The ComPWA Project

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

Remco de Boer
Ruhr University Bochum

8 September 2021
PWA 12 / ATHOS 7
held at the University of Bristol

What is the ComPWA project?

- “Common Partial Wave Analysis”
- Open-source GitHub organization
- Originally a C++ framework (ComPWA)
- Now maintains a collection of Python tools for amplitude analysis
- Developer group at Ruhr University Bochum, JGU Mainz, and GSI
- So far developed in the context of BESIII and PANDA analyses

github.com/ComPWA

What does ComPWA aim for?

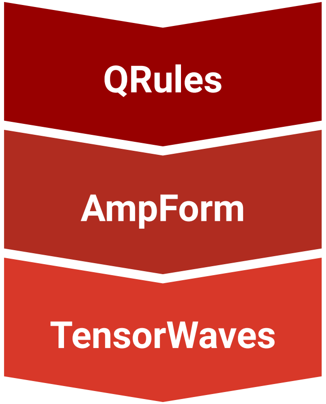
- **Academic continuity**:
  long-term, collaboration-independent
  PWA software development
- Provide an easy starting point
  for researchers new to the field of PWA
- Build up modern, interlinked, and
  interactive PWA knowledge-bases
- Maintain libraries that facilitate and automate
  common procedures in amplitude analysis

⇒ Narrow the gap between theory and code
⇒ Bring usage and development closer together

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 model templates

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.rtfd.io

QRules

Automated quantum number conservation rules

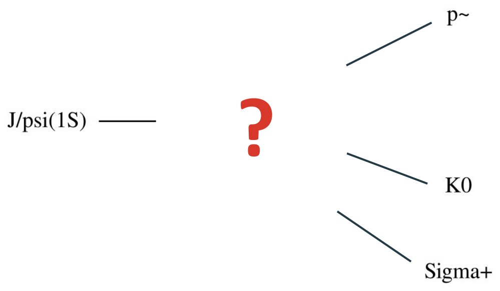
Aim: compute which particle reactions are allowed between a given initial and final state qrules.rtfd.io

QRules

Automated quantum number conservation rules

Aim: 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.)



