RUHR-UNIVERSITÄT BOCHUM
Institut für Experimentalphysik I

The ComPWA project

Speeding up amplitude analysis with a Computer Algebra System

Remco de Boer[1], Miriam Fritsch[1], Wolfgang Gradl[2], Stefan Pflüger[1], and Leonard Wollenberg[1]
[1]Ruhr-Universität Bochum — [2]Johannes Gutenberg Universität Mainz

remco.deboer@rub.de

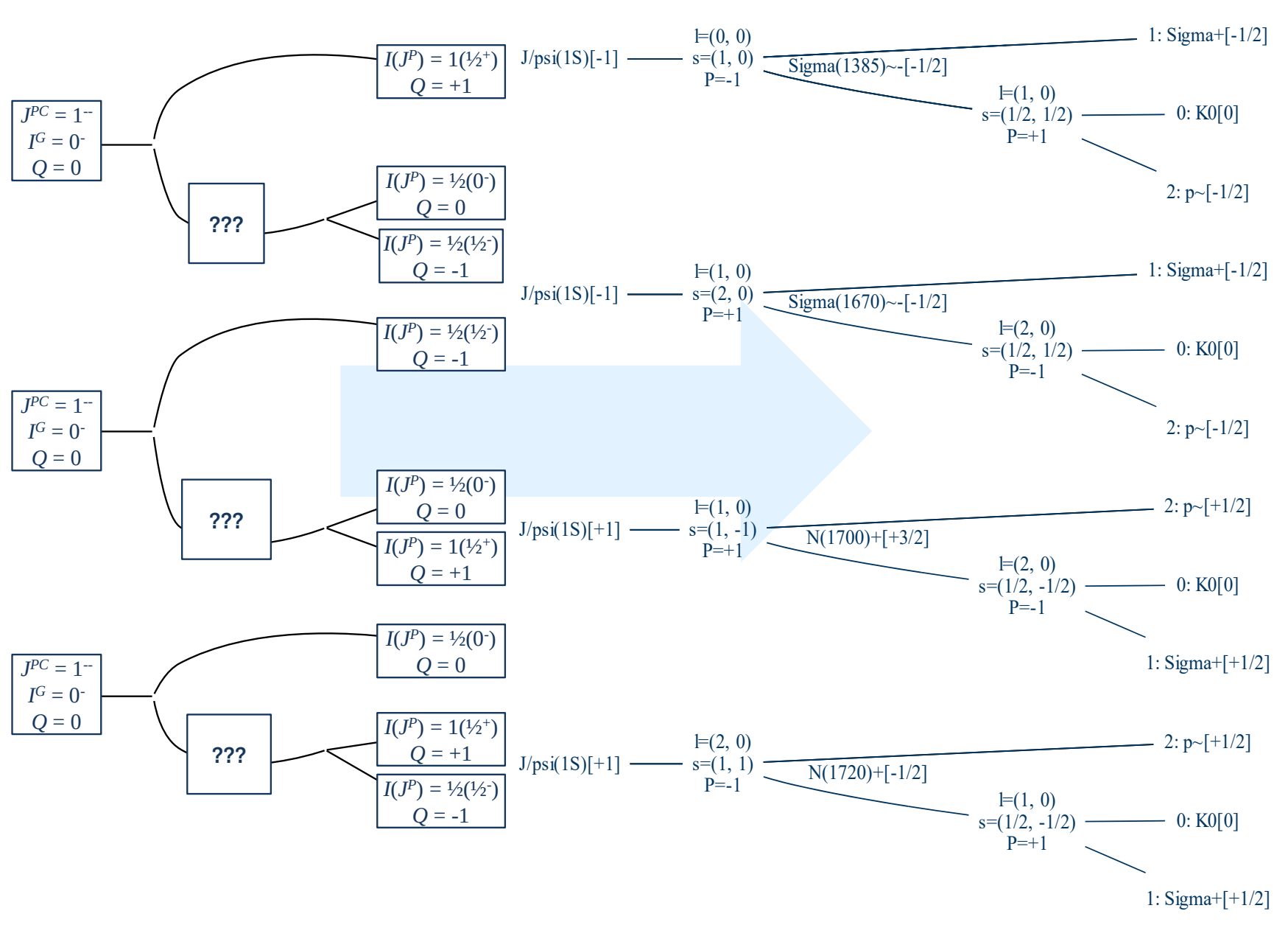
Three Python Libraries for a full amplitude analysis

