中国科学院大学
University of Chinese Academy of Sciences

Amplitude Analysis Tools at BESIII

Yi Jiang (蒋艺)
University of Chinese Academy of Sciences

Hadron 2023

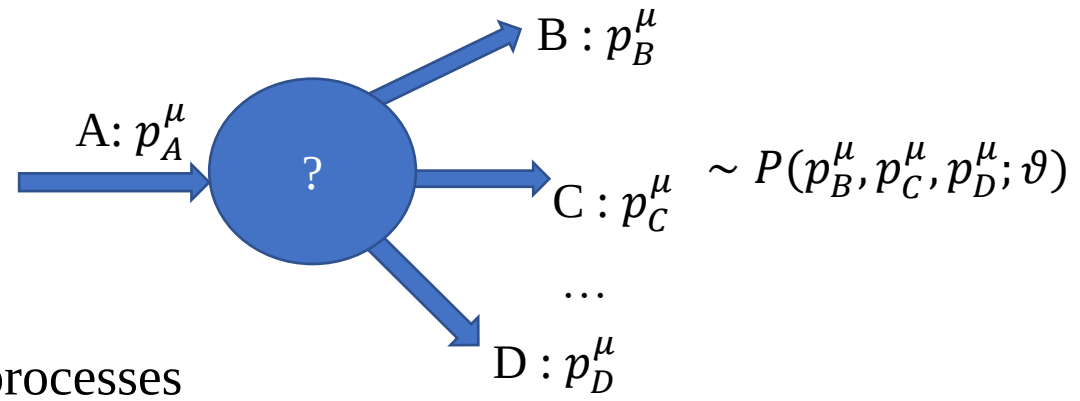
7 June 2023

Outline

- Introduction
- Designs of PWA library
- TF-PWA
- Automatic Differentiation and its application in TFPWA.
- Performances of TF-PWA.
- Current development
- Summary

Introduction

- Amplitude analysis / Partial wave analysis (PWA) is a powerful method to study multi-body decay processes, e.g.
  - to search for (exotic) resonances and measure their properties
  - to understand CP violation over phase space



- Most of previous fitters are designed for special processes or are time-consuming.

- A general PWA framework using modern acceleration technology (such as GPU, AD, …) is eagerly needed.

Main tools in BESIII

- Closed source / hand coded
  - Tensor formulism: most of charm decays. [$D^+ \to K_S^0\pi^+\pi^0\pi^0$: 2305.15879]
  - Helicity formulism: [$e^+e^- \to \omega\pi^+\pi^-$: 2303.09718]
- GPUPWA:
  - First PWA tools use GPU.
  - Used in most of partial wave analysis.
  - [$J/\psi \to \gamma\eta\eta$: PRD.87.092009(2013), $J/\psi \to \gamma\eta\eta'$: PRD.106.072012(2022)
- FDC-PWA:
  - Use Feynman Diagram Calculation.
  - Used in some baryon decay.
  - [$\psi' \to p\bar{p}\eta$:PRD.88.032010(2013), $e^+e^- \to pK^-\bar{\Lambda}$: 2303.01989]
- TF-PWA:
  - Main topics in this talk
  - [$\Lambda_c^+ \to \Lambda\pi^+\pi^0$:JHEP12(2022)033]
- Other tools:
  - Amptools: Some charmonium. [$\chi_{c1} \to \eta\pi^+\pi^-$: PRD.95.032002(2017)]
  - PAWIAN: [$J/\psi \to \phi\eta\eta$:10.13154/294-6468]
  - ComPWA: [$D^0 \to K_SK^+K^-$: 2006.02800]

Properties and Requirement of PWA

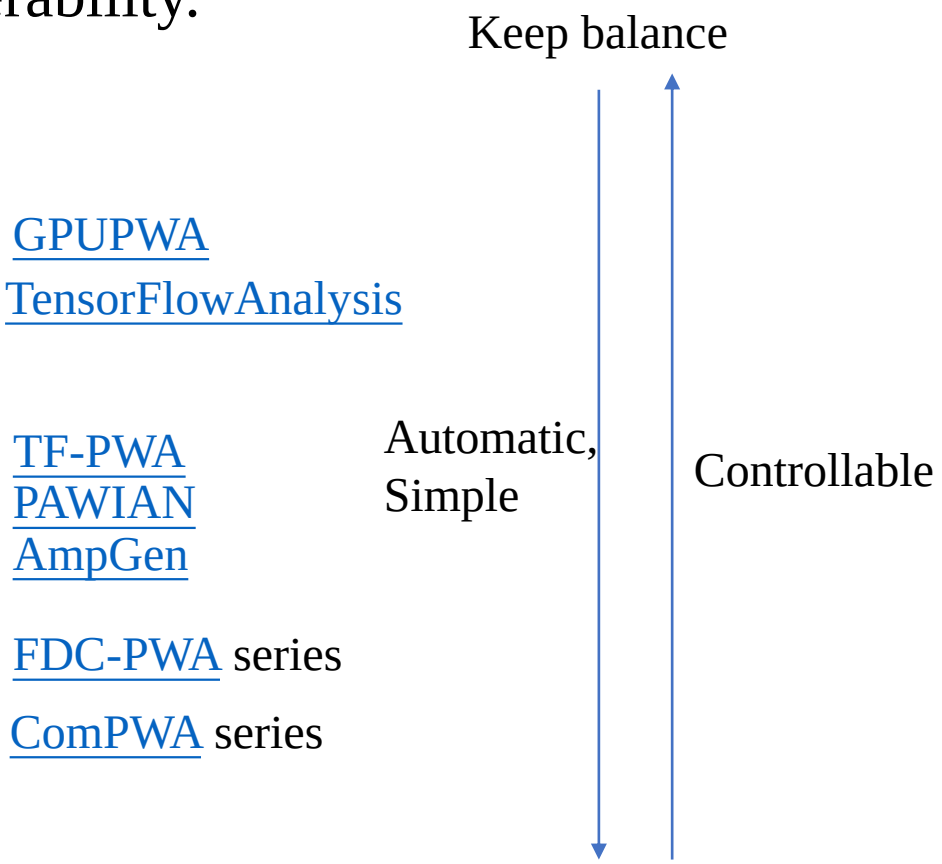
- Complex Formula
  - Avoid hand written code, automatic.
  - Rule based amplitude evaluation.
  - Constraints in special process.
- Multiple Dimension.
  - Study relation between many variables: masses and angles.
  - Proper way to consider resolution. (see backup)
  - Large size of MC for integration to normalize the PDF.
- Large size of data (e.g.,10B J/ψ in BESIII)
  - Fast calculation to reduce time cost.
  - Distribute the calculation into multi devices.

Configurable

High performance calculation.

Configuration

- Why Configurable?
  - Global representation for automation and transferability.
  - General way to support more decays
- Difference level
  - No configuration:
    - Hand coded / code templates
  - Decay Card like:
    - Key-value / command-parameters / structured .
    - Specify all possible decays (interactions).
    - With addition simplification rules.
  - Auto search:
    - Provide a large particles dataset.
    - Use rules to find all possible intermediate states.
    - Filter with requirement.



Symbolic and numeric approaches

- Symbolic approach
  - Requires a Computer Algebra System (CAS) to simplify formulas.
  - Write/generate code from CAS outputs
  - Procedure:
    - configuration -> CAS -> formula -> generated code -> function -> amplitude
  - Simplify the formula is difficult and time-consuming.
- Numeric approach
  - Combine Function directly,
  - Rule based evaluation
  - Procedure:
    - configuration -> function call -> amplitude
  - Without simplified formula, more computation might required
  - Allow caching rule to reduce computation

FDC-PWA series (REDUCE)
ComPWA series (SymPy)