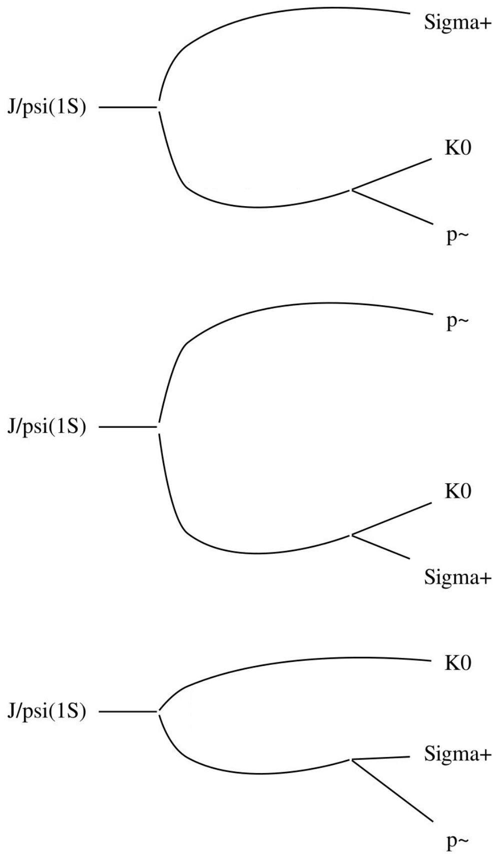
qrules.rtfd.io

QRules

Automated quantum number conservation rules

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



qrules.rtfd.io

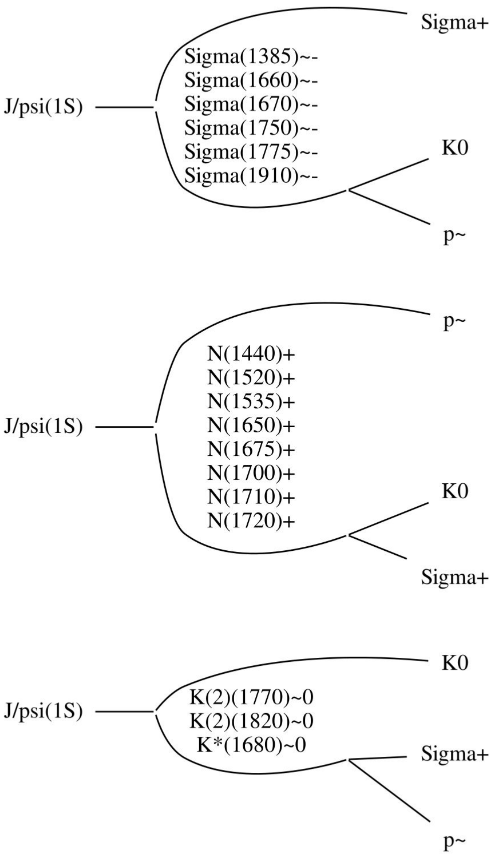
QRules

Automated quantum number conservation rules

Aim: 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:
   - gets corresponding particle properties from the PDG
     (or any custom definitions),
   - determines all possible decay topologies,
   - propagates quantum numbers through intermediate edges,
   - and selects all allowed transitions with its conservation laws

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



qrules.rtfd.io

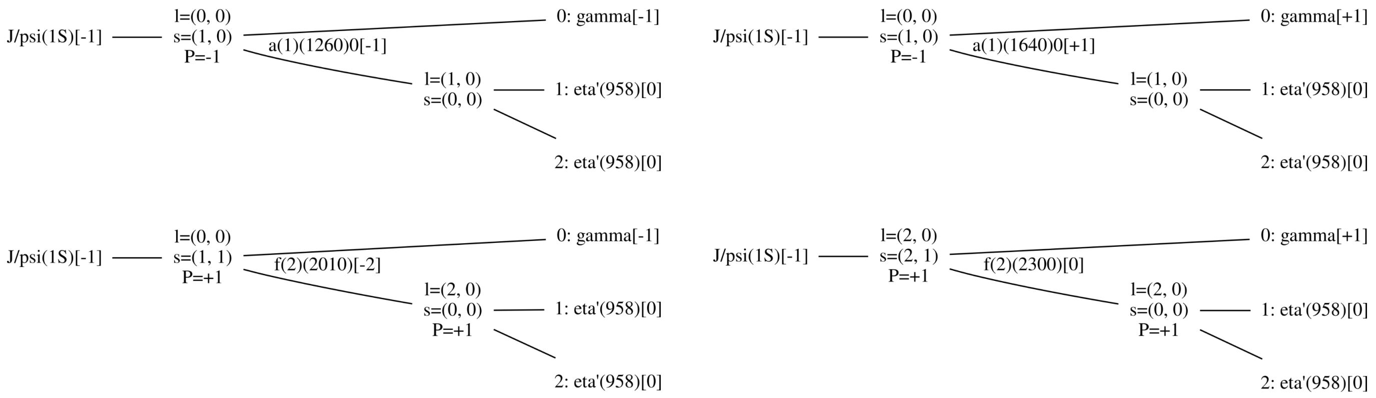
QRules

Automated quantum number conservation rules

The returned objects contain **all information to build an amplitude model**!

```python
reaction = qrules.generate_transitions(
    initial_state="J/psi(1S)",
    final_state=["gamma", "eta'(958)", "eta'(958)"],
)
```

Propagating quantum numbers: 100% 36/36 [00:04<00:00, 7.55it/s]



qrules.rtfd.io

QRules

Automated quantum number conservation rules

The library also provides several related features:

Check which conservation rules are violated:

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

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

Find particles by selecting quantum numbers:

```
selection = PDG.filter(lambda p: p.spin > 0 and p.charmness and p.mass > 2.82)
selection.names
```

```
['Lambda(c)(2880)~-', 'Lambda(c)(2880)+', 'Xi(c)(2815)0', 'Xi(c)(2815)~0']
```

Get particle properties:*

```
PDG = qrules.load_pdg()
PDG.find("f(0)(980)")
```

```
Particle(
  name='f(0)(980)',
  pid=9010221,
  latex='f_{0}(980)',
  spin=0.0,
  mass=0.99,
  width=0.06,
  isospin=Spin(0, 0),
  parity=+1,
  c_parity=+1,
  g_parity=+1,
)
```

* PDG info computed from the scikit-hep particle package

ampform.rtfd.io

AmpForm

Symbolic amplitude model formulation

- Implements **spin formalisms and dynamics**
- Can express QRules' state transitions as an amplitude model
- Amplitude models are formulated as algebraic expressions (SymPy CAS)
- User can further modify the expressions
- The models serve as a **mathematical template for fitter packages**