`QRules`: Automated quantum number conservation rules

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

Input: boundary conditions

- initial and final state particles
- optional intermediate state restrictions



Output: state transitions

- Determines possible decay topologies
- Gets corresponding particle properties from the PDG (or custom definitions)
- Propagates quantum numbers through intermediate states
- Selects all allowed transitions with its conservation laws

qrules.rtfd.io

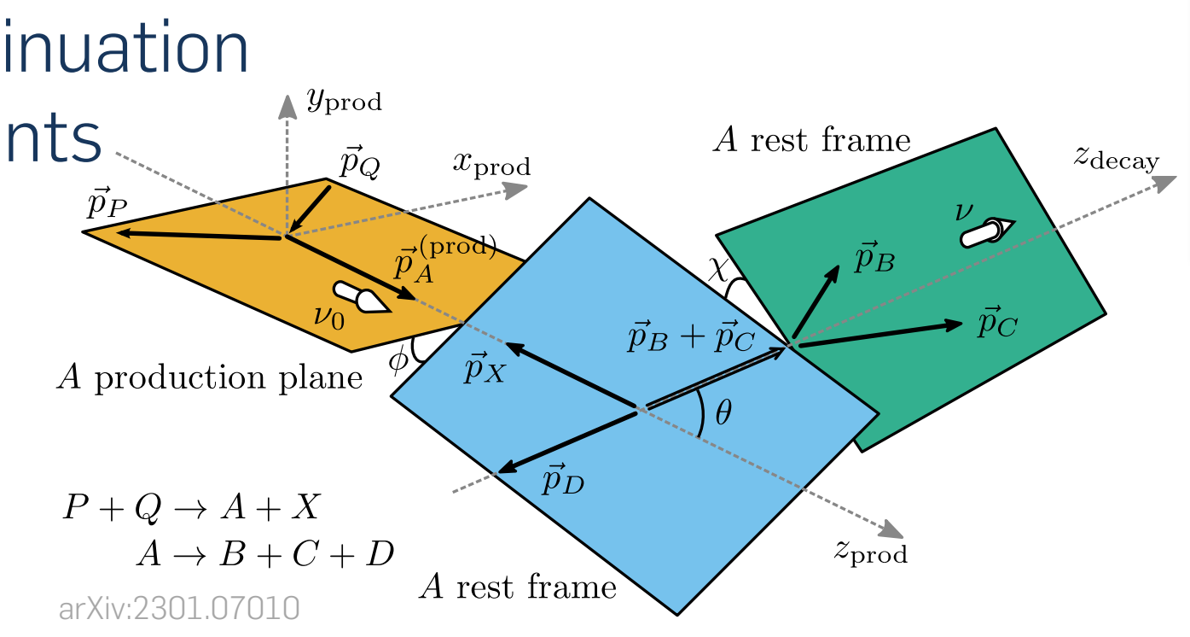
`AmpForm`: Symbolic amplitude models

Core responsibilities:

- Library of spin formalisms and dynamics
- Formulate amplitude model for QRules output

Implemented formalisms:

- Helicity formalism for multibody decays
- Transformation to canonical basis
- Dynamics parametrisations, such as analytic continuation
- Spin alignments



