

Student Lightning Talk

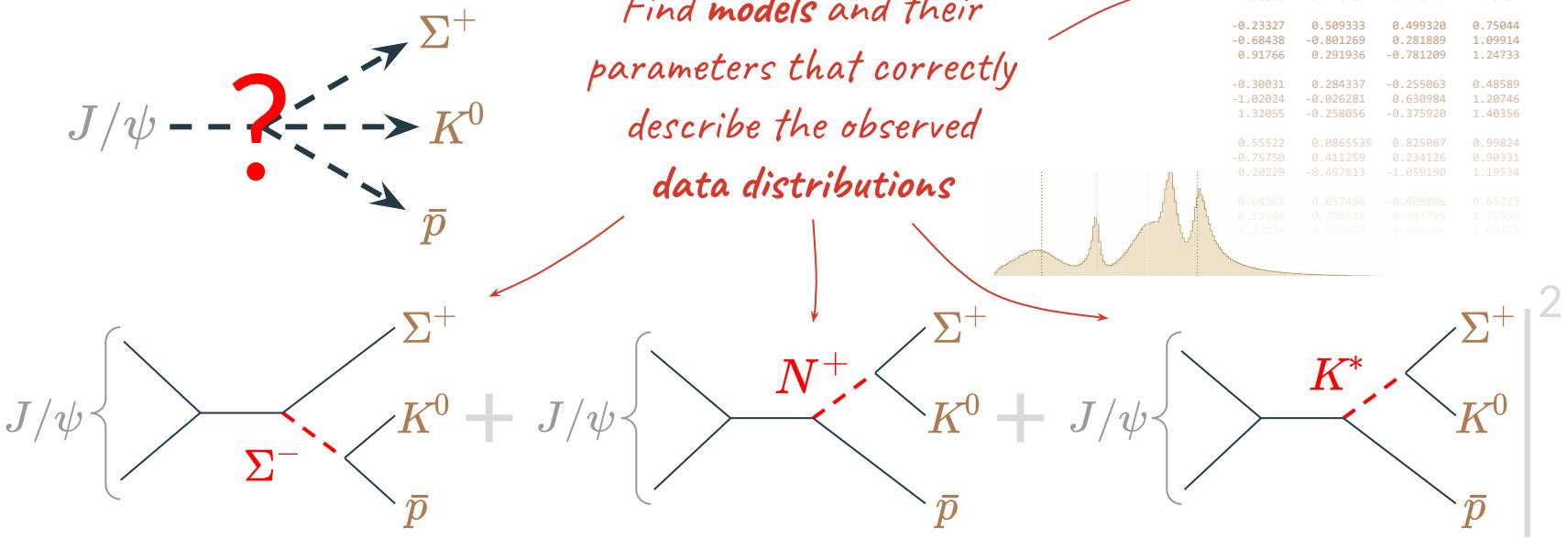
Speeding up differentiable programming with a Computer Algebra System