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

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

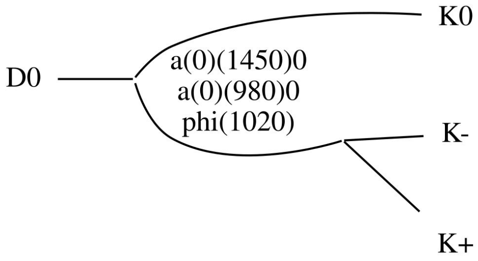
```python
matrix = NonRelativisticKMatrix.formulate(
    n_poles=1,
    n_channels=2,
).doit()
matrix[0, 0].simplify()
```

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

ampform.rtfd.io

AmpForm

Symbolic amplitude model formulation



Example

Building an amplitude model for $D \to K^0 K^- K^+$ with three resonances

```python
builder = ampform.get_builder(reaction)
for p in reaction.get_intermediate_particles():
    builder.set_dynamics(p.name, create_relativistic_breit_wigner_with_ff)
model = builder.formulate()
```

$$\left|A_{D^{0}_{0} \to K^{0}_{0} \phi(1020)_{0}; \phi(1020)_{0} \to K^{+}_{0} K^{-}_{0}} + A_{D^{0}_{0} \to K^{0}_{0} a_{0}(1450)^{0}_{0}; a_{0}(1450)^{0}_{0} \to K^{+}_{0} K^{-}_{0}} + A_{D^{0}_{0} \to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}_{0} \to K^{+}_{0} K^{-}_{0}}\right|^{2}$$

LaTeX generated by the code!

- User selects Breit-Wigner to parametrize each resonance
- AmpForm takes care of spin (helicity formalism)
- Resulting amplitude model expressed symbolically

ampform.rtfd.io

AmpForm

Symbolic amplitude model formulation

Expression for the amplitude model can be further inspected:

```python
some_amplitude = model.components[
    R"A_{D^{0}_{0} \to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}_{0} \to K^{+}_{0} K^{-}_{0}}"
]
```

$$\frac{C_{D^{0}\to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}\to K^{+}_{0} K^{-}_{0}} \Gamma_{a_{0}(980)^{0}} m_{a_{0}(980)^{0}} \sqrt{B_{0}^{2}\left(\left(d_{a_{0}(980)^{0}}\right)^{2} q_{122}^{2}\left(m_{12}^{2}\right)\right)} D^{0}_{0,0}\left(-\phi_{1+2}, \theta_{1+2}, 0\right) D^{0}_{0,0}\left(-\phi_{1,1+2}, \theta_{1,1+2}, 0\right)}{-m_{12}^{2} + \left(m_{a_{0}(980)^{0}}\right)^{2} - i m_{a_{0}(980)^{0}} \Gamma\left(m_{12}^{2}\right)}$$

Spin formalism

Dynamics

$$= \frac{C_{D^{0}\to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}\to K^{+}_{0} K^{-}_{0}} \Gamma_{a_{0}(980)^{0}} m_{a_{0}(980)^{0}}}{-\frac{i\Gamma_{a_{0}(980)^{0}}\left(m_{a_{0}(980)^{0}}\right)^{2}\sqrt{\frac{\left(m_{12}^{2}-(m_{1}-m_{2})^{2}\right)\left(m_{12}^{2}-(m_{1}+m_{2})^{2}\right)}{m_{12}^{2}}}}{m_{12}\sqrt{\frac{\left(\left(m_{a_{0}(980)^{0}}\right)^{2}-(m_{1}-m_{2})^{2}\right)\left(\left(m_{a_{0}(980)^{0}}\right)^{2}-(m_{1}+m_{2})^{2}\right)}{\left(m_{a_{0}(980)^{0}}\right)^{2}}}} - m_{12}^{2} + \left(m_{a_{0}(980)^{0}}\right)^{2}}$$

$$= \frac{0.074}{-m_{12}^{2} + 0.96 - \frac{0.599\sqrt{m_{12}^{2}-0.975}}{m_{12}}}$$

ampform.rtfd.io

AmpForm

Symbolic amplitude model formulation

Expression for the amplitude model can be further inspected:

```
some_amplitude = model.components[
    R"A_{D^{0}_{0} \to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}_{0} \to K^{+}_{0} K^{-}_{0}}"
]
```