AmpGen
GPUPWA

Self hold tensor library

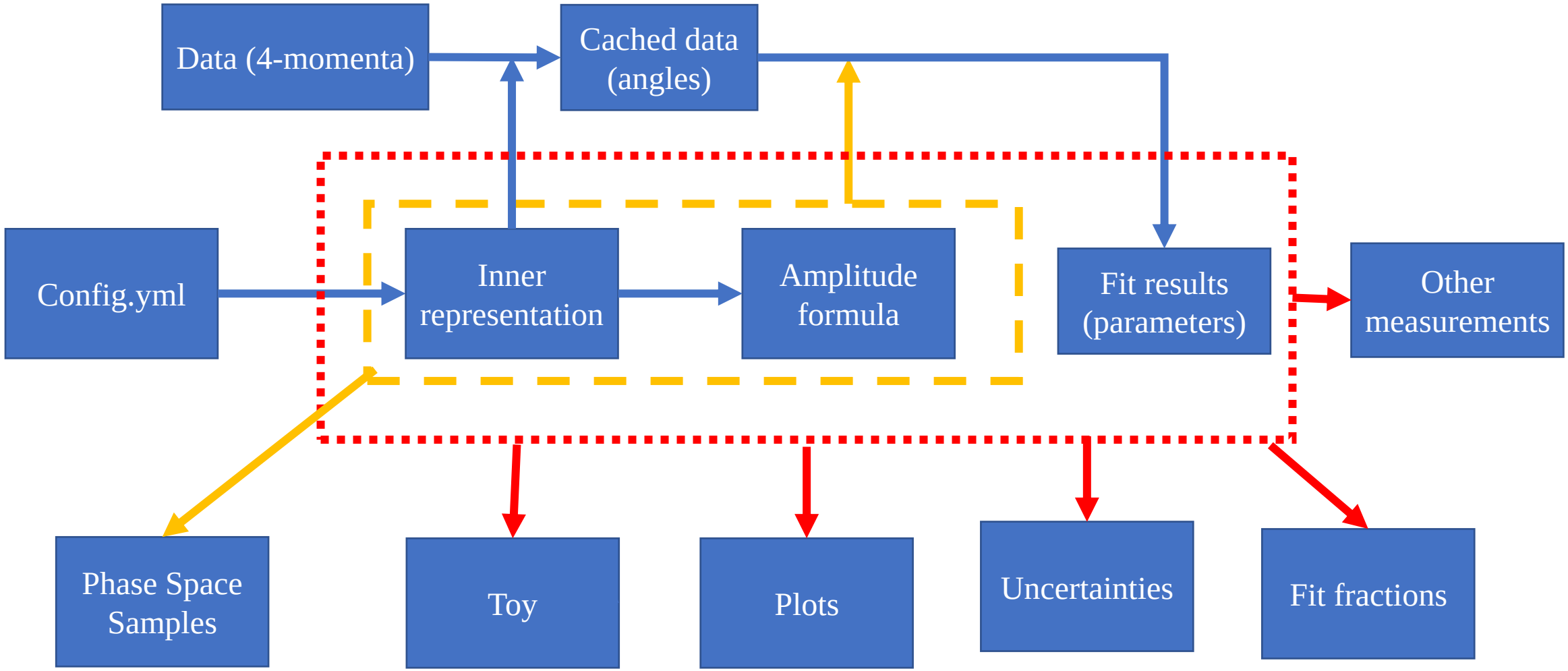
TF-PWA
PAWIAN
TensorFlowAnalysis

TFPWA

TF-PWA: Partial Wave Analysis with TensorFlow

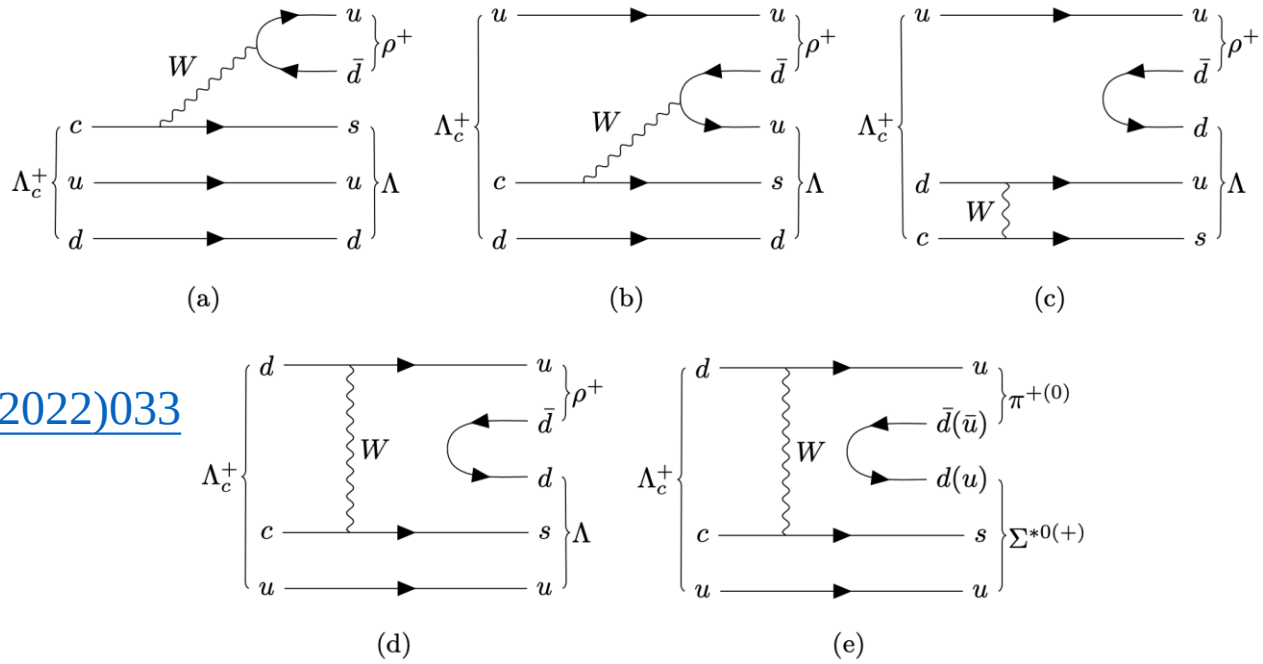
- Fast
  - GPU based
  - Vectorized calculation
  - Automatic differentiation
    - Quasi-Newton Method: scipy.optimize
- General
  - Custom model available
- Easy to use
  - Simple configuration file (example provided)
  - Most of the processing is automatic
  - All necessary functions implemented
  - Developing more functions
- Open access and well supported https://github.com/jiangyi15/tf-pwa

Configuration as global representation



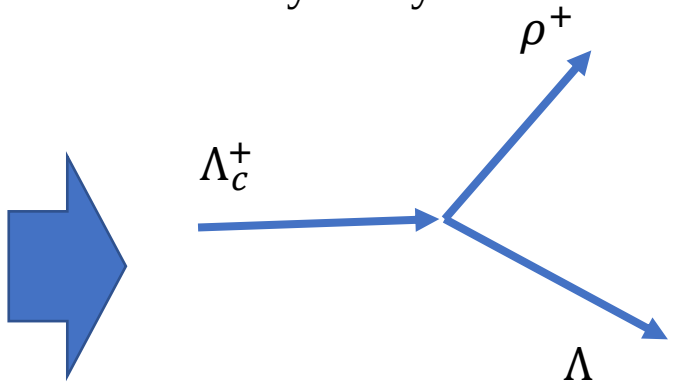
Configuration

- What is needed?
  - Particles (Resonances, line) and their properties
  - Decays (interaction, vetex) and their properties
- Possible process in this Λ_c decay $\Lambda_c^+ \to \Lambda\pi^+\pi^0$



JHEP12(2022)033

2-body decay



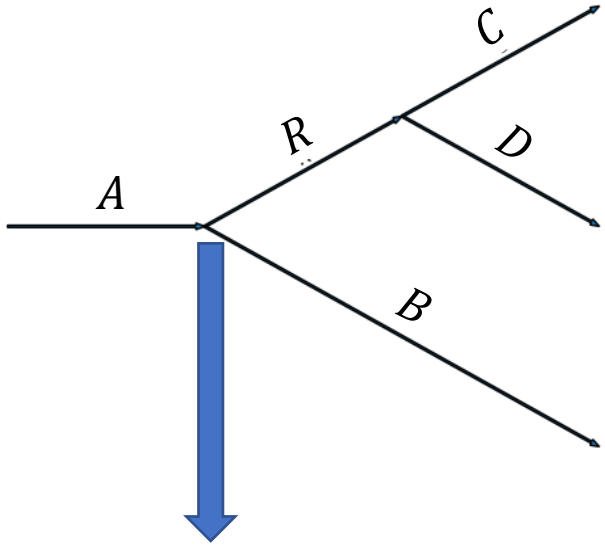
Serialize as

```
Lambda_c: [
  [Lambda, rho],
  [Sigma_starp, pi0 ],
  [Sigma_star0, pip ],
]
```

Config.yml In TF-PWA

```
rho:
  J: 1
  P: -1
  mass:  0.77511
  width:  0.1491
  model: GS_rho
…
```