Remco de Boer
Ruhr University Bochum

7 September 2022
CERN School of Computing

Amplitude analysis

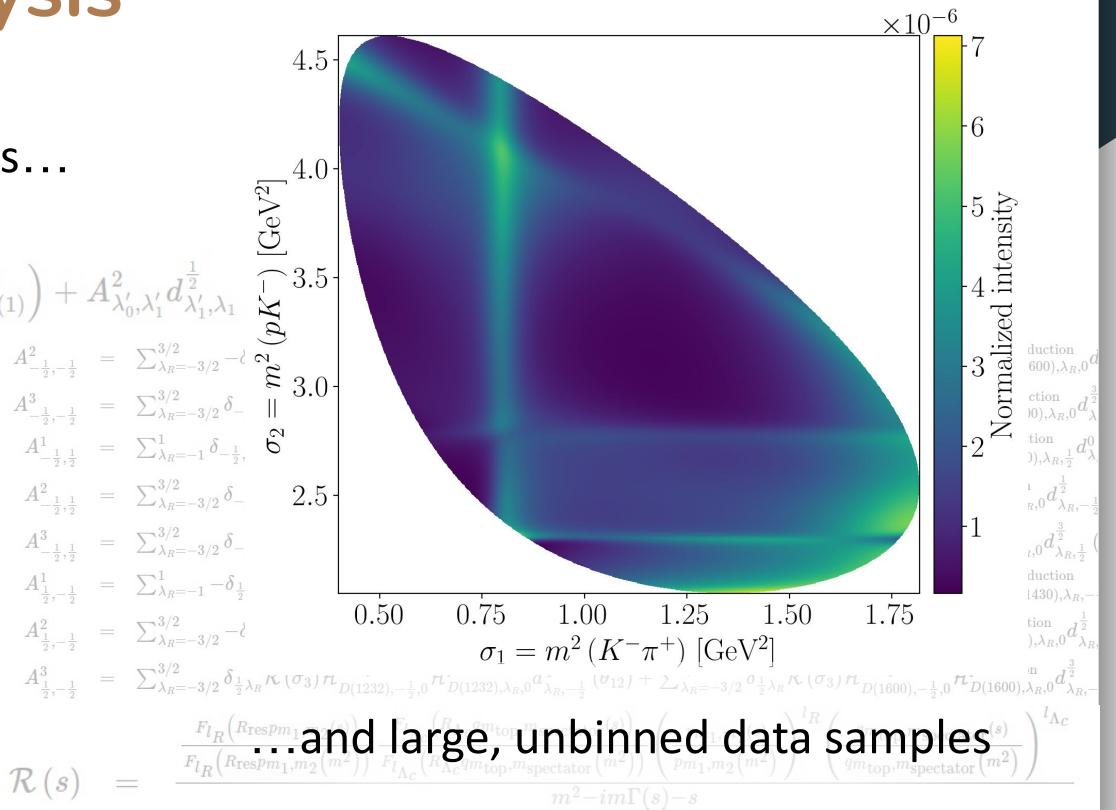
Aim: study of intermediate hadronic states



Amplitude analysis

Very large model descriptions...

$$\sum_{\lambda'_0=-1/2}^{1/2} \sum_{\lambda'_1=-1/2}^{1/2} A_{\lambda'_0, \lambda'_1}^1 d_{\lambda'_1, \lambda_1}^{\frac{1}{2}} \left(\zeta_{1(1)}^1 \right) d_{\lambda_0, \lambda'_0}^{\frac{1}{2}} \left(\zeta_{1(1)}^0 \right) + A_{\lambda'_0, \lambda'_1}^2 d_{\lambda'_1, \lambda_1}^{\frac{1}{2}} \left(\zeta_{1(1)}^1 \right) d_{\lambda_0, \lambda'_0}^{\frac{1}{2}} \left(\zeta_{1(1)}^0 \right)$$
$$\zeta_{2(1)}^0 = -\text{acos} \left(\frac{-2m_0^2(-m_1^2-m_2^2+\sigma_3)+(m_0^2+m_1^2-\sigma_1)(m_0^2+m_2^2-\sigma_2)}{\sqrt{\lambda(m_0^2, m_0^2, \sigma_2)} \sqrt{\lambda(m_0^2, \sigma_1, m_1^2)}} \right)$$
$$\zeta_{2(1)}^1 = \text{acos} \left(\frac{2m_1^2(-m_0^2-m_3^2+\sigma_3)+(m_0^2+m_1^2-\sigma_1)(-m_1^2-m_3^2+\sigma_2)}{\sqrt{\lambda(m_0^2, m_1^2, \sigma_1)} \sqrt{\lambda(\sigma_2, m_1^2, m_3^2)}} \right)$$
$$\zeta_{3(1)}^0 = \text{acos} \left(\frac{-2m_0^2(-m_1^2-m_3^2+\sigma_2)+(m_0^2+m_1^2-\sigma_1)(m_0^2+m_3^2-\sigma_3)}{\sqrt{\lambda(m_0^2, m_1^2, \sigma_1)} \sqrt{\lambda(m_0^2, \sigma_3, m_3^2)}} \right)$$
$$\zeta_{3(1)}^1 = -\text{acos} \left(\frac{2m_1^2(-m_0^2-m_2^2+\sigma_2)+(m_0^2+m_1^2-\sigma_1)(-m_1^2-m_2^2+\sigma_3)}{\sqrt{\lambda(m_0^2, m_1^2, \sigma_1)} \sqrt{\lambda(\sigma_3, m_1^2, m_2^2)}} \right)$$



Differentiable programming

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



e.g. gradient
descent algorithm



Not just Machine Learning!

