Computing and Software for Big Science (2024) 8:3
https://doi.org/10.1007/s41781-023-00112-x

RESEARCH

KinFit: A Kinematic Fitting Package for Hadron Physics Experiments

Waleed Esmail[1] · Jana Rieger[2] · Jenny Taylor[1,2] · Malin Bohman[2] · Karin Schönning[2]

Received: 7 September 2023 / Accepted: 20 December 2023
© The Author(s) 2024

Abstract
A kinematic fitting package, `KinFit`, based on the Lagrange multiplier technique has been implemented for generic hadron physics experiments. It is particularly suitable for experiments where the interaction point is unknown, such as experiments with extended target volumes. The `KinFit` package includes vertex finding tools and fitting with kinematic constraints, such as mass hypothesis and four-momentum conservation, as well as combinations of these constraints. The new package is distributed as an open source software via GitHub. This paper presents a comprehensive description of the KinFit package and its features, as well as a benchmark study using Monte Carlo simulations of the $pp \rightarrow pK^+\Lambda \rightarrow pK^+p\pi^-$ reaction. The results show that `KinFit` improves the parameter resolution and provides an excellent basis for event selection.

Keywords Kinematic fitting · Lagrange multiplier technique · Hyperons · Hadrons

Introduction

Kinematic fitting is a powerful tool widely used in particle and nuclear physics analyses, recognized for its ability to improve the resolution of measured particle track parameters, suppress background, reconstruct undetected particles and to determine the position of vertices. Available information from measurements, such as momenta, angles and energy, combined with physics constraints, such as four-momentum conservation in a production or displaced decay vertex, or the mass of an undetected unstable particle through its decay products, are exploited. With this information, a mathematical minimization problem can be formulated and solved using, e.g., the Lagrange multiplier technique.

Kinematic fitting techniques have been available in particle physics since the 1960s [1] and gained momentum during the bubble chamber days. However, existing packages like RAVE [2] from ILD or the full decay chain fitters from Belle II [3] or PANDA [4] are often embedded in the respective experiment software and specialized for the detector setup. In the measurement of complex decays of heavy particles, it is beneficial to apply kinematic tree fits [3] to complex decay chains or jets [2, 5]. However, also track- and kinematic fitting packages that are independent of the experiment are available. One such library is ACTS [6], developed at CERN, which performs tracking finding, track fitting and vertexing. The detector geometry is in this case provided by the user. Another library is KFParticle [7] which is part of the CBM first level event trigger and can also be used for other experiments when the geometry is provided. KFParticle can for example perform a missing mass fit to reconstruct neutral particles as has been done in analysis of data from STAR [8].

In many hadron physics experiments like the HADES experiment [9] at GSI, the interaction point is not fixed to a point but can be located within a target volume that covers several centimeters of the path along the beamline. Hence, the interaction vertex position needs to be determined from the measured track parameters of the outgoing particles. In addition, weakly decaying particles such as hyperons

✉ Jenny Taylor
j.taylor@gsi.de

Waleed Esmail
w.esmail@gsi.de

Jana Rieger
jana.rieger@physics.uu.se

Malin Bohman
malin.bohman@physics.uu.se

Karin Schönning
karin.schonning@physics.uu.se

1 GSI-FFN, GSI Helmholtzzentrum fur Schwerionenforschung GmbH, Planckstraße 1, 642 91 Darmstadt, Hesse, Germany

2 Department of Physics and Astronomy, Uppsala University, Lägerhyddsvägen 1, 752 37 Uppsala, Sweden

Published online: 07 January 2024

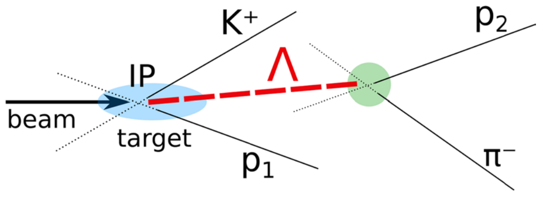


Fig. 1 Illustration of a Λ hyperon produced by a proton beam hitting a proton target. The position of the interaction point (IP, marked in blue) and the Λ decay vertex (marked in green) are determined by the point of closest approach of the p_1 and K and p_2 and π^- tracks, respectively

produce daughter particles in a secondary vertex, located a measurable distance away from the interaction point. These are crucial to reconstruct since many hyperons, e.g., Λ and Σ^0, are neutral and could otherwise escape detection. Figure 1 illustrates an example where a Λ hyperon is produced in the target volume and subsequently decays in a secondary vertex after travelling some distance. Tracks from secondary particles are more challenging to reconstruct and hence, the reconstructed track parameters have larger measurement uncertainties compared to those from primary particles. All this calls for kinematic fitting. A new package, `KinFit`, has been developed to facilitate the hyperon program at HADES [10] but is provided in an experiment-agnostic way and is therefore applicable also to other experiments. `KinFit` performs kinematic fitting of reconstructed tracks with various constraints in contrast to the previously mentioned packages which also performs track fitting. The package has the benefit that the used does not need to integrate a detector geometry in the code in order to use it but can pass already reconstructed quantities direclty to the fitting procedure. Hadron physics experiments at J-PARC [11], JLab [12] or COMPASS [13] and AMBER at CERN, may profit from this package.