Helicity formalism

- 

$$A^R_{\lambda_A,\lambda'_B,\lambda'_C,\lambda'_D}$$

=

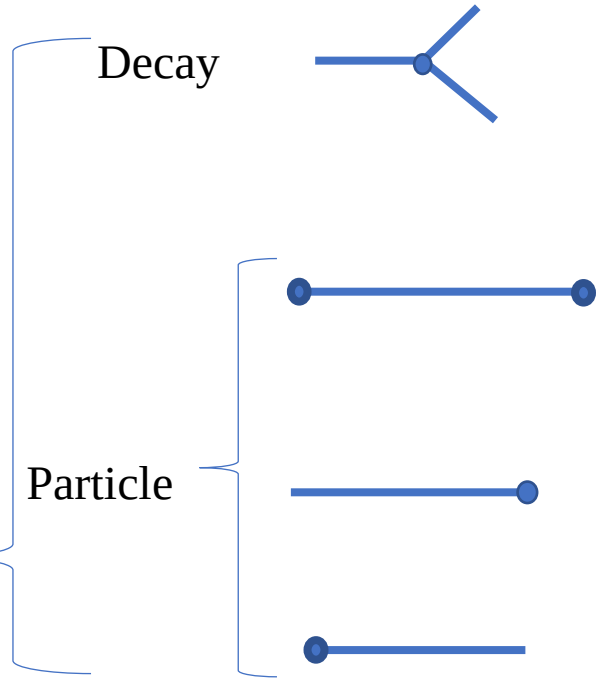
$$\sum_\lambda H_{\lambda_R\lambda_B} D^{j_A\star}_{\lambda_A,\lambda_R-\lambda_B}(\varphi_1,\theta_1,0) R(M) H_{\lambda_C,\lambda_D} D^{j_R\star}_{\lambda_R,\lambda_C-\lambda_D}(\varphi_2,\theta_2,0)$$

$$D^{j_B\star}_{\lambda_B,\lambda_{B'}}(\alpha_B,\beta_B,\gamma_B) D^{j_C\star}_{\lambda_C,\lambda_{C'}}(\alpha_C,\beta_C,\gamma_C) D^{j_D\star}_{\lambda_D,\lambda_{D'}}(\alpha_D,\beta_D,\gamma_D)$$

$$\frac{d\sigma}{d\Phi} \propto \sum_{\lambda_A} \sum_{\lambda_B,\lambda_C,\lambda_D} \left| \sum_R A^R_{\lambda_A,\lambda_B,\lambda_C,\lambda_D} \right|^2$$

Automatically calculated from decay structure

Feynman rules



User defined

$$A^{0\to1+2} = H_{\lambda_1,\lambda_2} D^{j_0\star}_{\lambda_0,\lambda_1-\lambda_2}(\varphi,\theta,0)$$

Wigner-D matrix

$$R(M) = \frac{1}{m_0^2 - M^2 - i m_0 \Gamma}, \cdots$$

1 or $\rho = 1 + \vec{p}\cdot\vec{\sigma}$

$$D^{j_1\star}_{\lambda_1,\lambda_{1'}}(\alpha,\beta,\gamma)$$

alignment

probability: |$\mathcal{A}$|^2

Decay Group: $\mathcal{A} = \tilde{A}_1 + \tilde{A}_2 + \cdots$

Decay Chain: $\tilde{A} = A_1 R A_2 \cdots$

Decay: Wigner D-matrix, $A = HD^{*J}\ (\phi,\theta,0)$

Particle: Breit-Wigner: $R(m)$, user defined

Factor system: automatic factorization of amplitude

- Amplitude can be written as the combination of summation and production.

$$A = (\sum g_i A_i)(\sum g'_j A'_j)\cdots \Rightarrow A = \sum_{ij} (g_i g'_j)(A_i A'_j)$$

$$G_{(i,j)} = g_i g'_j = \begin{cases} 1 & i,j = (a,b) \\ 0 & i,j \neq (a,b) \end{cases} \Rightarrow A = B_{(i,j)} = A_i A'_j$$

- No need known for the exact formula A_i, just use the parameters g_i.
- Some special treatment is implemented as option for better performances. (comparing in Page 21)
  - Amplitude caching method
    - mass dependent:

      $A(p_i^\mu) = \sum g_i R(m) A_i(p_i^\mu) \Rightarrow A(m) = \sum g_i R(m) B_i$

      Allow fit parameters related
    - factor only: (only for MC integration)
      - $\sum |A|^2 \rightarrow G_i G_j^* (\sum C_{ij})$

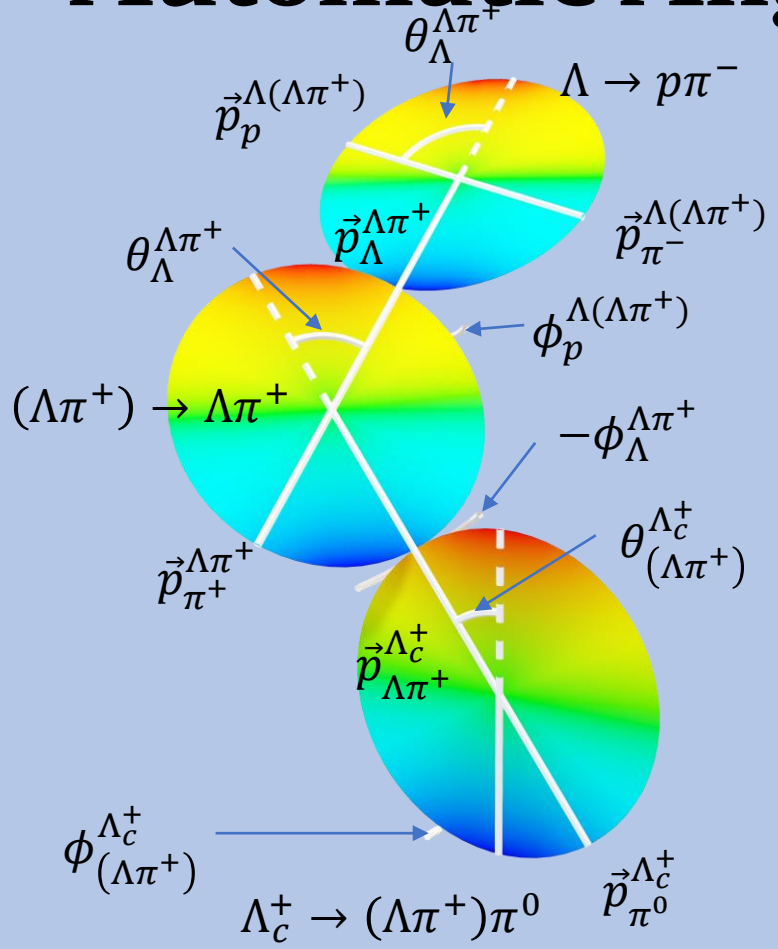
        calculate only once
      - Required all shape parameters fixed
  - Special in simultaneous fit
    - Mixed likelihood, avoid small size data