Can be used for any fast numerical computations

Differentiable programming

Some of the techniques these back-ends offer:

- Vectorization
- Just-in-time compilation
- Automatic differentiation
- Support for multithreading, GPUs, ...

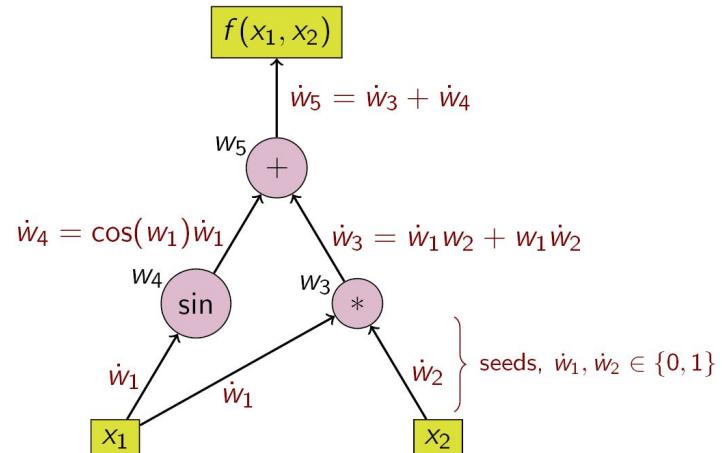


```
for (i = 0; i < rows; i++): {
    for (j = 0; j < columns; j++): {
        c[i][j] = a[i][j]*b[i][j];
    }
}

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

A red arrow points from the handwritten text "Heavy lifting by optimized backend" to the Python code block.

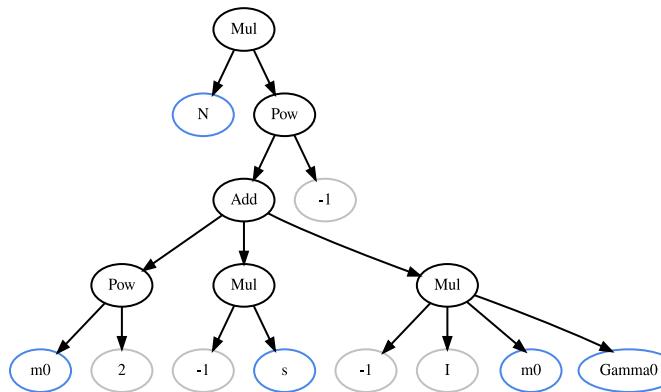
Heavy lifting by optimized backend





Computer Algebra System

Can we narrow the gap between theory and code?



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

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

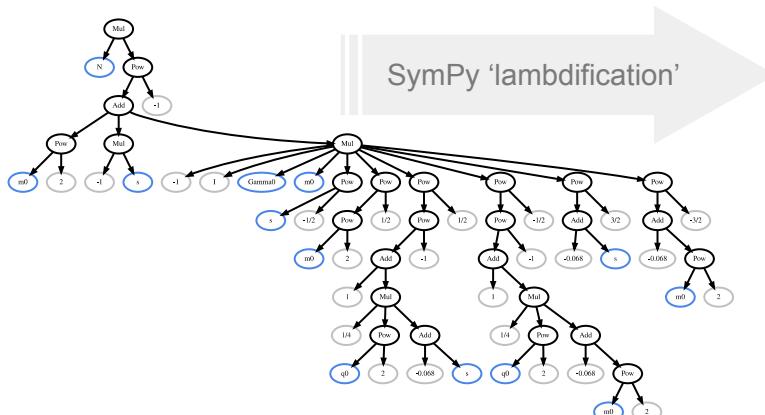
Can serve as template to computational back-ends!

CAS represents expression as a tree



Computer Algebra System

Can we narrow the gap between theory and code?



Expression tree can serve as template to computational back-ends

Fortran

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

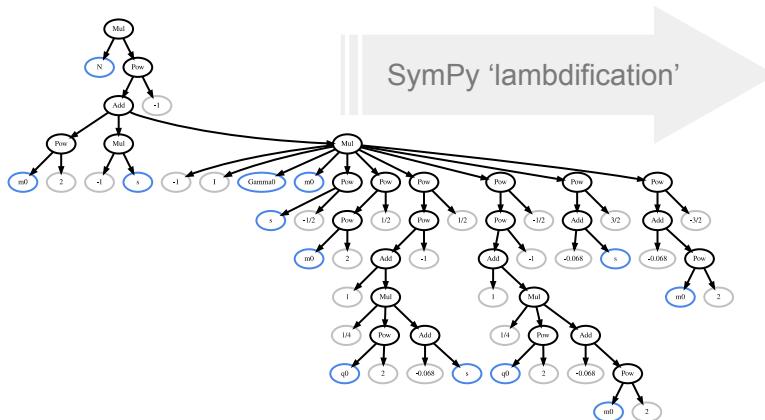
my_expr = N/(-cmplx(0,1)*Gamma0*m0**3*sqrt((s - 0.25d0)*(s - 0.01d0)/s)* &
           (1 + (1.0d0/4.0d0)*(m0**2 - 0.25d0)*(m0**2 - 0.01d0)/m0**2)*(s - &
           0.25d0)*(s - 0.01d0)*sqrt(m0**2)/(s***(3.0d0/2.0d0)*sqrt(m0**2 - &
           0.25d0)*(m0**2 - 0.01d0)/m0**2)*(1 + (1.0d0/4.0d0)*(s - 0.25d0)*( &
           s - 0.01d0)/s)*(m0**2 - 0.25d0)*(m0**2 - 0.01d0)) + m0**2 - s)

end function
```