$$\frac{C_{D^{0} \to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0} \to K^{+}_{0} K^{-}_{0}} \Gamma_{a_{0}(980)^{0}} m_{a_{0}(980)^{0}} \sqrt{B_{0}^{2}\left(\left(d_{a_{0}(980)^{0}}\right)^{2} q^{2}_{122}\left(m_{12}^{2}\right)\right)} D^{0}_{0,0}\left(-\phi_{1+2}, \theta_{1+2}, 0\right) D^{0}_{0,0}\left(-\phi_{1,1+2}, \theta_{1,1+2}, 0\right)}{-m_{12}^{2} + \left(m_{a_{0}(980)^{0}}\right)^{2} - i m_{a_{0}(980)^{0}} \Gamma\left(m_{12}^{2}\right)}$$

$$= \frac{C_{D^{0} \to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0} \to K^{+}_{0} K^{-}_{0}} \Gamma_{a_{0}(980)^{0}} m_{a_{0}(980)^{0}}}{- \frac{i \Gamma_{a_{0}(980)^{0}} \left(m_{a_{0}(980)^{0}}\right)^{2} \sqrt{\frac{\left(m_{12}^{2} - (m_{1} - m_{2})^{2}\right)\left(m_{12}^{2} - (m_{1} + m_{2})^{2}\right)}{m_{12}^{2}}}}{m_{12} \sqrt{\frac{\left(\left(m_{a_{0}(980)^{0}}\right)^{2} - (m_{1} - m_{2})^{2}\right)\left(\left(m_{a_{0}(980)^{0}}\right)^{2} - (m_{1} + m_{2})^{2}\right)}{\left(m_{a_{0}(980)^{0}}\right)^{2}}}} - m_{12}^{2} + \left(m_{a_{0}(980)^{0}}\right)^{2}}$$

CAS performs algebraic simplifications

$$= \frac{0.074}{-m_{12}^{2} + 0.96 - \frac{0.599 \sqrt{m_{12}^{2} - 0.975}}{m_{12}}}$$

AmpForm

Symbolic amplitude model formulation

Expression for the amplitude model can be further inspected:

```
some_amplitude = model.components[
    R"A_{D^{0}_{0} \to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}_{0} \to K^{+}_{0} K^{-}_{0}}"
]
```

$$\frac{C_{D^{0}\to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}\to K^{+}_{0} K^{-}_{0}} \Gamma_{a_{0}(980)^{0}} m_{a_{0}(980)^{0}} \sqrt{B_{0}^{2}\left(\left(d_{a_{0}(980)^{0}}\right)^{2} q_{122}^{2}\left(m_{12}^{2}\right)\right)} D^{0}_{0,0}\left(-\phi_{1+2}, \theta_{1+2}, 0\right) D^{0}_{0,0}\left(-\phi_{1,1+2}, \theta_{1,1+2}, 0\right)}{-m_{12}^{2} + \left(m_{a_{0}(980)^{0}}\right)^{2} - i m_{a_{0}(980)^{0}} \Gamma\left(m_{12}^{2}\right)}$$

$$= \frac{C_{D^{0}\to K^{0}_{0} a_{0}(980)^{0}_{0}; a_{0}(980)^{0}\to K^{+}_{0} K^{-}_{0}} \Gamma_{a_{0}(980)^{0}} m_{a_{0}(980)^{0}}}{-\frac{i \Gamma_{a_{0}(980)^{0}} \left(m_{a_{0}(980)^{0}}\right)^{2} \sqrt{\frac{\left(m_{12}^{2} - (m_{1} - m_{2})^{2}\right)\left(m_{12}^{2} - (m_{1} + m_{2})^{2}\right)}{m_{12}^{2}}}}{m_{12} \sqrt{\frac{\left(\left(m_{a_{0}(980)^{0}}\right)^{2} - (m_{1} - m_{2})^{2}\right)\left(\left(m_{a_{0}(980)^{0}}\right)^{2} - (m_{1} + m_{2})^{2}\right)}{\left(m_{a_{0}(980)^{0}}\right)^{2}}}} - m_{12}^{2} + \left(m_{a_{0}(980)^{0}}\right)^{2}}$$

$$= \frac{0.074}{-m_{12}^{2} + 0.96 - \frac{0.599 \sqrt{m_{12}^{2} - 0.975}}{m_{12}}}$$