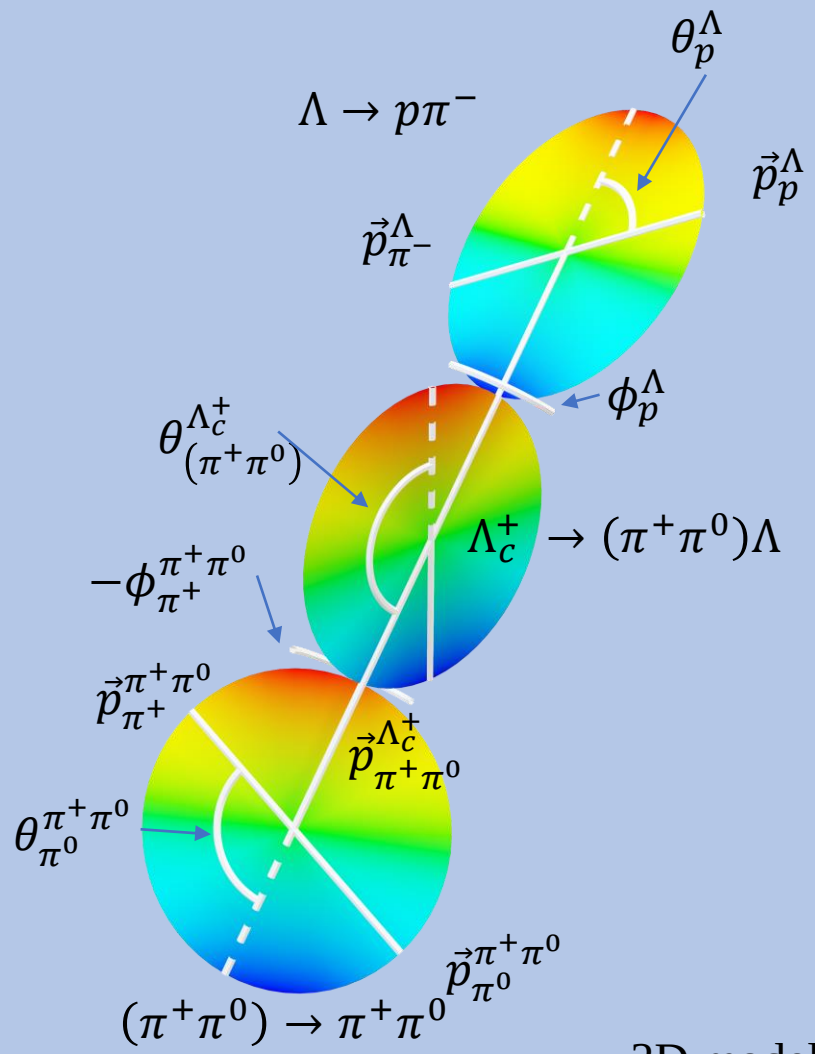
$$-\ln L_1 - \ln L_1 = -\sum_{data1+data2} \ln|A|^2 + N_1 \ln \sum_{mc1} |A|^2 + N_2 \ln \sum_{mc2} |A|^2$$

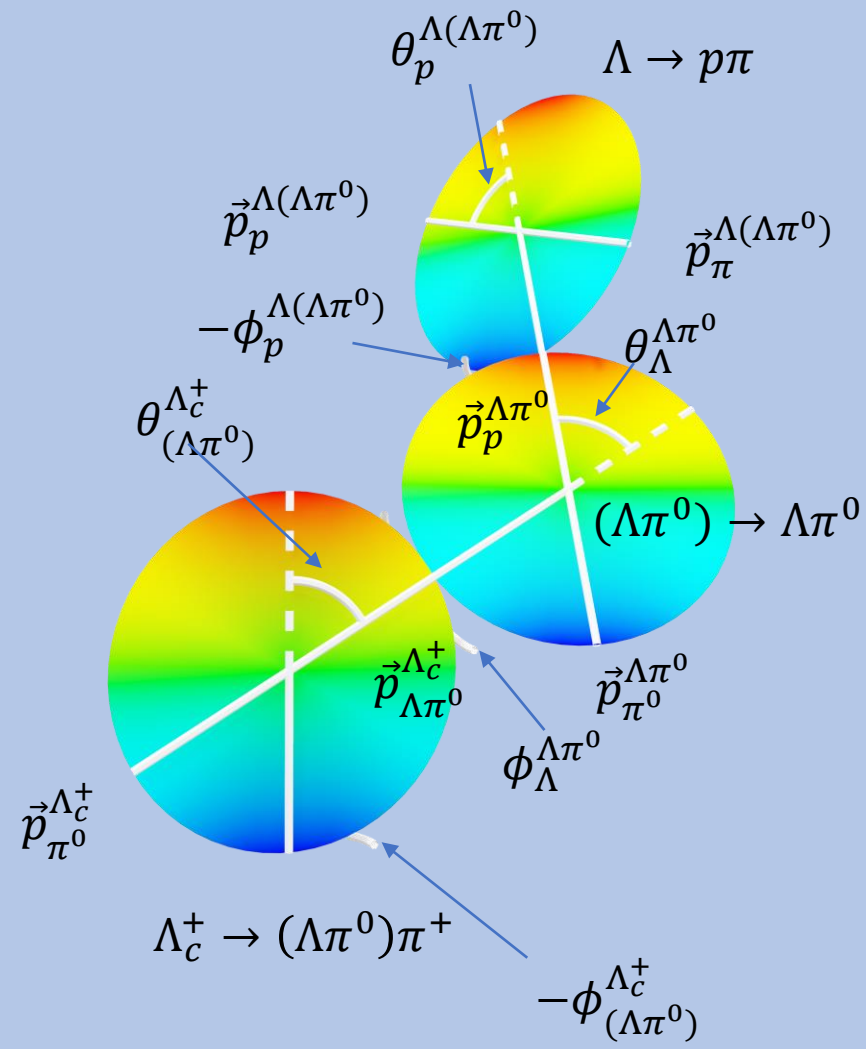
Add control options,
the processes are automatics.

Automatic Angle Plot

$\vec{p}_B^{\,A}$ means momentum of B in the rest frame of A
ϕ means the rotation is anticlockwise, while $-\phi$ for clockwise
The sign Is dependent on data







Auto calculated by TF-PWA,
Only required the 4-monmenta

TF-PWA also provide reverse process:
Mass + helicity angle -> 4-monmenta

3D model generated by a script using TF-PWA.

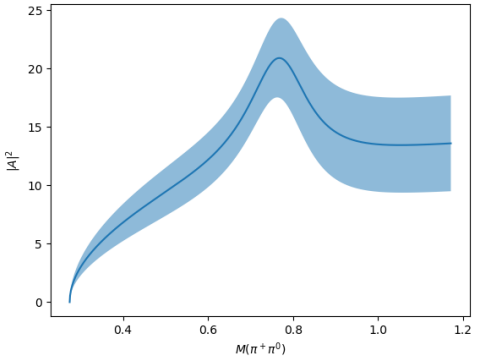
Amplitude as a function

```
f1 = config.get_particle_function("R1")
f2 = config.get_particle_function("R2")

m = f1.mass_linspace(1000)
# plot the first wave
amp = tf.abs(f1(m)[:,0] + f2(m)[:,0])**2
plt.plot(m, amp)
```

- Reverse process of angle calculation
  - Mass + Helicity angle -> 4- momenta
  - $(m_0, \phi_0, \theta_0, m_{12}, \phi_{12}, \theta_{12}) \xrightarrow{\text{transform}} p_1^\mu, p_2^\mu, p_3^\mu$
- Factor system:
  - Eval amplitude of special partial wave though control of parameters
  - $A(p_1^\mu, p_2^\mu, p_3^\mu) \xrightarrow{g_{i\neq j}=0} g_i A_i(p_1^\mu, p_2^\mu, p_3^\mu)$
- Combine Together: Lineshape function of special wave
  - Set angle to 0, $D_{m,m'}(0,0,0) = \delta_{m,m'}$ is constant.
  - Vary mass, then get the shape of masses
  - $f(m_{12}) = g_i A_i\ (m_0, \phi_0 = 0, \theta_0 = 0, m_{12}, m_{12}, \phi_{12} = 0, \theta_{12} = 0)$
  - No worries for the complex formula
- 2D function of amplitude
  - 2D plot Dalitz variables
  - 2D plot of Mass and $\cos\theta$

