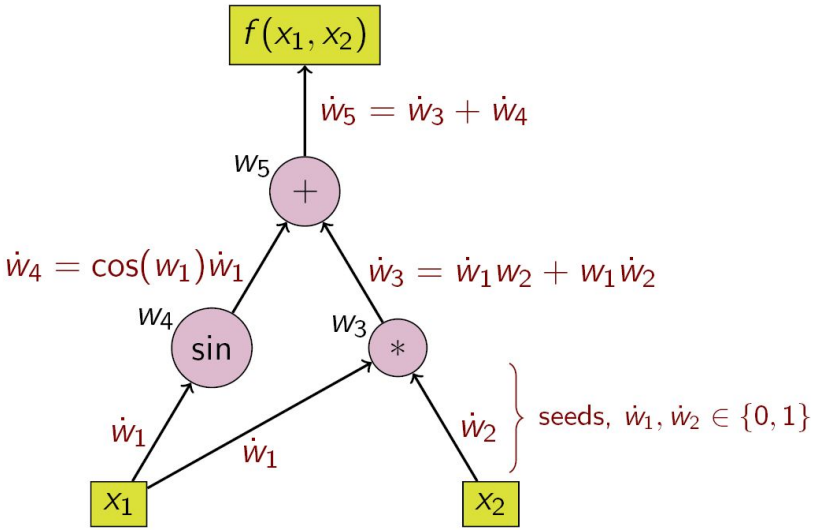
tensorwaves.rtfd.io

# 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.rtfd.io

# 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.rtfd.io

# 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.rtfd.io

# 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

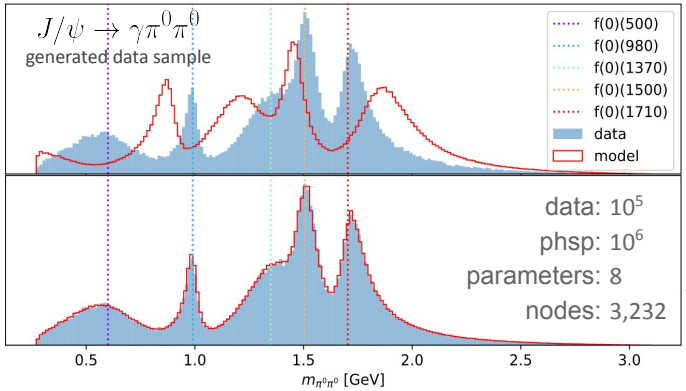
tensorwaves.rtfd.io

# TensorWaves

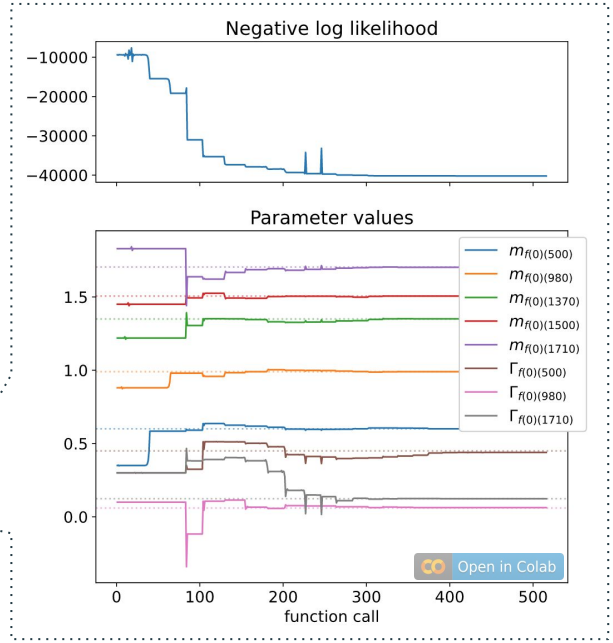
## Fit and generate data with multiple computational back-ends