← *AmpForm provides suggested parameter values*

ampform.rtfd.io

AmpForm

Symbolic amplitude model formulation

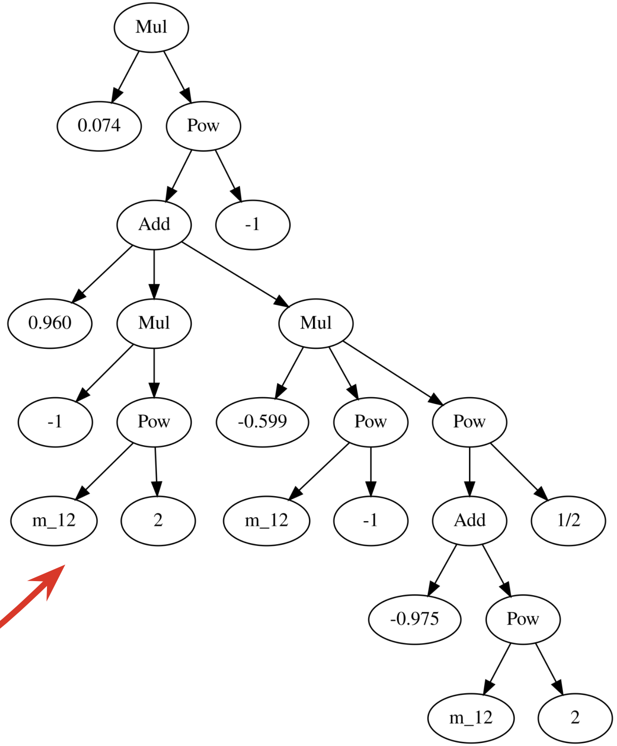
Original motivation: these expressions are actually **trees that represent fundamental mathematical operations**!

⇒ Can serve as template for faster computational software

$$\frac{\cdots}{-m_{12}^2 + \left(m_{a_0(980)^0}\right)^2 - i m_{a_0(980)^0}\Gamma\left(m_{12}^2\right)}$$

$$= \frac{C_{D^0\to K^0_0 a_0(980)^0_0; a_0(980)^0\to K^+_0 K^-_0}\Gamma_{a_0(980)^0} m_{a_0(980)^0}}{-\frac{i\Gamma_{a_0(980)^0}\left(m_{a_0(980)^0}\right)^2\sqrt{\frac{\left(m_{12}^2-(m_1-m_2)^2\right)\left(m_{12}^2-(m_1+m_2)^2\right)}{m_{12}^2}}}{m_{12}\sqrt{\frac{\left(\left(m_{a_0(980)^0}\right)^2-(m_1-m_2)^2\right)\left(\left(m_{a_0(980)^0}\right)^2-(m_1+m_2)^2\right)}{\left(m_{a_0(980)^0}\right)^2}}} - m_{12}^2 + \left(m_{a_0(980)^0}\right)^2}$$

$$= \frac{0.074}{-m_{12}^2 + 0.96 - \frac{0.599\sqrt{m_{12}^2-0.975}}{m_{12}}}$$



tensorwaves.rtfd.io

TensorWaves

Fit and generate data with multiple computational back-ends

- General fitter package
- Express amplitude templates 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, ...)

```
intensity = LambdifiedFunction(template, backend="jax")  # numpy, tensorflow
estimator = UnbinnedNLL(intensity, data_sample, phsp_sample)
optimizer = Minuit2()  # Scipy
fit_result = optimizer.optimize(estimator, initial_parameters)
```

187/? [00:33<00:00, 13.15it/s, estimator=-7.59e+4]

TensorWaves

Fit and generate data with multiple computational back-ends

Why is this nice?