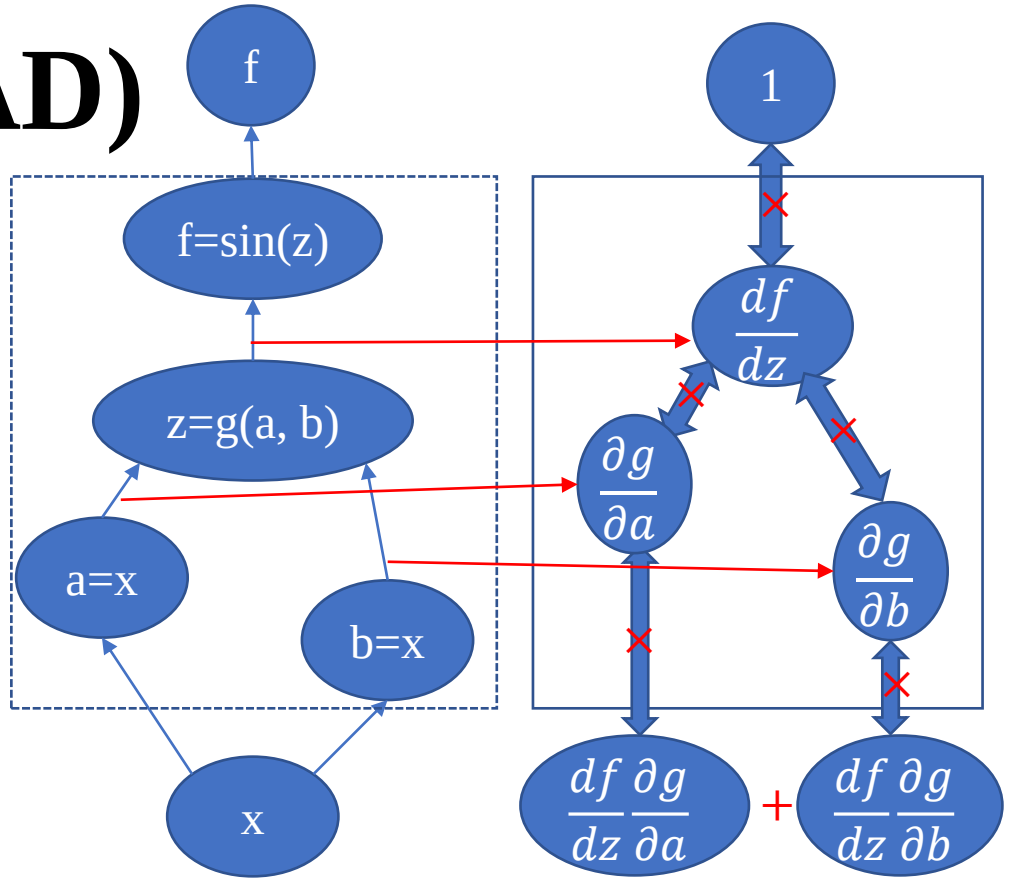
Uncertainties from error propagation



Automatic Differentiation(AD)

- Widely used in Optimize problem
  - Calculate gradient automatically
  - No need for exact formula.
  - Recorded the compute graphs.
    - Operator: used function (sin, g)
    - Intermediate value: (z= 1^1)
  - Mostly matrix form (Jacobian).
  - Just combine the operator (Chain Rules)
- Chain Rules
  - ×: $\frac{\partial f(g(x))}{\partial x_i} = \frac{df}{dg} \times \frac{\partial g}{\partial x_i}$
  - +: $\frac{\partial f(h(x), g(x))}{\partial x_i} = \frac{\partial f}{\partial h}\frac{\partial h}{\partial x_i} + \frac{\partial f}{\partial g}\frac{\partial g}{\partial x_i}$

The same rule based way for our amplitude calculation.



$$f = \sin(x^x) = \sin(z), z = x^x$$
$$g(a, b) = a^b$$

Automatic Differentiation (numerically):

$$\frac{df}{dx}(1) = \frac{df}{dz}(1^1)[\frac{\partial g}{\partial a}(1,1) + \frac{\partial g}{\partial b}(1,1)] = 0.5403$$

backward

AD in amplitude fit

- Minimize of $-\ln L\,(\vartheta)$
  - $-\frac{\partial \ln L}{\partial \vartheta}$ is the steepest descent direction, used by most of optimizer
  - Error matrix $V_{ij} = \left[-\frac{\partial^2 \ln L}{\partial \vartheta_i \partial \vartheta_j}\right]^{-1}$ can also be estimated though AD
  - AD advantage:
    - Automatic.
    - Fast estimation: Time cost for eval $-\ln L\,(\vartheta)$ and $-\frac{\partial \ln L}{\partial \vartheta}$ are on the same level.
    - Accurate gradient: more stable results.
  - AD disadvantage:
    - Require well defined gradients (continuous).
      - Avoid step function, delta function.
    - Only support function with predefined gradients.
      - Use TensorFlow only, but also have an interface to define functions.
    - Large (GPU) memory cost for recording intermediate values.
      - Split data into small batches (discuss later).

AD for error propagation

- Error propagation
  - $\sigma_y^2 = \frac{\partial y}{\partial x}\sigma_x^2\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial x}$ can be calculated by AD
  - Simple interface (see right)
  - Example: uncertainties of fit fractions in TF-PWA
- Advance usage
  - Define function with gradient
    - AD + some part of numerical function
    - $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial x}$, numerical: $\frac{\partial g}{\partial x} = \frac{g(x+\Delta x)-g(x-\Delta x)}{2\Delta x}$
    - Example: Solve pole mass in TF-PWA (a iteration process)
  - Systematic of fixed parameters (fixed mass and width)
    - $-\ln L = -\ln L(\vartheta, z)$, ϑ: fit parameters, z: fixed parameters
    - Minimum condition as implicit function
    - $-\frac{\partial \ln L}{\partial \vartheta} = 0 \Rightarrow \frac{\partial \vartheta_i}{\partial z} = -\left[\frac{\partial^2 \ln L}{\partial \vartheta_i \partial \vartheta_j}\right]^{-1}\frac{\partial^2 \ln L}{\partial \vartheta_j \partial z}$

```
with config.params_trans() as pt:
    # g1 is fixed to 1
    g2_r = pt["Lmdc->piz.Sigma(1385)p_g_ls_1r"]
    g2_phi = pt["Lmdc->piz.Sigma(1385)p_g_ls_1i"]
    alpha = 2*g2_r*tf.cos(g2_phi) / (1+g2_r*g2_r)
print(alpha, pt.get_error(alpha))
```

$$\alpha_{\Sigma(1385)\pi} = \frac{|H^{\Sigma(1385)}_{0,\frac{1}{2}}|^2 - |H^{\Sigma(1385)}_{0,\frac{-1}{2}}|^2}{|H^{\Sigma(1385)}_{0,\frac{1}{2}}|^2 + |H^{\Sigma(1385)}_{0,\frac{-1}{2}}|^2} = \frac{2\Re\left(g^{\Sigma(1385)}_{1,\frac{3}{2}} \cdot \bar{g}^{\Sigma(1385)}_{2,\frac{3}{2}}\right)}{|g^{\Sigma(1385)}_{1,\frac{3}{2}}|^2 + |g^{\Sigma(1385)}_{2,\frac{3}{2}}|^2}$$

AD in Large size of data

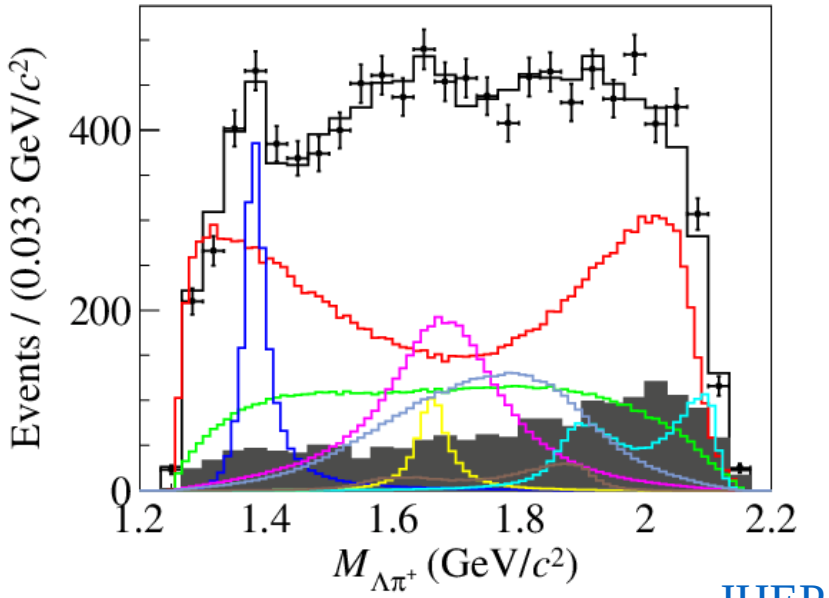
- Basic Log-Likelihood function
  - $\ln L(\vartheta) = \sum \ln \left[(1 - f_2) \frac{P_1(x;\vartheta)}{\int P_1(x;\vartheta)dx} + f_2 \frac{P_2(x;\vartheta)}{\int P_2(x;\vartheta)dx} \right]$
  - $I_i = \int P_i(x)dx \approx \sum \omega_j P_i(x_j)$
  - Required recording all $P_i(x_j)$ and intermediate values before gradients evaluations.
- Split large data into small batches (only record value of a small batch)
  - $\ln L(\vartheta) \Rightarrow \ln L(\vartheta; I_i)$
  - $\frac{\partial \ln L(\vartheta)}{\partial \vartheta} \Rightarrow \frac{\partial \ln L(\vartheta;I_i)}{\partial \vartheta} + \sum_i \frac{\partial \ln L(\vartheta;I_i)}{\partial I_i} \frac{\partial I_i}{\partial \vartheta}$
  - $\frac{\partial I_i}{\partial \vartheta} = \underbrace{\frac{\partial [\sum \omega_j P_i(x_j)]}{\partial \vartheta}}_{\text{Batch 1}} + \underbrace{\frac{\partial [\sum \omega_j P_i(x_j)]}{\partial \vartheta}}_{\text{Batch 2}} + \cdots$
- Expand the power of AD to multi GPU, even multi cluster