**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** |




*optimise parameters*

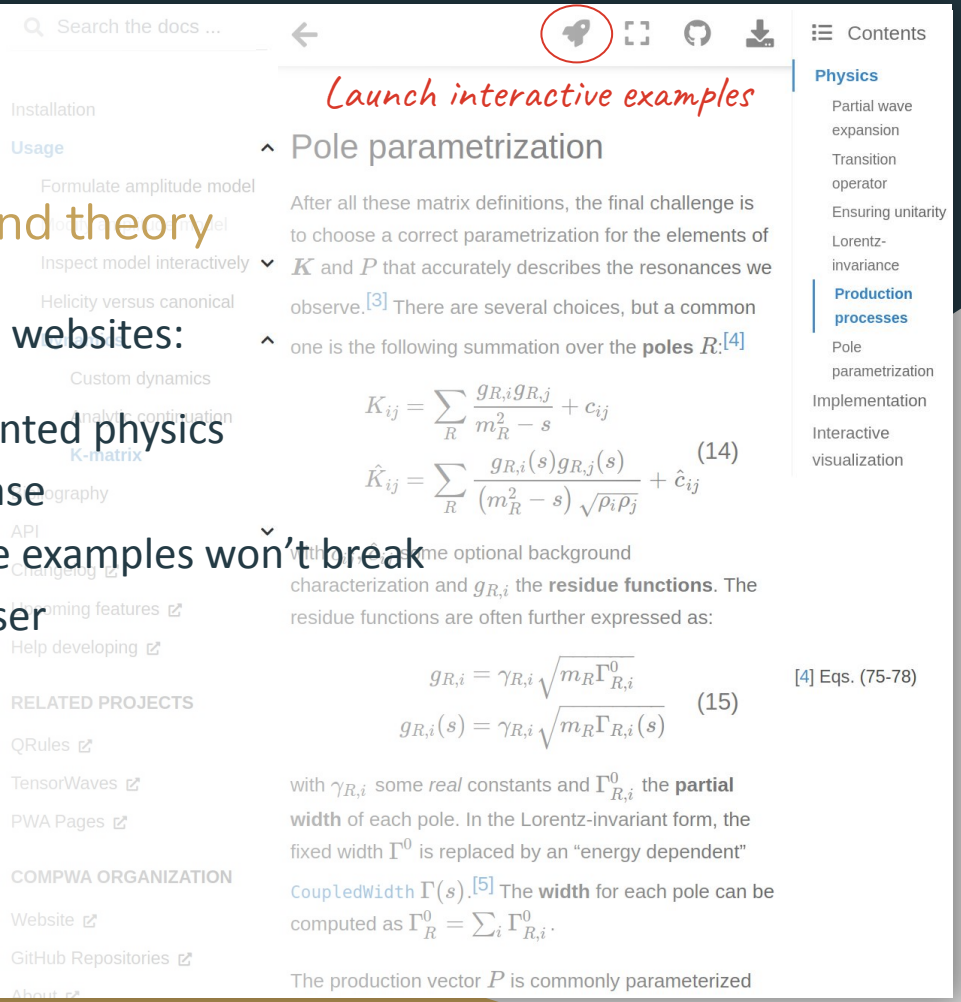


# Documentation

## 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)

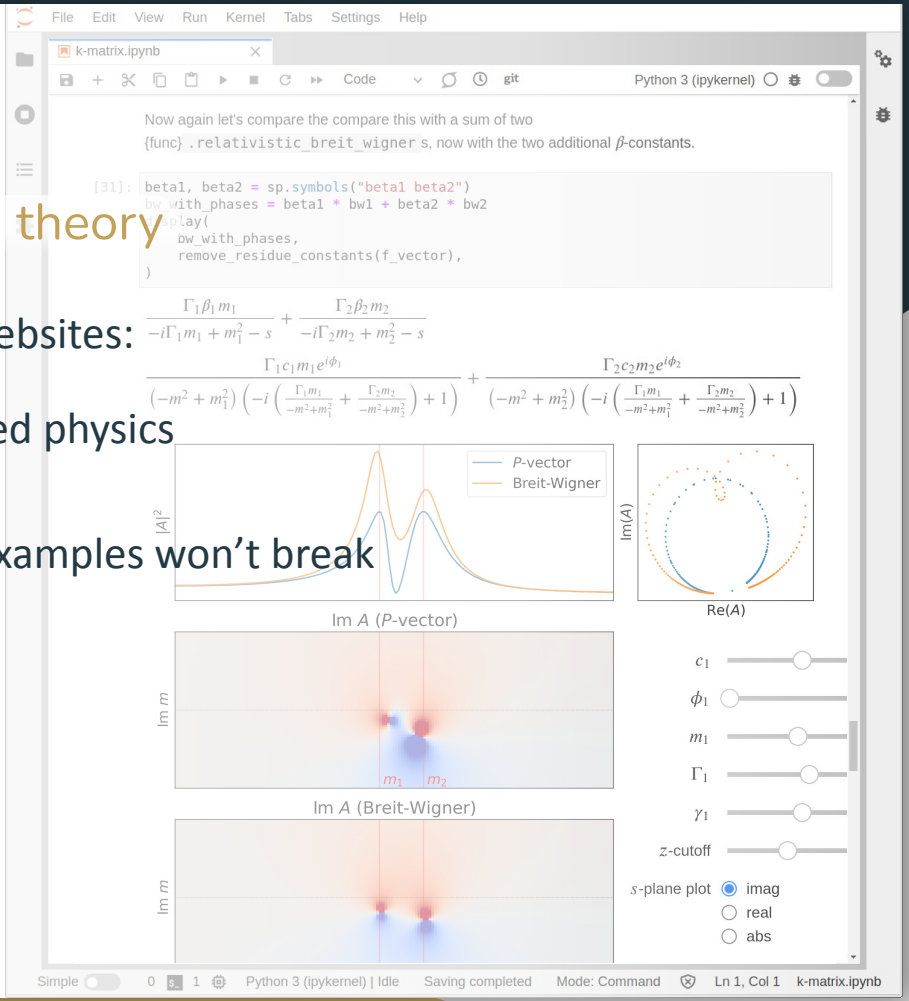


# Documentation

## 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)

- 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

Search the docs ...

Introduction

Formalisms

Tensor formalisms

Dynamics

Experimental data

Miscellaneous topics

SOFTWARE

PWA Frameworks

APPENDICES

Glossary

Bibliography

RELATED PROJECTS

QRules

TensorWaves

COMPWA ORGANIZATION

Website

**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.

| | | Since | Lastest commit | C++ | Python | Cuda |
|---|---|---|---|---|---|---|
| AmpGen | CLEO / LHCb | 2018 | 11/2021 | ✓ | | |
| AmpTools | | 2011 | 11/2021 | ✓ | | ✓ |
| ComPWA (C++) | | 2012 | 06/2020 | ✓ | | |
| ComPWA project<br>• QRules<br>• AmpForm<br>• TensorWaves | | 2020 | 12/2021 | | ✓ | |
| cFit | | 2012 | 05/2018 | ✓ | | |
| DPWA | BESIII | 2011 | | ✓ | | |
| 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<br>• AmpliTF<br>• TFA2 | LHCb | 2016 | 11/2021 | | ✓ | |
| 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.rtfd.io/en/stable/develop.html

compwa-org.rtfd.io

# 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

PyPI Conda

```
python3 -m pip install tensorwaves
```

This installs the latest, stable release that you can find on the `stable` branch.