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

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

```python
matrix = NonRelativisticKMatrix.formulate(
    n_channels=2,
    n_poles=1,
).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

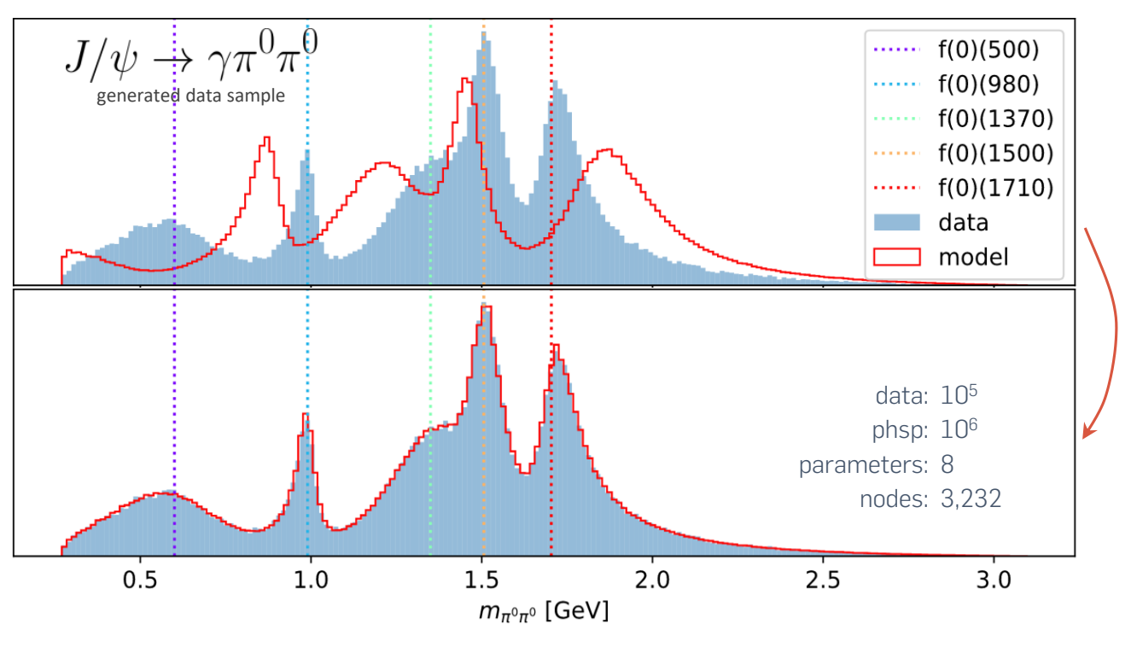
`TensorWaves`: Fit data with multiple computational back-ends

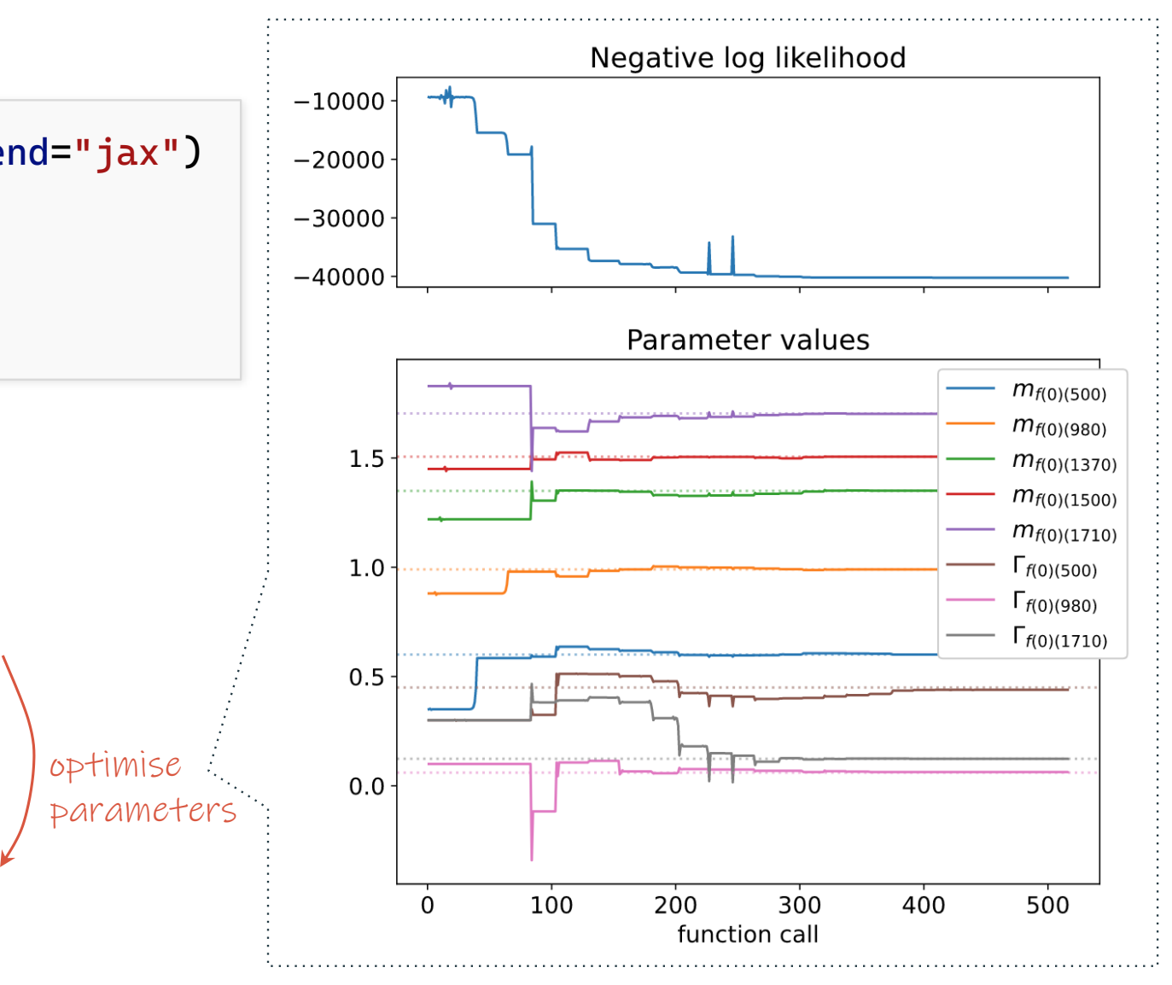
- Mathematical expressions → Computational back-end
- Generation of amplitude-based Monte Carlo samples
- Fits with TensorFlow, NumPy, JAX, ...
- Different optimizers: Minuit2, SciPy, ...

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

NumPy

Numba

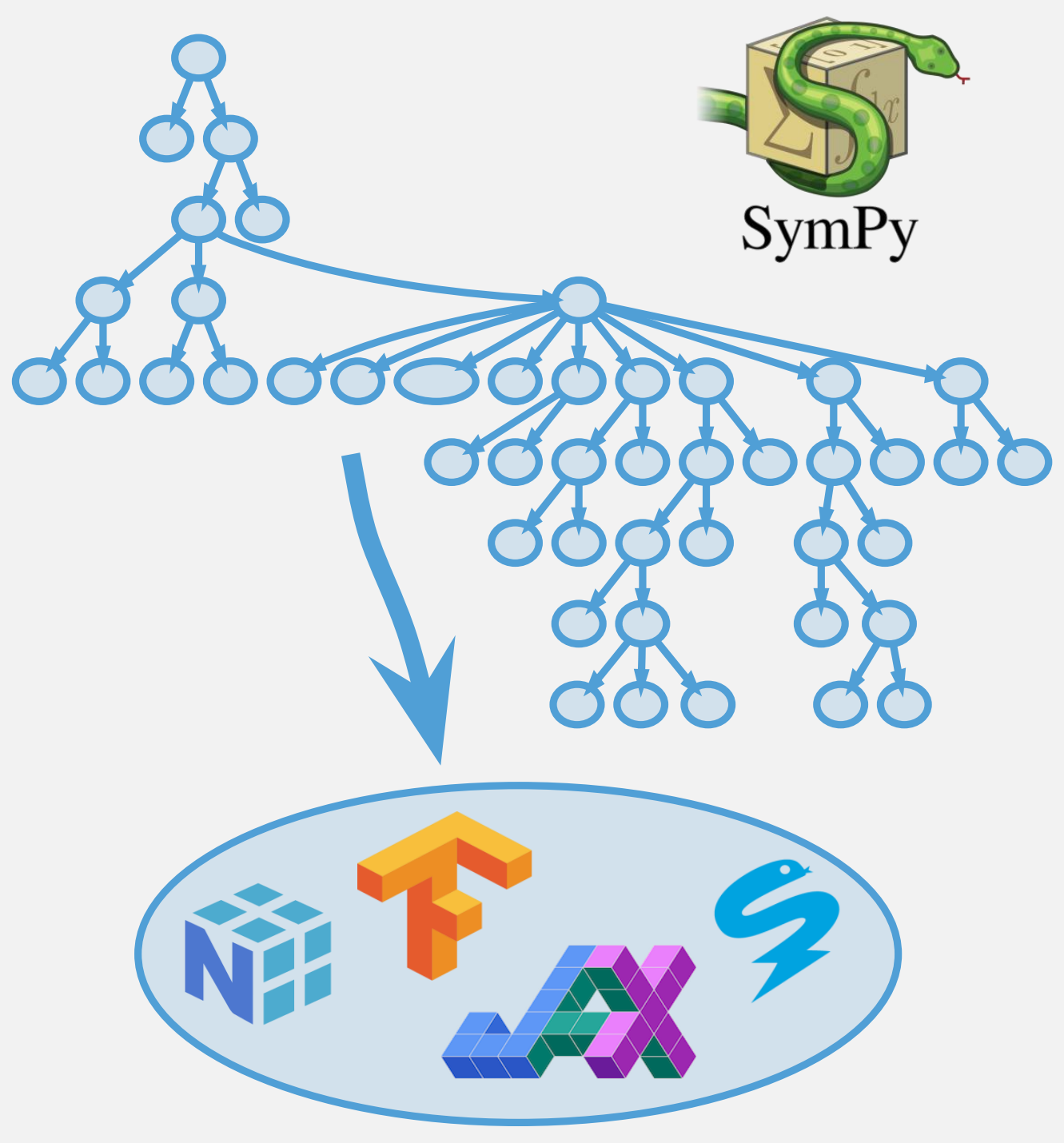