In addition to kinematic fitting, tools are provided to construct a particle candidate from its daughter tracks. A functionality for running a kinematic fit for event selection on a ROOT [14] file in an automated way is included as well. The particle tracks are assumed to originate from a region free of magnetic fields, meaning they are propagated as straight tracks.

This paper is outlined as follows: the methodology including the fitting procedure and available constraint equations are described in "Methodology" section. In "Class Descriptions" section, the classes contained in the package are described and in "Performance Study" section, a benchmark study is provided for Λ hyperon reconstruction to demonstrate the performance of the fitter. In addition, a comprehensive user's guide is provided in Appendix 7.

Methodology

The goal of a kinematic fitting algorithm is to find an improved set of track parameters as close as possible to the true values, such that they fulfill a set of kinematic and/or geometric constraints. This concept can be translated quantitatively into a χ^2 minimization expressed as follows:

$$\chi^2 = (\vec{y} - \vec{\eta})^T \, V^{-1}(\vec{y}) \, (\vec{y} - \vec{\eta}) = \text{minimum}\,, \qquad (1)$$

where $\vec{y}$ is a vector of the $N \in \mathbb{N}$ measured track parameters provided by the tracking algorithm, $V(\vec{y})$ is the corresponding covariance matrix and $\vec{\eta}$ are the estimated track parameters.

In addition, the kinematic and geometric constraints are implemented in the procedure by constraint equations $\vec{f}$. These are in general represented by a set of $K \in \mathbb{N}$ continuously differentiable functions of the estimated parameters $\vec{\eta}$ and a set of $J \in \mathbb{N}_0$ unmeasured parameters combined in $\vec{\xi}$:

$$\vec{f} = \vec{f}(\vec{\eta}, \vec{\xi}) = 0.$$

The objective is to minimize the χ^2 from equation 1 subject to these constraints.

The Iterative Lagrange Multiplier Method

The method of Lagrange multipliers [15] is well-suited for addressing such problems. This technique enables the transformation of the constrained minimization into the minimization of a single *Lagrange function*, $\mathcal{L}$:

$$\mathcal{L} = (\vec{y} - \vec{\eta})^T V^{-1} (\vec{y} - \vec{\eta}) + 2\vec{\lambda}^T f(\vec{\eta}, \vec{\xi})\,, \qquad (2)$$

where the K additional variables summarized in $\vec{\lambda}$ are introduced, referred to as *Lagrange multipliers*.

Minimizing the Lagrange function (equation 2) involves finding the derivatives of $\mathcal{L}$ with respect to all unknowns $\vec{\eta}$, $\vec{\xi}$, $\vec{\lambda}$. By setting these derivatives to zero and subsequently solving for $\vec{\eta}$ and $\vec{\xi}$, the minimization can be achieved. As this problem is generally non-linear, an iterative procedure is employed, with each iteration yielding improved approximations for $\vec{\eta}$ and $\vec{\xi}$. Assuming the values of all quantities of the iteration ν have already been extracted, we want to express the quantities of the subsequent iteration $\nu + 1$ in terms of the values of iteration ν. We then proceed as follows [15]:

1. First, the following notations are introduced:

$$\vec{r} = \vec{f}^\nu + F^\nu_\eta(\vec{y} - \vec{\eta}^\nu),$$
$$S = F^\nu_\eta V (F^\nu_\eta)^T\,,$$

where F_η is a $K \times N$ Jacobian matrix ($F_\eta = D_\eta \vec{f}$), i.e., the derivative of the constraint equations with respect to $\vec{\eta}$.

2. The updated, unmeasured variables, $\vec{\xi}^{\nu+1}$, are obtained by

$$\vec{\xi}^{\nu+1} = \vec{\xi}^{\nu} - (F_\xi^T S^{-1} F_\xi)^{-1} F_\xi^T S^{-1} \vec{r},$$

where F_ξ is a $K \times J$ Jacobian matrix ($F_\xi = D_\xi \vec{f}$), i.e., the derivative of the constraint equations with respect to $\vec{\xi}$.