- Heavy computations outsourced to specialized packages from the machine learning and data science community
- Get support for GPUs, multithreading, etc. for free
- Very small code-base — easy to maintain
- More time for physics!

```
intensity = LambdifiedFunction(template, backend="jax")  # numpy, tensorflow
estimator = UnbinnedNLL(intensity, data_sample, phsp_sample)
optimizer = Minuit2()  # Scipy
fit_result = optimizer.optimize(estimator, initial_parameters)
```

187/? [00:33<00:00, 13.15it/s, estimator=-7.59e+4] tensorwaves.rtfd.io

TensorWaves

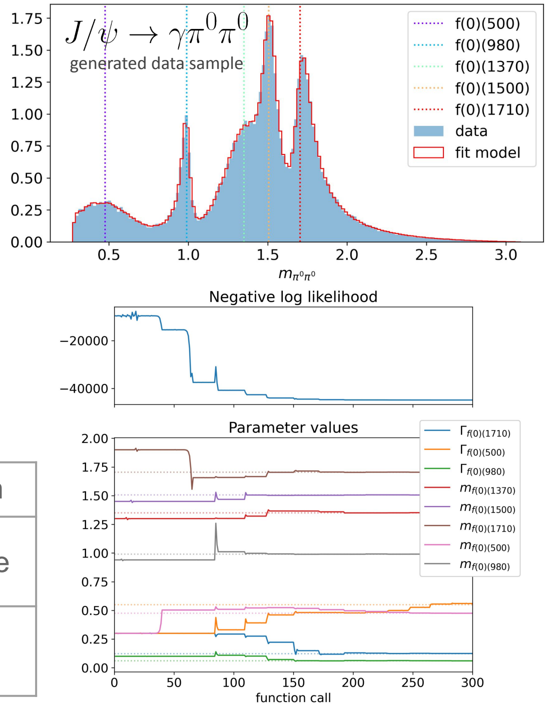
Fit and generate data with multiple computational back-ends

Why is this nice?

- Heavy computations outsourced to specialized packages from the machine learning and data science community
- Get support for GPUs, multithreading, etc. for free
- Very small code-base — easy to maintain
- More time for physics!

Some JAX+Minuit2 performance numbers on a single machine:

	data	phsp	params.	expr. complexity	duration
mini-demo on the right	10^5	10^6	8	2,187 nodes	~1 minute
benchmark fit $J/\psi \to K^0 \Sigma^+ \bar{p}$	1.2×10^5	2.3×10^5	84	176,023 nodes	3 hours