tensorwaves.rtfd.io

Core idea:

1. Formulate amplitude model as formulas
   → Computer Algebra System
2. Usage of **fast numerical back-ends**

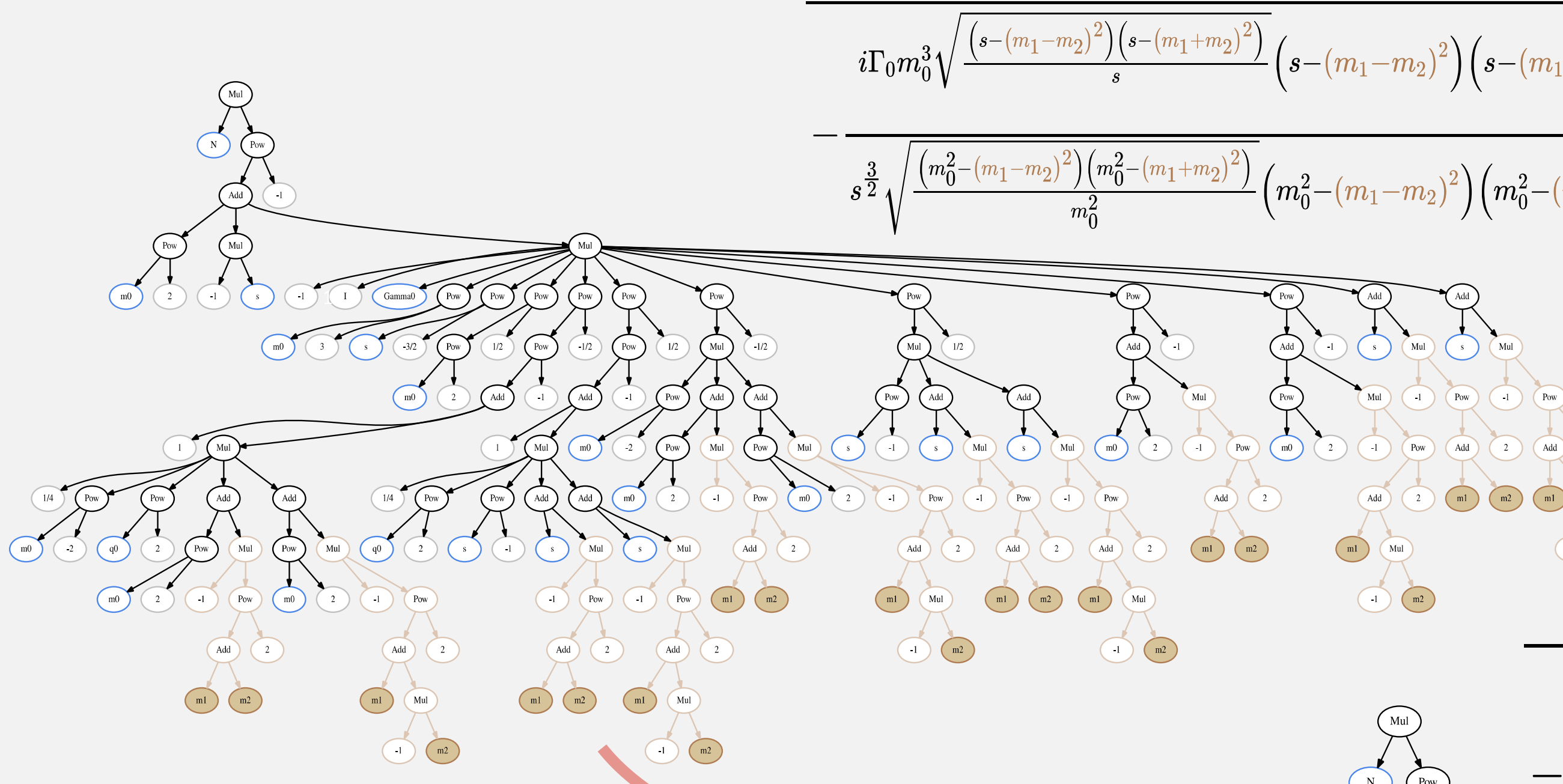


Main benefits:

- Physics separated from number crunching
- Switch back-end without changing analysis code
- Symbolic expressions result in **self-documenting workflow**
- Computations outsourced to fast, optimized back-ends with a large user-base
- Out-of-the-box GPU and multithreading support

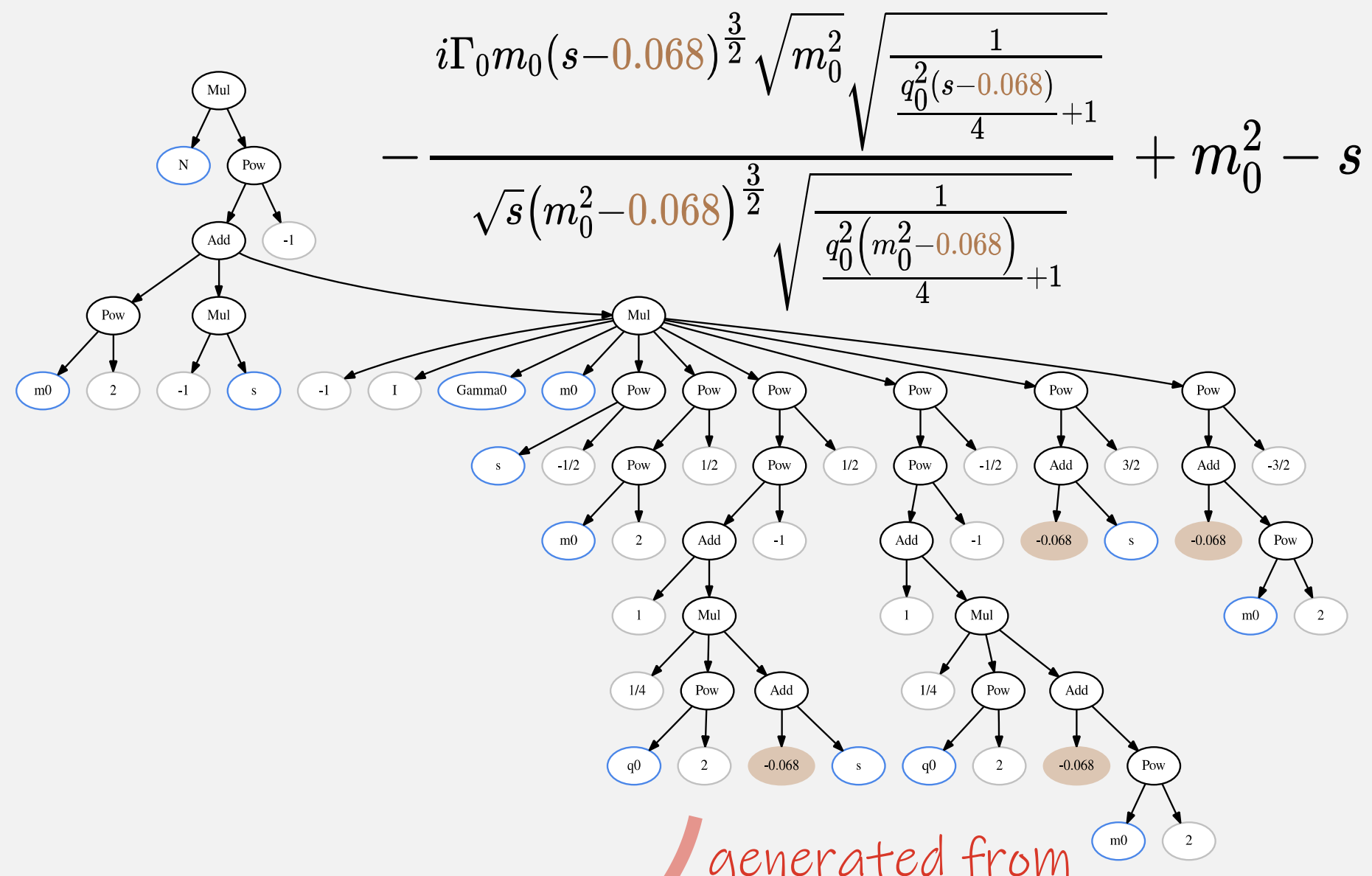
CAS simplifications result in performance boosts