3. The updated Lagrange multipliers, $\vec{\lambda}^{\nu+1}$, are obtained from

$$\vec{\lambda}^{\nu+1} = S^{-1}\left(\vec{r} + F_\xi(\vec{\xi}^{\nu+1} - \vec{\xi}^{\nu})\right).$$

4. The updated measured parameters, $\vec{\eta}^{\nu+1}$, are calculated as

$$\vec{\eta}^{\nu+1} = \vec{y} - V F_\eta^T \vec{\lambda}^{\nu+1}.$$

5. Finally, the new $\mathcal{L}$ is calculated and the results compared with the previous iteration. To decide when the solution is sufficiently close to the minimum, convergence criteria are defined, see "Convergence" section.

In the end, the new covariance matrix, $V^{\nu+1}$, is calculated:

$$V^{\nu+1} = V - V[F_\eta^T S^{-1} F_\eta - (F_\eta^T S^{-1} F_\xi)(F_\xi^T S^{-1} F_\xi)^{-1}(F_\eta^T S^{-1} F_\xi)^T]V. \quad (3)$$

This ansatz assumes a quadratic minimum of the χ^2 function. The function will look different if it deviates too much from its minimum. Currently there is no guard against this, but the user should be mindful of potential deviations and assess the validity of the results accordingly.

Track Parametrization

The input objects for `KinFit` include track parameters such as momentum and angles, defined in polar coordinates, as well as the point of closest approach to the beam axis. Functions for converting from Cartesian coordinates are provided to ensure compatibility with other experiments (see the User's Guide in Appendix 7). In addition, the covariance matrix is required as input. The parameters employed here to uniquely describe a track that is parametrized as a straight line include:

- Inverse momentum $1/p$ [MeV^{-1}c].
- Polar angle θ [radians] defined from 0 to π.
- Azimuthal angle ϕ [radians] defined from $-\pi$ to π relative to the beam (z) axis.
- R [mm], the distance between the beam axis and the point of closest approach of the track to the beam axis.
- Z [mm], the z coordinate of the point of closest approach of the track to the beam axis, and
- Covariance matrix, where the diagonal entries correspond to the uncertainties in the parameters $1/p$, θ, ϕ, R and Z.

For the purely kinematic fits, only the track parameters $1/p$, θ and ϕ are required.

Constraints

`KinFit` offers a variety of constraints that can be selected for kinematic fitting.

1C: Vertex Constraint

This purely geometrical vertex constraint minimizes the distance between two tracks, ensuring they originate from the same vertex. A straight line in 3D is uniquely defined by a base vector that points from the origin of a chosen coordinate system to a coordinate on the line and the direction of the line. The components of these vectors are:

$$\begin{cases} b_x = R \cdot \cos(\phi + \pi/2), \\ b_y = R \cdot \sin(\phi + \pi/2), \\ b_z = Z, \end{cases} \quad (4)$$

and

$$\begin{cases} d_x = \sin(\theta) \cdot \cos(\phi), \\ d_y = \sin(\theta) \cdot \sin(\phi), \\ d_z = \cos(\theta). \end{cases} \quad (5)$$

In the base vector, $\pi/2$ is added to ϕ to ensure the base vector is constructed in the HADES coordinate system. A suitable constraint equation can then be formulated as follows:

$$f = (\vec{d}_1 \times \vec{d}_2) \cdot (\vec{b}_1 - \vec{b}_2) = 0. \quad (6)$$

This expression is proportional to the minimum distance between the straight lines parameterized by the respective base and direction vectors $\vec{b}_1$, $\vec{b}_2$, $\vec{d}_1$ and $\vec{d}_2$.

The vertex fit can be performed either separately or as part of a series of consecutive fitting procedures. In addition, there is the possibility to perform a mass and a geometrical vertex fit simultaneously.

4C: Four-Momentum Conservation in the Beam–Target Interaction

The 4C constraint demands that the sum of the four-momenta of the final state particles equals that of the initial beam–target system (in the following denoted with the suffix ini). Since all parameters are treated as measured parameters with uncertainties, there are four over-constraints. For N particles, the constraint equations are

$$f = \begin{cases} \sum_{i=1}^{N} p_i \sin\theta_i \cos\phi_i - p_{\mathrm{ini},x} = 0 \;\; (p_x), \\ \sum_{i=1}^{N} p_i \sin\theta_i \sin\phi_i - p_{\mathrm{ini},y} = 0 \;\; (p_y), \\ \sum_{i=1}^{N} p_i \cos\theta_i - p_{\mathrm{ini},z} = 0 \;\; (p_z), \\ \sum_{i=1}^{N} \sqrt{p_i^2 + m_i^2} - E_{\mathrm{ini}} = 0 \;\; (E). \end{cases} \tag{7}$$