GPU support

Tip

To start developing, simply run the following from a cloned repository on your machine:

Conda Python venv

```
conda env create
conda activate ampform
pre-commit install
```

See Virtual environment for more info.

compwa-org.rtfd.io

# 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

```
$ 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
```

compwa-org.rtfd.io

# 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

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
```

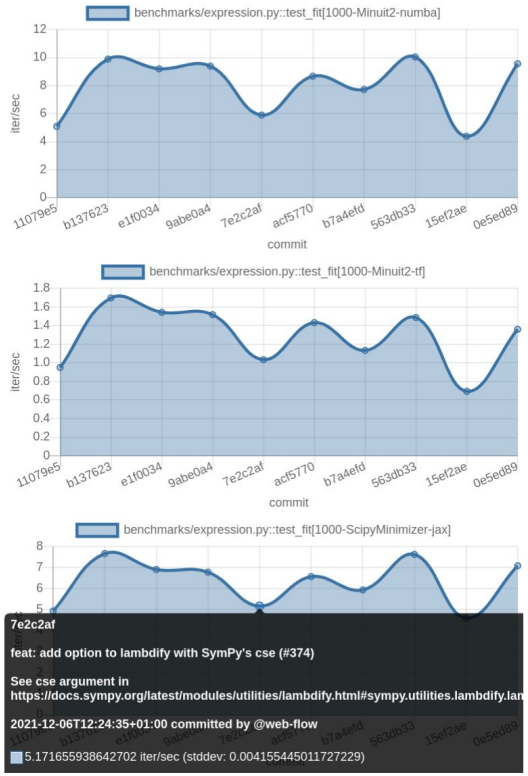
compwa-org.rtfd.io

# 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

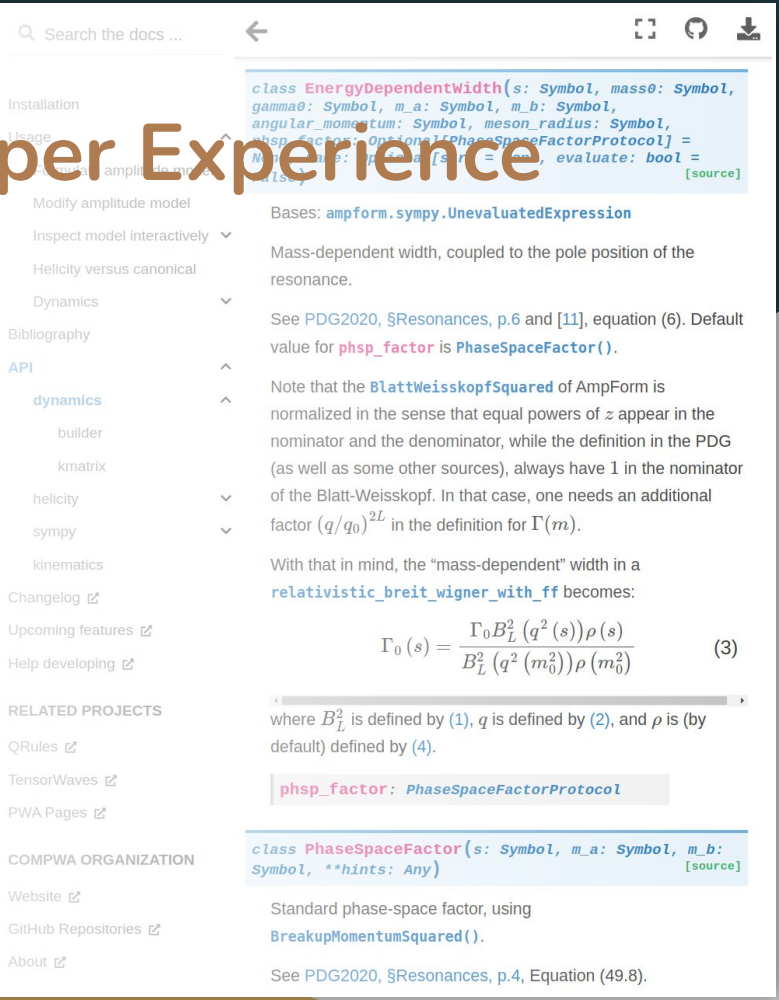


# Code quality and Developer Experience

**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](compwa-org.rtfd.io/en/stable/develop.html)

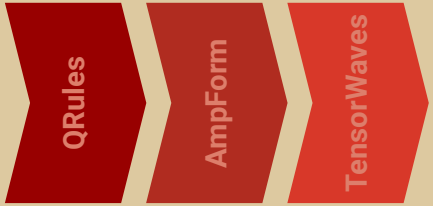