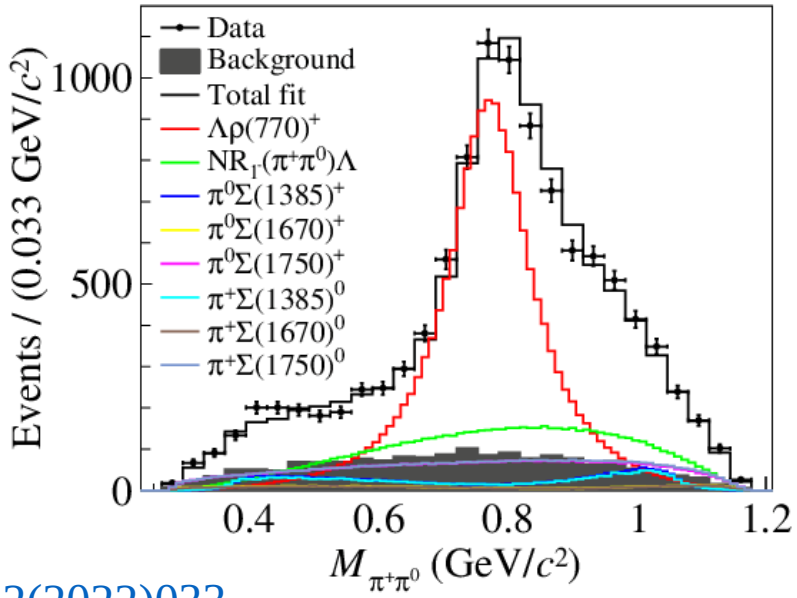
Base fit results

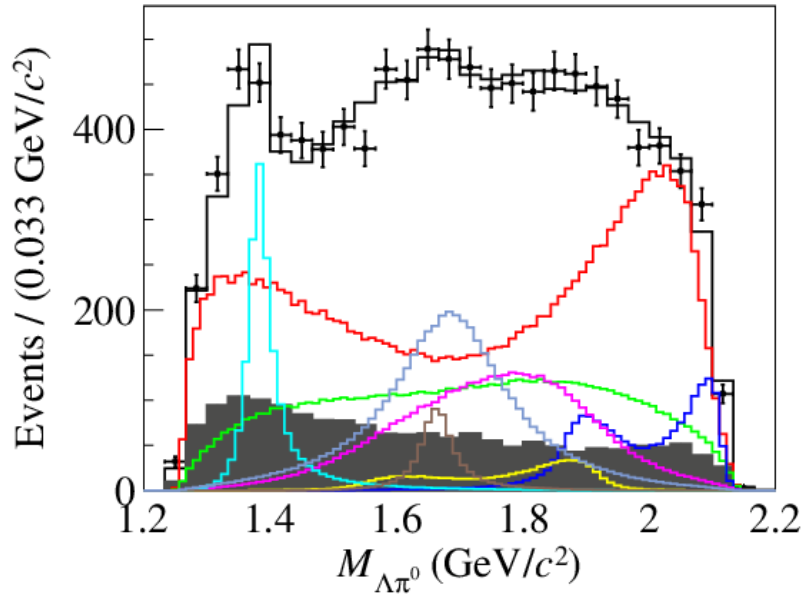
- $\Lambda_c^+ \to \Lambda(\to p\pi^-)\pi^+\pi^0$
  - Simultaneous fit
    - 7 energy points
    - Total around 10k events, 854k MC
    - 38 free parameters
  - Dominated by $\Lambda_c^+ \to \Lambda\rho$: 57.2 ± 4.2%
  - Clear peak for $\Lambda_c^+ \to \pi\Sigma(1385)$

Plot thought TF-PWA with simple config.yml
All decay chain will add automatic

```yaml
plot:
  mass:
    Sigma_star0:  # name in page 5
      display: "$M_{\\Lambda\\pi^{0}}$"
      bins: 30
      range: [1.2, 2.2]
      legend: False
```







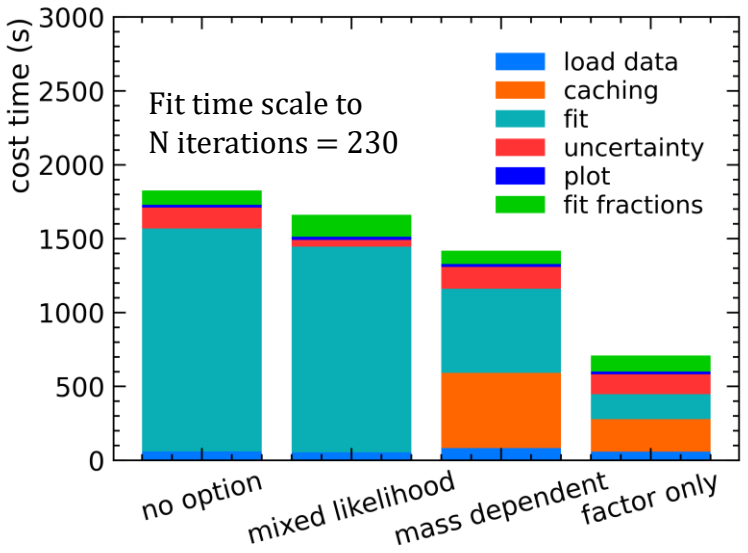
JHEP12(2022)033

Real analysis performance

Environment:
NVIDIA RTX 3080
TensorFlow 2.2
CUDA 10.1

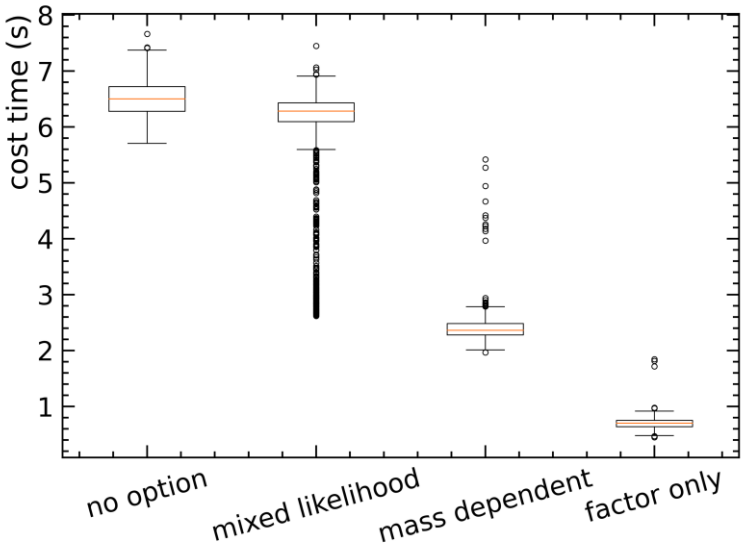
- Optimized method in Factor System page
  - Caching method (in Page 14)
    - Large time for caching
    - required more memory
    - limited to special cases
  - All the process is automatic (from config.yml to all basic results)