$$\frac{N}{-\frac{i\Gamma_0 m_0^3\sqrt{\frac{\left(s-(m_1-m_2)^2\right)\left(s-(m_1+m_2)^2\right)}{s}}\left(s-(m_1-m_2)^2\right)\left(s-(m_1+m_2)^2\right)\sqrt{m_0^2}\sqrt{\frac{1}{\frac{q_0^2\left(s-(m_1-m_2)^2\right)\left(s-(m_1+m_2)^2\right)}{4s}+1}}}{s^{\frac{3}{2}}\sqrt{\frac{\left(m_0^2-(m_1-m_2)^2\right)\left(m_0^2-(m_1+m_2)^2\right)}{m_0^2}}\left(m_0^2-(m_1-m_2)^2\right)\left(m_0^2-(m_1+m_2)^2\right)\sqrt{\frac{1}{1+\frac{q_0^2\left(m_0^2-(m_1-m_2)^2\right)\left(m_0^2-(m_1+m_2)^2\right)}{4m_0^2}}}}+m_0^2-s}$$



CAS simplifies expression tree!

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

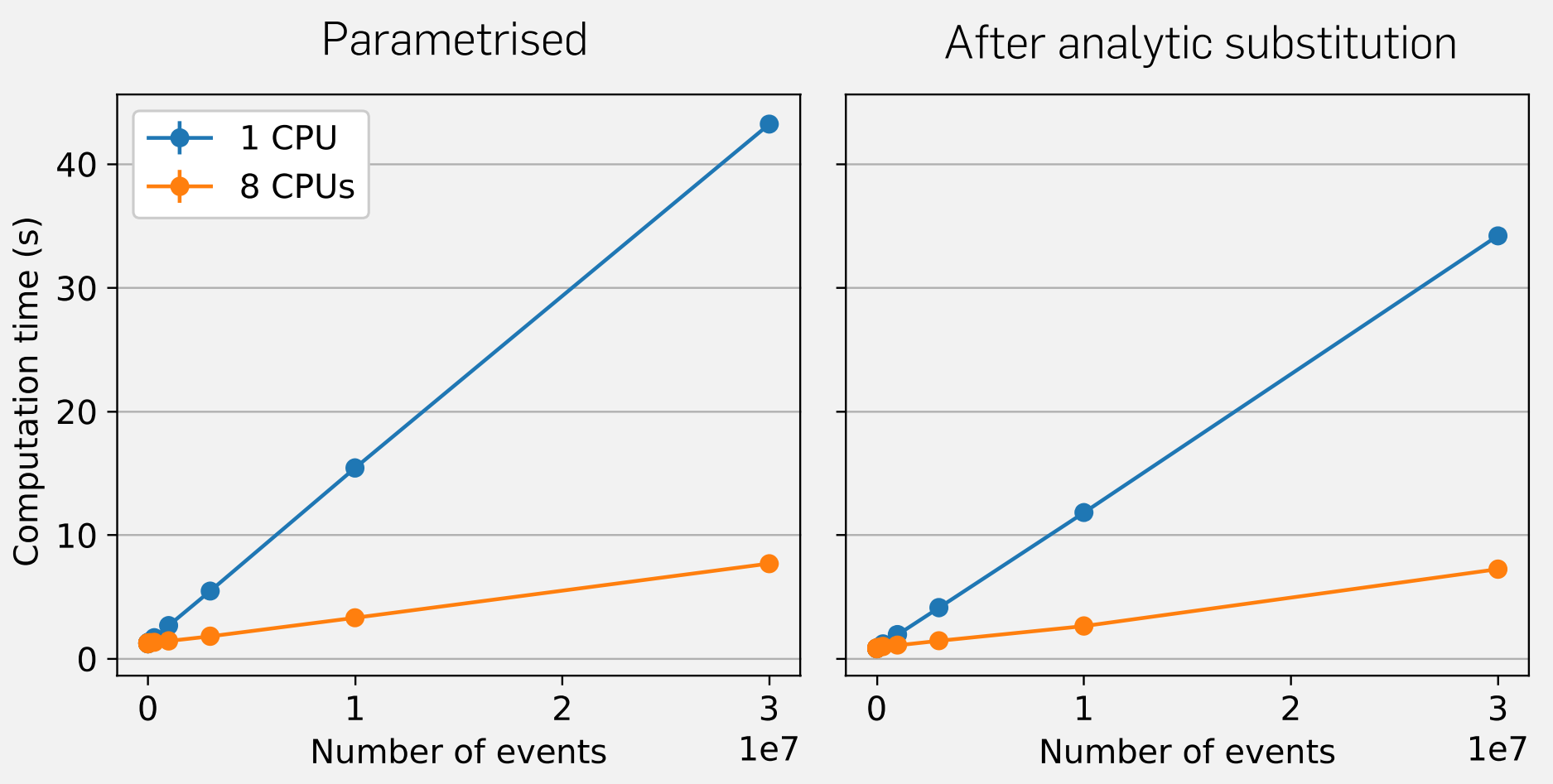


Example
Amplitude model with
12 resonances and 59 parameters

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

Backend: JAX