Computer Algebra System

Can we narrow the gap between theory and code?



Expression tree can serve as template to computational back-ends

C++

```
// my_expr.h
#ifndef PROJECT__MY_EXPR_H
#define PROJECT__MY_EXPR_H
double my_expr (double Gamma0, double N, double m0, double s);
#endif

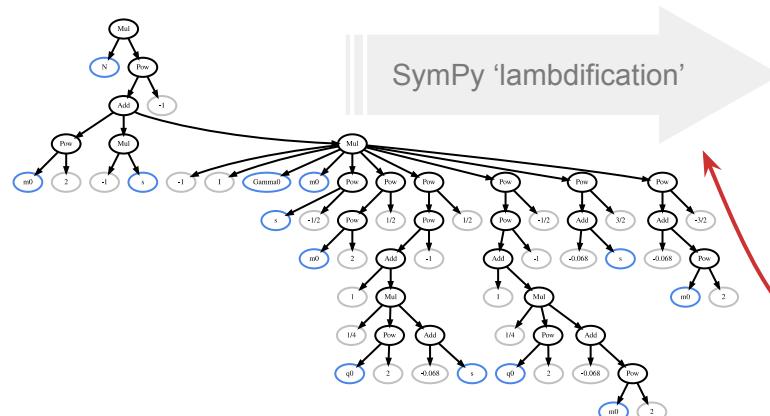
// my_expr.c
#include "my_expr.h"
#include <math.h>

double my_expr (double Gamma0, double N, double m0, double s) {
    double my_expr_result;
    my_expr_result = N/(-I*Gamma0* pow(m0, 3)*sqrt((s - 0.25)*(s - 0.01)/s)*(1 +
(1.0/4.0)*(pow(m0, 2) - 0.25)*(pow(m0, 2) - 0.01)/pow(m0, 2))*(s - 0.25)*(s -
0.01)*sqrt(pow(m0, 2))/(pow(s, 3.0/2.0)*sqrt((pow(m0, 2) - 0.25)*(pow(m0, 2) -
0.01)/pow(m0, 2)))*(1 + (1.0/4.0)*(s - 0.25)*(s - 0.01)/s)*(pow(m0, 2) -
0.25)*(pow(m0, 2) - 0.01) + pow(m0, 2) - s);
    return my_expr_result;
}
```



Computer Algebra System

Can we narrow the gap between theory and code?