Total fit time



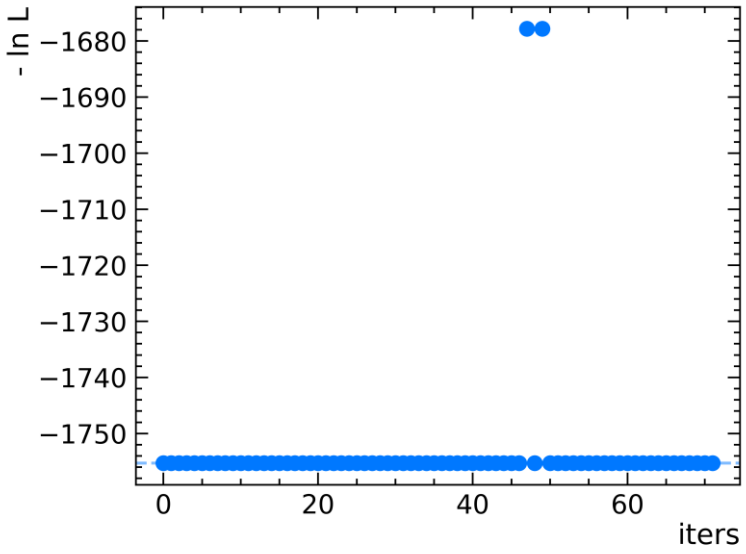
Only half of hours
for ALL the process

Time in each iterations



Caching provide 8 times speed up
for fit parts

Fit stability test



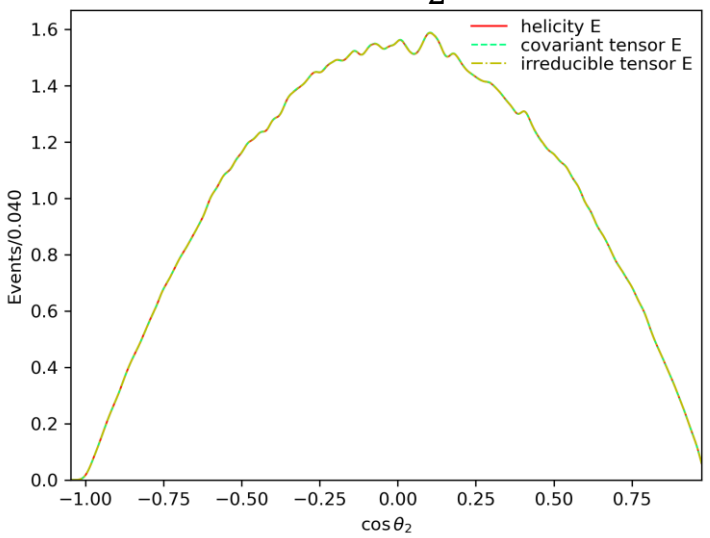
Fit stability looks well
in random initial parameters

Implement of Covariant tensor formula

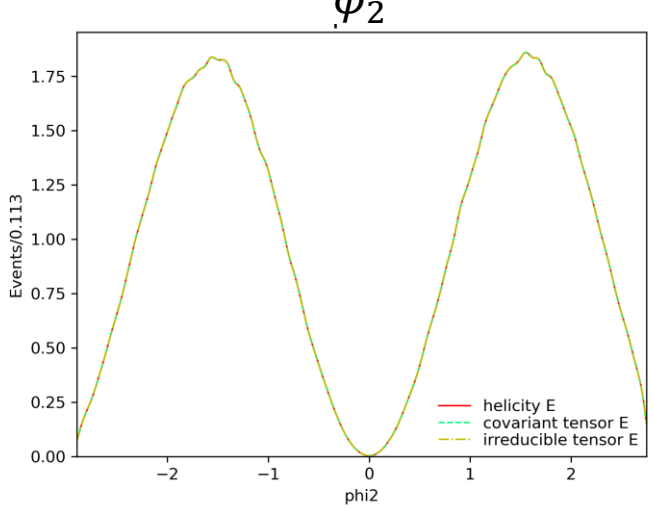
- Covariant tensor formula
  - Covariant orbital-spin scheme for any spin based on irreducible tensor arxiv:2301.01575
  - Traditional covariant tensor formula Eur. Phys. J. A 16 537-547(2003) Zou, Hugg
- Include some symbolic method for automation.
- Angular distribution consist in simple cases.
  - $0^- \to R_1(1^+)0^-, R_1(1^+) \to R_2(1^-)0^-, R_2(1^-) \to 0^-0^-$
- Preliminary implement, working in process.

Fluctuations due to MC sample size

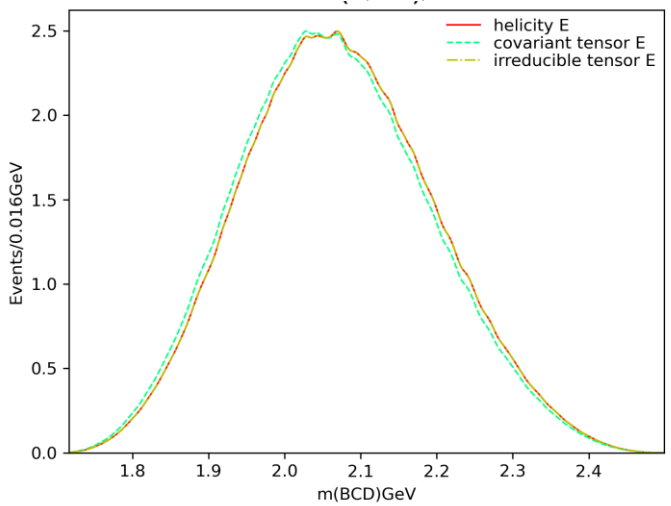
$\cos\theta_2$



ϕ_2



$M(R_1)$



Summary

- BESIII have already used many tools.
- TF-PWA
  - Convenient configuration, general proposed
  - Automatic for all procedure though numeric approach.
  - Use powerful AD in fitting and error propagation.
  - Achieve high performance
  - Developing more functions

Thank you for your attentions!

Backup

-

Formula of Resolution

- Detector: Combine effect of resolution and efficiency
- An event x was detected as y with probability
  - $p(x \to y) = \epsilon_x(x) R_x(x \to y)$
- The real probability of y: $p(y) = \int |A|^2(x) p(x \to y) dx$

$$\epsilon_y(y) = \int \epsilon_x(x) R_x(x \to y) dx, R_y(x \to y) = \frac{p(x \to y)}{\epsilon_y(y)}$$

$$p(y) = \int |A|^2(x) p(x \to y) dx = \epsilon_y(y) \int |A|^2(x) R_y(x \to y) dx$$

- $\epsilon_y(y)$ can be obtained by Phase Space MC. The distribution of reconstructed variables
- $R_y(x \to y)$ in normalize probability function that y from all possible x.
- Use $R_y(x \to y)$ to do the convolution,
  - Use phase space MC for flat x, then $R_y(x \to y)$ is the projection of $p(x, y)$ with fixed y
  - Normalized $\int R_y(x \to y) dx = 1$.
- Use MC truth to do the integration
  - $\int |A|^2(x) \int \epsilon_y(y) R_y(x \to y) dy dx = \int |A|^2(x) \epsilon_x(x) dx$.

Alignment angle

$Boost$: $B_z(\omega), Rotation$: $R_z(\phi), R_y(\theta)$

For final state $|out\rangle = |p_1\rangle \otimes |p_2\rangle \otimes |p_3\rangle$,choose a single particle state $|p_1\rangle$.
The final state define in $0 \to R, 2; R \to 1, 3$:

$$|p_1\rangle_R = B_z(\omega_1)R_z(0)R_y(\theta_1)R_z(\phi_1)|p_1\rangle = L_1|p_1\rangle$$
$$|p_1\rangle_1 = B_z(\omega_2)R_z(0)R_y(\theta_2)R_z(\phi_2)|p_1\rangle = L_2|p_1\rangle_R$$

On the other decay chain $0 \to R', 3; R' \to 1, 2$:

$$|p_1\rangle_{R'} = B_z(\omega_1')R_z(0)R_y(\theta_1')R_z(\phi_1')|p_1\rangle = L_1'|p_1\rangle$$
$$|p_1\rangle_2 = B_z(\omega_2')R_z(0)R_y(\theta_2')R_z(\phi_2')|p_1\rangle = L_2'|p_1\rangle_{R'}$$