1 CPU: Intel i7-8750H 2.20GHz

→ 25% faster computation time

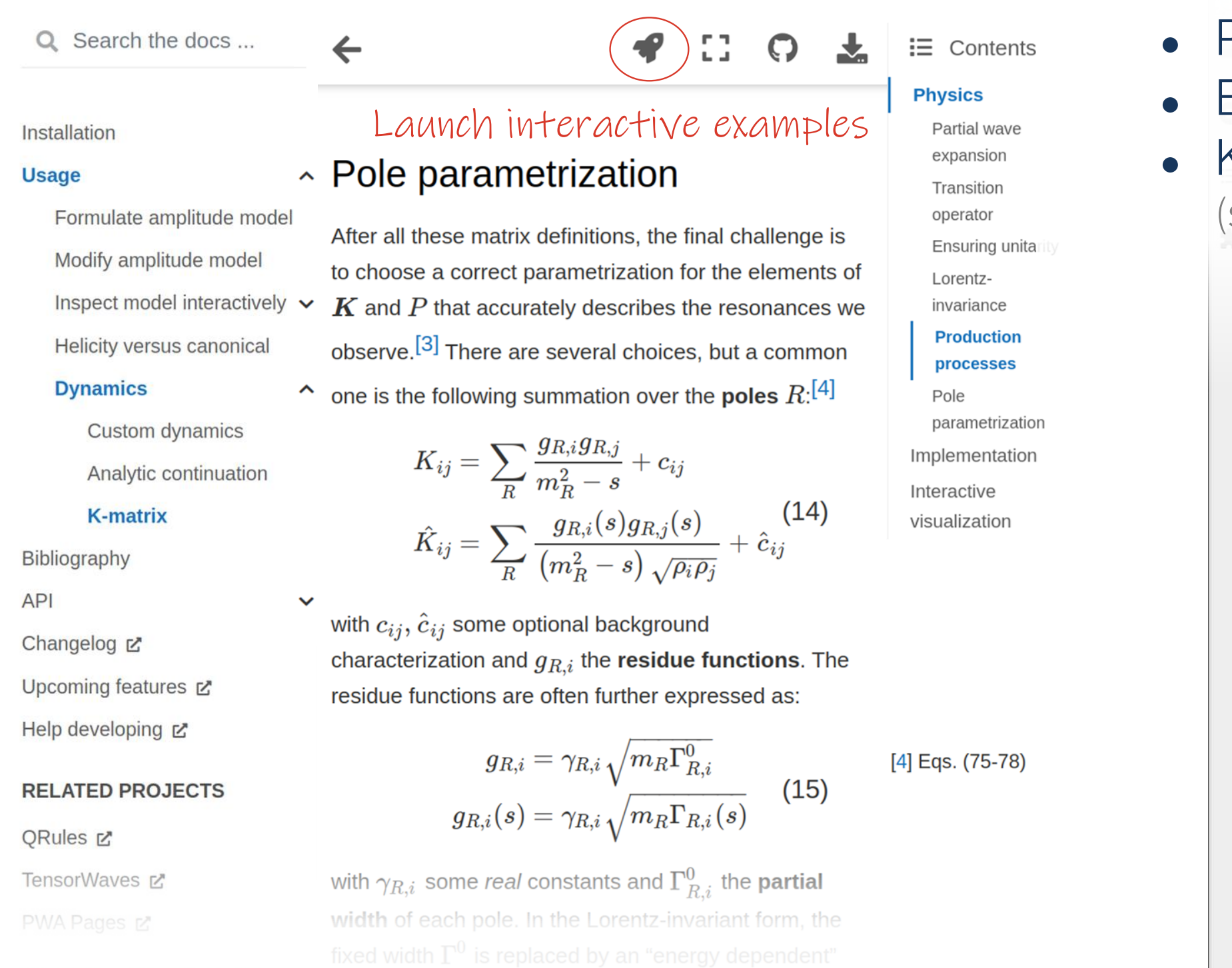


generated from expression tree

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

Self-documenting workflow

compwa-org.rtfd.io



Installation
Usage
Formulate amplitude model
Modify amplitude model
Inspect model interactively
Helicity versus canonical
Dynamics
Custom dynamics
Analytic continuation
K-matrix
Bibliography
API
Changelog
Upcoming features
Help developing

RELATED PROJECTS
QRules
TensorWaves
PWA Pages

Pole parametrization

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

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