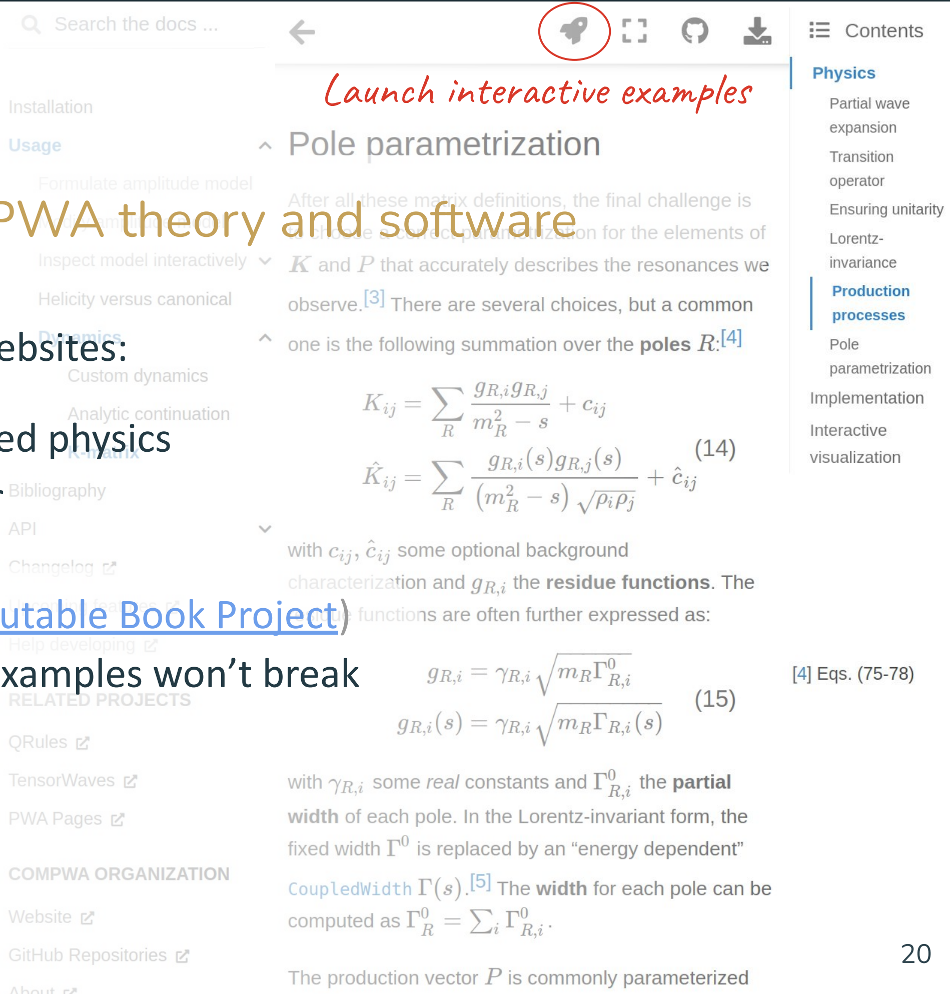
PWA Software Pages

Interactive knowledge-base for PWA theory and software

All packages come with well-maintained websites:

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

Narrow the gap between code and theory!



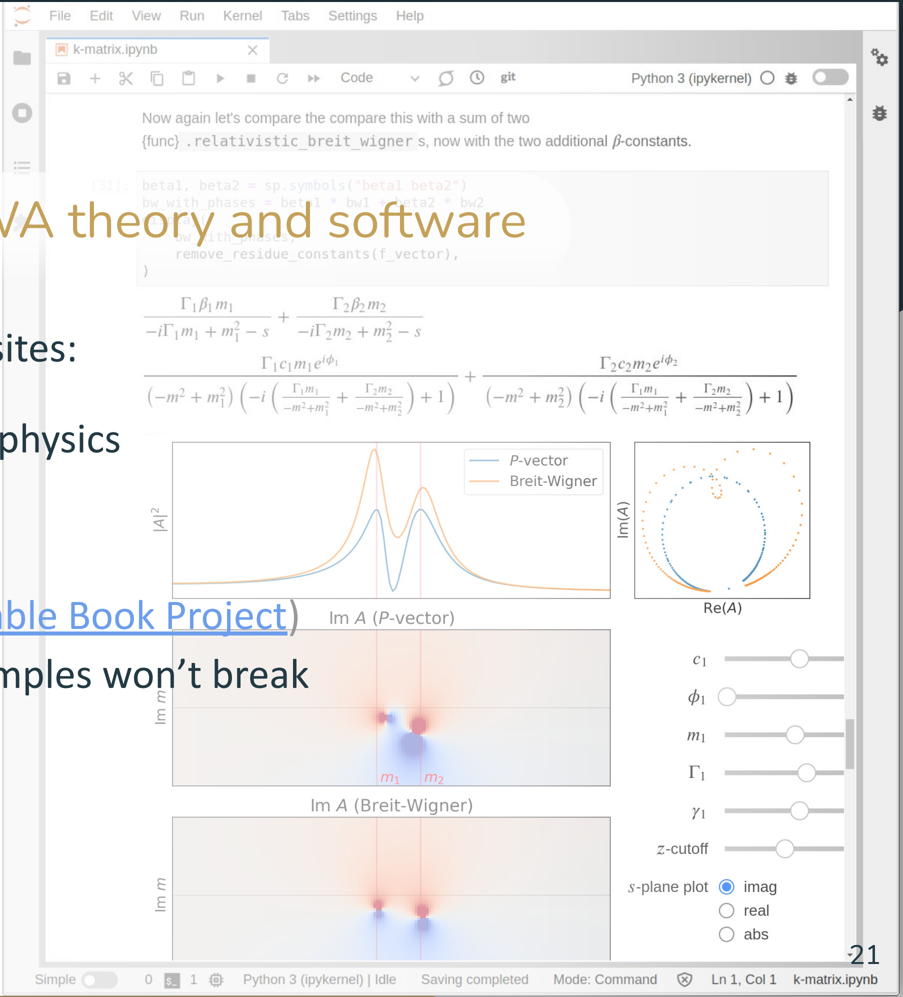
PWA Software Pages

Interactive knowledge-base for PWA theory and software

All packages come with well-maintained websites:

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

Narrow the gap between code and theory!



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
- Readers can easily navigate to literature or existing PWA software
- Currently skeletal, but infrastructure is there and easy contribute to
- No need to know HTML, CSS, etc.

Happy to include or reference your PWA project!

pwa.rtfd.io

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
- Readers can easily navigate to literature or existing PWA software
- Currently skeletal, but infrastructure is there and easy contribute to
- No need to know HTML, CSS, etc.

Happy to include or reference your PWA project!

Thank you for your attention!