The alignment angle is the rotation

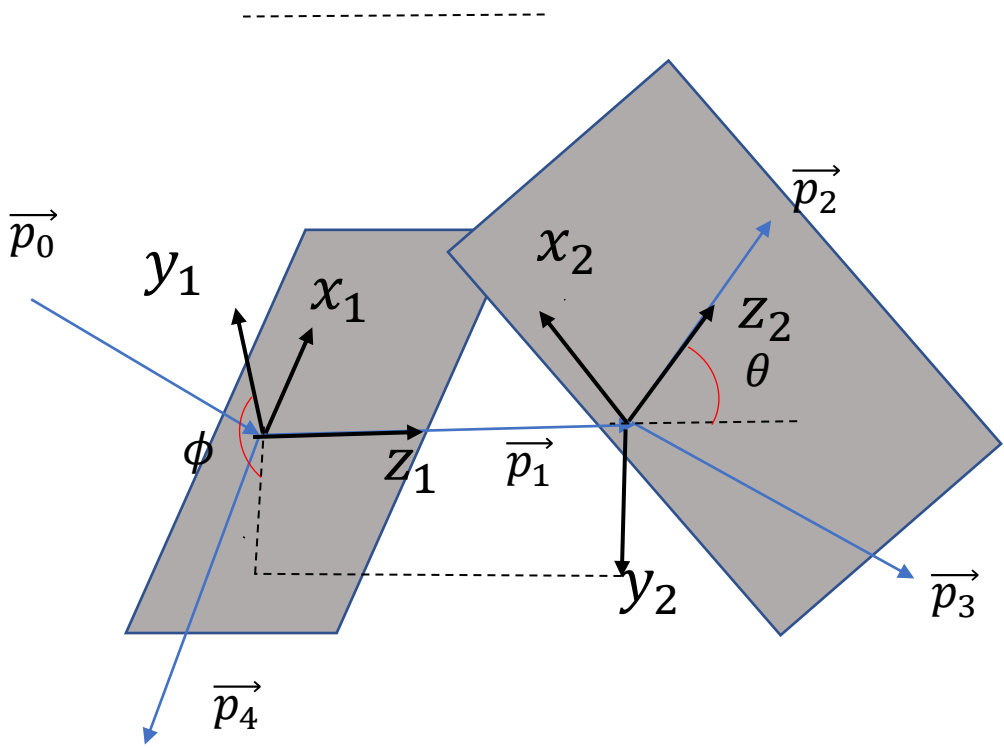
$$|p_1\rangle_2 = L_r|p_1\rangle_1 = B_z(\omega)R_z(\gamma)R_y(\beta)R_z(\alpha)|p_1\rangle_1$$

So

$$L_r = B_z(\omega)R_z(\gamma)R_y(\beta)R_z(\alpha) = L_2'L_1'L_1^{-1}L_2^{-1}$$

In general

$$L_r = L_aL_b^{-1} = \left(\prod_i L'_{n-i}\right)\left(\prod_j L_j^{-1}\right)$$



choose SU(2) representation as

$$\omega = \operatorname{arccosh}\frac{1}{\sqrt{1-\frac{v^2}{c^2}}} \qquad B_z(\omega) = \begin{pmatrix} e^{-\frac{\omega}{2}} & 0 \\ 0 & e^{\frac{\omega}{2}} \end{pmatrix}, R_z(\phi) = \begin{pmatrix} e^{-\frac{i\phi}{2}} & 0 \\ 0 & e^{\frac{i\phi}{2}} \end{pmatrix}, R_y(\theta) = \begin{pmatrix} \cos\frac{\beta}{2} & -\sin\frac{\beta}{2} \\ \sin\frac{\beta}{2} & \cos\frac{\beta}{2} \end{pmatrix}$$

$$L_{ab} = B_z(\omega)R_z(\gamma)R_y(\beta)R_z(\alpha) = \begin{pmatrix} e^{-\frac{\omega}{2}}\cos\frac{\beta}{2}e^{-\frac{i(\alpha+\gamma)}{2}} & -e^{-\frac{\omega}{2}}\sin\frac{\beta}{2}e^{\frac{i(\alpha-\gamma)}{2}} \\ e^{\frac{\omega}{2}}\sin\frac{\beta}{2}e^{-\frac{i(\alpha-\gamma)}{2}} & e^{\frac{\omega}{2}}\cos\frac{\beta}{2}e^{\frac{i(\alpha+\gamma)}{2}} \end{pmatrix}$$

$$\cos\beta = \cos^2\frac{\beta}{2} - \sin^2\frac{\beta}{2} = L_{11}L_{22} + L_{12}L_{21}, \beta \in [0,\pi]$$

$$\omega = \ln\left|\frac{L_{22}}{L_{11}}\right|$$

$$\alpha + \gamma = -2\operatorname{ang}L_{11} = 2\operatorname{ang}L_{22}, \alpha - \gamma = -2\operatorname{ang}L_{12} = -2\operatorname{ang}L_{21}$$

$$|L_{ab}| = 1 \qquad L_{ab}^{-1} = \begin{pmatrix} L_{22} & -L_{12} \\ -L_{21} & L_{11} \end{pmatrix}$$

Custom Model

Line: $R(M;a) = M + a$

```
from tf_pwa.amp import register_particle
from tf_pwa.amp import Particle

@register_particle("Line")
class LineModel(Particle):

    def init_params(self):  # define parameters
        self.a = self.add_var("a")

    def get_amp(self, *args, **kwargs):
        """ model as m + a """
        m = args[0]["m"]
        zeros = tf.zeros_like(m)
        return tf.complex(m + self.a(), zeros)
```

Use register_particle to register it,
then it can be used in config.yml

$$H_{[\lambda_R\lambda_B]}(x;\vartheta)D^{j_{A}\star}_{[\lambda_A,\lambda_R-\lambda_B]}(x)R\ (x;\vartheta)H_{[\lambda_C,\lambda_D]}(x;\vartheta)D^{j_{R}\star}_{[\lambda_R,\lambda_C-\lambda_D]}(x)$$
$$D^{j_{B}\star}_{[\lambda_B,\lambda'_B]}(x)D^{j_{C}\star}_{[\lambda_C,\lambda'_C]}(x)D^{j_{D}\star}_{[\lambda_D,\lambda'_D]}(x) \to A_{[\lambda_A,\lambda'_B,\lambda'_C,\lambda'_D]}(x;\vartheta)$$

$$\lambda_{[RB][ARB][\,][CD][RCD][BB'][CC'][DD']\to[AB'C'D']}$$

The shape is (number of events,), type is complex128

$R(x;\vartheta)$

- x: all data, (*args, **kwargs)
- ϑ: all parameters (self.a , …)

here the data, (*args, **kwargs) is passed from DecayChain.

For convenience, different data will be divided into different parts to pass to get_amp(self, *args, **kwargs)

The parameters are directly defined in the class, and the values are obtained from VarsManager.

Batch for Hessian

- $\frac{\partial \ln L(\vartheta)}{\partial \vartheta} = \frac{\partial \ln L(\vartheta,I)}{\partial \vartheta} + \frac{\partial \ln L(\vartheta,I)}{\partial I}\frac{\partial I}{\partial \vartheta}$
- $\frac{\partial^2 \ln L(\vartheta)}{\partial \vartheta \partial \vartheta} = [\frac{\partial^2 \ln L(\vartheta,I)}{\partial \vartheta \partial \vartheta} + \frac{\partial^2 \ln L(\vartheta,I)}{\partial \vartheta \partial I}\frac{\partial I}{\partial \vartheta}] + (\frac{\partial^2 \ln L(\vartheta,I)}{\partial I \partial \vartheta}\frac{\partial I}{\partial \vartheta} + \frac{\partial^2 \ln L(\vartheta,I)}{\partial I \partial I}\frac{\partial I}{\partial \vartheta}\frac{\partial I}{\partial \vartheta}) + \frac{\partial \ln L(\vartheta,I)}{\partial I}\frac{\partial^2 I}{\partial \vartheta \partial \vartheta}$
- Step1: eval $(I, \frac{\partial I}{\partial \vartheta}, \frac{\partial^2 I}{\partial \vartheta \partial \vartheta})$ in small batch
- Step2: eval $(\ln L(\vartheta'), \frac{\partial \ln L(\vartheta')}{\partial \vartheta'}, \frac{\partial \ln L(\vartheta')}{\partial \vartheta' \partial \vartheta'})$ in small batch
  - Here: $\vartheta_i' = (\vartheta_i, I), \frac{\partial \vartheta_i'}{\partial \vartheta_j} = (\delta_{ij}, \frac{\partial I}{\partial \vartheta_j})$
- Step3: $\frac{\partial^2 \ln L(\vartheta)}{\partial \vartheta_i \partial \vartheta_j} = \frac{\partial^2 \ln L(\vartheta')}{\partial \vartheta_k' \partial \vartheta_l'}\frac{\partial \vartheta_k'}{\partial \vartheta_i}\frac{\partial \vartheta_l'}{\partial \vartheta_j} + \frac{\partial^2 \ln L(\vartheta')}{\partial I}\frac{\partial^2 I}{\partial \vartheta_i \partial \vartheta_j}$

FDC-PWA

www1.ihep.ac.cn/wjx/pwa/index.html

- FDC: Feynman Diagram Calculation
  - Construct the Lagrangian and deduce Feynman rules automatically
  - Generation of all Feynman diagrams and amplitudes for a given process.
- FDC-PWA:
  - Construct effective strong interaction model
  - Generate Fortran code to calculate Partial waves amplitudes
  - Fit for coupling parameters with TensorFlow on GPU (include AD)

Many steps,
but automatic