with $c_{ij}, \hat{c}_{ij}$ some optional background characterization and $g_{R,i}$ the **residue functions**. The residue functions are often further expressed as:

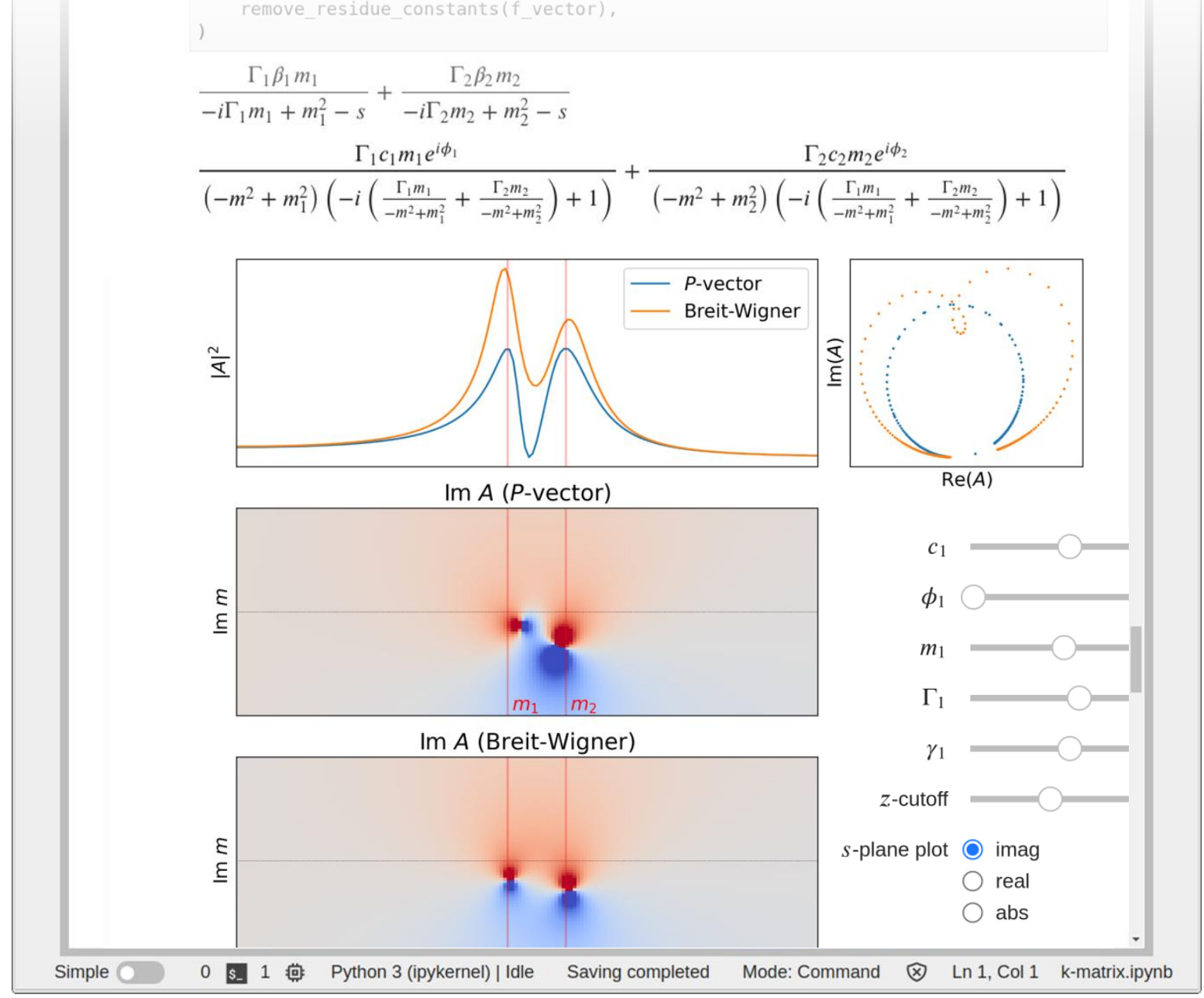
$$g_{R,i} = \gamma_{R,i}\sqrt{m_R\Gamma^0_{R,i}} \qquad g_{R,i}(s) = \gamma_{R,i}\sqrt{m_R\Gamma_{R,i}(s)} \tag{15}$$

[4] Eqs. (75-78)

with $\gamma_{R,i}$ some *real* constants and $\Gamma^0_{R,i}$ the **partial width** of each pole. In the Lorentz-invariant form, the fixed width Γ^0 is replaced by an "energy dependent"

Physics
Partial wave expansion
Transition operator
Ensuring unitarity
Lorentz-invariance
Production processes
Pole parametrization
Implementation
Interactive visualization

- Powered by symbolic expressions
- Explanations of implemented physics
- Kind of an **interactive book** in the browser
  (see Executable Book Project)



- Thoroughly integrated with codebase
- Continuously tested links and code

Spin-off project: polarimetry vector field of $\Lambda_c^+ \to pK^-\pi^+$

Polarimeter vector field $\vec{\alpha}$ for:

- determination of polarisation
- improvement of the sensitivity of amplitude models

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

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

ComPWA offers the flexibility to compute vector fields and easily test several parametrisations

lc2pkpi-polarimetry.docs.cern.ch

DOI 10.48550/arXiv.2301.07010