Expression tree can serve as template to computational back-ends

Python with JAX

```
@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
    )
```

Any CAS simplifications
optimize the back-end code!

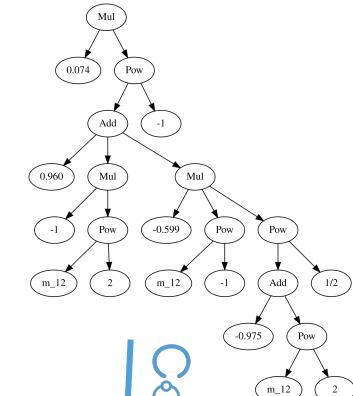
Com PWA The ComPWA Project

Set of Python libraries that:

- Standardize and automate amplitude analysis theory in CAS code
- Streamline and improve conversion from CAS to back-end
- Generate amplitude-based Monte Carlo samples
- Perform fits with different optimizers (Minuit2, SciPy, ...)

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

Any symbolic input



Com
PWA



ComPWA

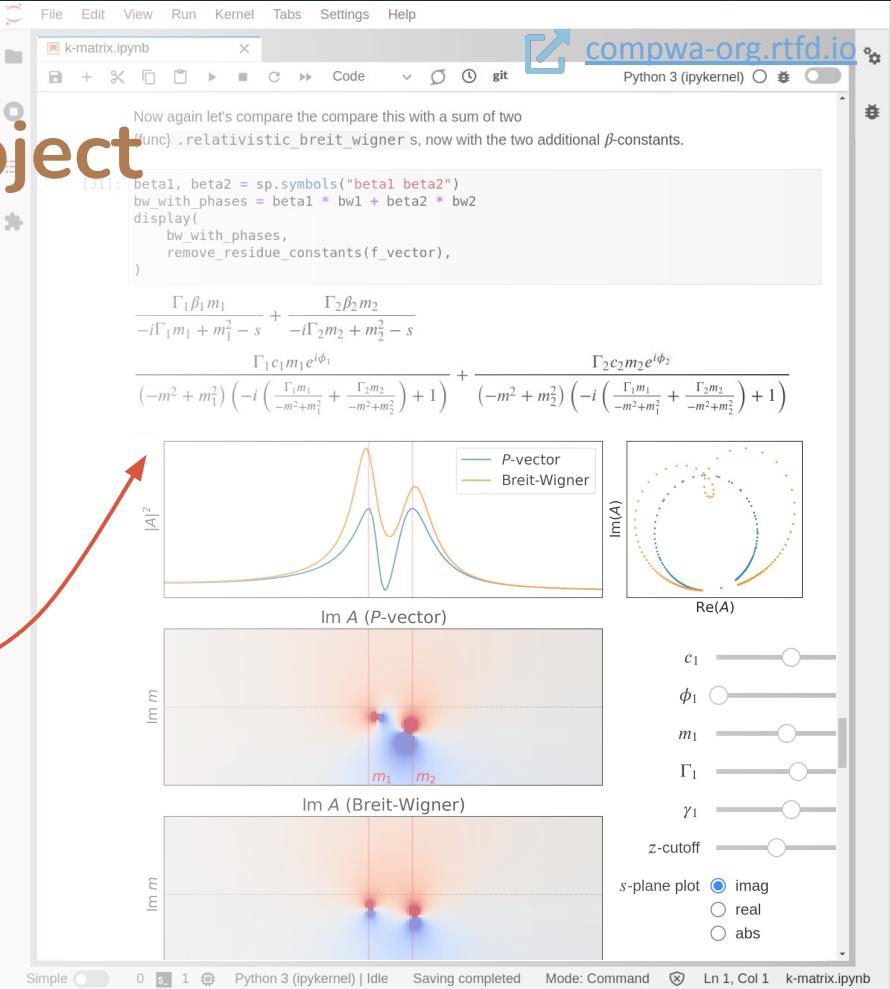
The ComPWA Project

Set of Python libraries that:

- Standardize and automate amplitude analysis
- Streamline and improve conversion from Feynman diagrams
- Generate amplitude-based Monte Carlo
- Perform fits with different optimizers (Minuit2, JAX)

Self-documenting — Any source code is a workflow

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



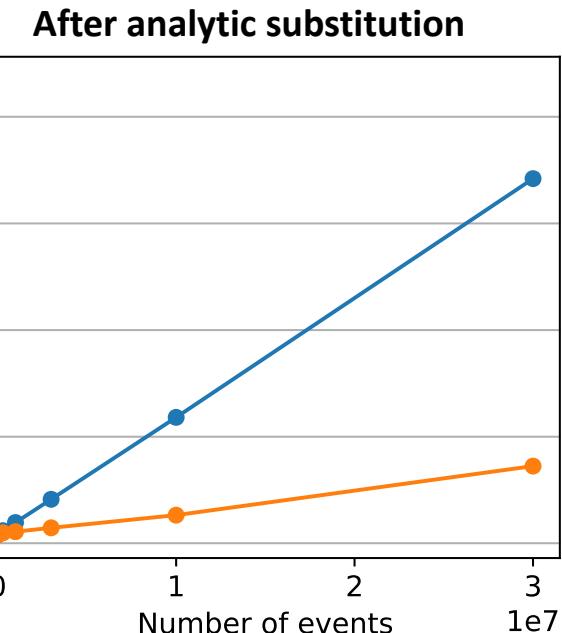
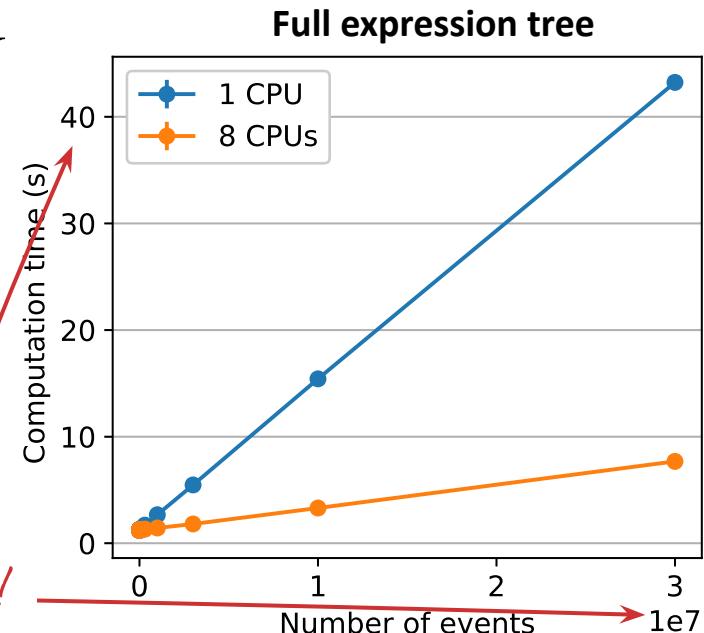
Com PWA The ComPWA Project

Amplitude model for $\Lambda_c \rightarrow p\pi K$
12 resonances, 59 parameters,
DPD alignment for 3 subsystems

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

Backend: JAX

*Great performance
for a single fit iteration!*



Com PWA The ComPWA Project

Amplitude model for $\Lambda_c \rightarrow p\pi K$
12 resonances, 59 parameters,
DPD alignment for 3 subsystems

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

Backend: JAX

Great performance
for a single fit iteration!