compwa-org.rtfd.io

# Summary — Key ComPWA features

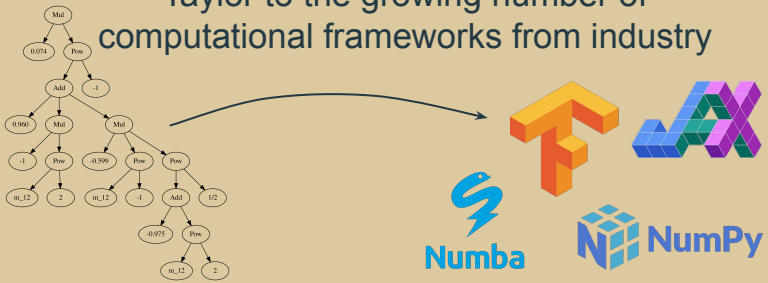
## Modularity

Separate libraries that facilitate common procedures in amplitude analysis



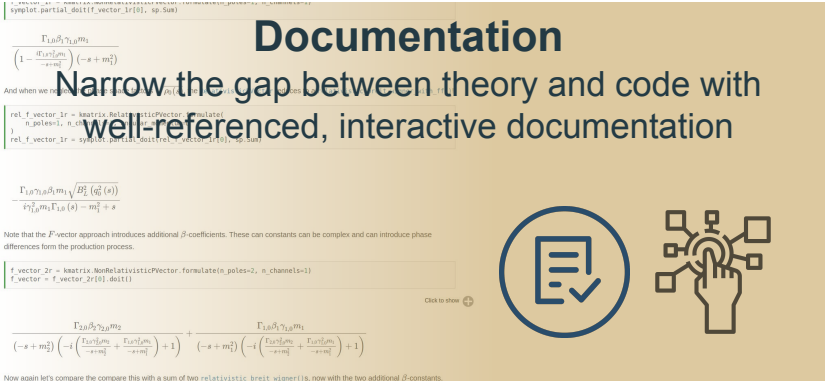

## Scalability

Taylor to the growing number of computational frameworks from industry



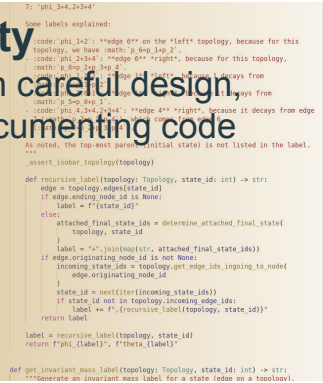

## Documentation

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



## Code quality

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

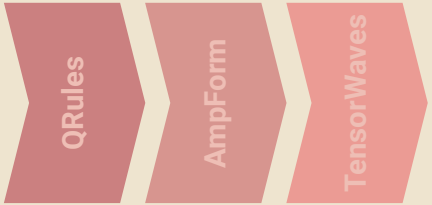


compwa-org.rtfd.io

# Summary — Key ComPWA features

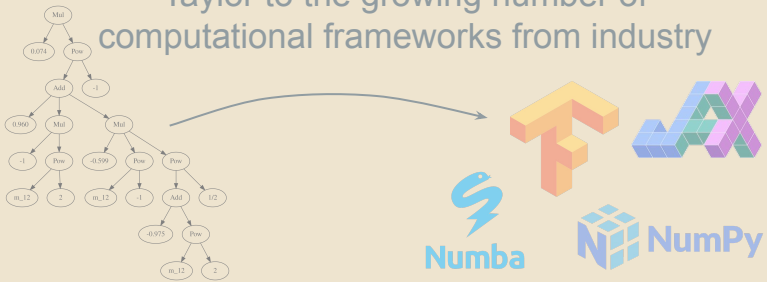
## Modularity

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

## Code quality

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

*Thank you for your attention!*