3C: Four-Momentum Conservation in a Displaced Vertex

The 3C constraint utilizes four-momentum conservation at a given decay vertex, where a mother particle (M) decays weakly into N particles. This procedure relies on measured information about the primary vertex and the decay vertex, and on a mass hypothesis of the mother particle as, e.g., Λ or K_s. The angles of the mother particle, θ_M and ϕ_M, are determined by the direction of the vector pointing from the primary to the decay vertex and are therefore treated as measured parameters, whereas the momentum p_M is unmeasured and hence obtained from the fit. This results in three over-constraints (3C). The initial value is estimated from energy conservation of the decay products using

$$p_M = \sqrt{\left(\sum_i \sqrt{p_i^2 + m_i^2}\right)^2 - m_M^2} \,. \tag{8}$$

In this expression, the subscript M represents the mother particle, while i refers to the decay products. The user must provide the mass of the mother particle as a mass hypothesis. The uncertainties in the vertex positions must estimated in the previous step and are propagated to the uncertainties in the parameters. Since the momentum of the mother particle p_M is an unmeasured quantity, its uncertainties are not known a priori but are, as the momentum itself, obtained from the fit. The constraint equations ensure four-momentum conservation in the decay of the mother:

$$f = \begin{cases} \sum_{i=1}^{N} p_i \sin\theta_i \cos\phi_i - p_M \sin\theta_M \cos\phi_M = 0 \;\; (p_x), \\ \sum_{i=1}^{N} p_i \sin\theta_i \sin\phi_i - p_M \sin\theta_M \sin\varphi_M = 0 \;\; (p_y), \\ \sum_{i=1}^{N} p_i \cos\theta_i - p_M \cos\theta_M = 0 \;\; (p_z), \\ \sum_{i=1}^{N} \sqrt{p_i^2 + m_i^2} - \sqrt{p_M^2 + m_M^2} = 0 \;\; (E). \end{cases} \tag{9}$$

1C: Four-Momentum Conservation with a Missing Particle

Four-momentum conservation at the interaction point enables the reconstruction of one undetected or missing (indicated by the suffix miss) particle with a mass hypothesis m_{miss}. I all other inital and final state particles are known or measured, the four-momentum conservation results in the following constraint equations:

$$f = \begin{cases} \sum_{i=1}^{N} p_i \sin\theta_i \cos\phi_i + p_{\mathrm{miss},x} - p_{\mathrm{ini},x} = 0 \;\; (p_x), \\ \sum_{i=1}^{N} p_i \sin\theta_i \sin\phi_i + p_{\mathrm{miss},y} - p_{\mathrm{ini},y} = 0 \;\; (p_y), \\ \sum_{i=1}^{N} p_i \cos\theta_i + p_{\mathrm{miss},z} - p_{\mathrm{ini},z} = 0 \;\; (p_z), \\ \sum_{i=1}^{N} \sqrt{p_i^2 + m_i^2} + \sqrt{p_{\mathrm{miss}}^2 + m_{\mathrm{miss}}^2} - E_{\mathrm{ini}} \\ \quad = 0 \;\; (E). \end{cases} \tag{10}$$

Here, one has measured particles with indices i, a missing particle with mass m_{miss} and the initial beam–target four vector p^{μ}_{ini}. The mass, m_{miss} needs to be provided by the user as a hypothesis as well as p^{μ}_{ini}.

The four-momentum of the missing particle can be extracted after the fit. Since there are three unmeasured variables, i.e., the three-momentum of the missing particle, only one over-constraint (1C) remains.

1C: Missing Mass Constraint

The missing mass constraint is suitable in the case when there is one undetected final state particle whose momentum is not of interest. Instead, a missing mass constraint can be formulated from the mass hypothesis of the missing particle:

$$f = \sqrt{\left(E_{\mathrm{ini}} - \sum_{i=1}^{N} E_i\right)^2 - \left(\vec{p}_{\mathrm{ini}} - \sum_{i=1}^{N} \vec{p}_i\right)^2} - m_{\mathrm{miss}} = 0. \tag{11}$$

1C: Invariant Mass Constraint

The mass constraint can be used to constrain a set of particles to originate from a common mother particle, whose mass is known. The invariant mass of this set of particles is then constrained to the mass of the hypothetical mother particle:

$$f = \sqrt{\left(\sum_{i=1}^{N} E_i\right)^2 - \left(\sum_{i=1}^{N} \vec{p}_i\right)^2} - m_{\mathrm{mother}} = 0 \,. \tag{12}$$

Convergence

The iterative fitting procedure is terminated when convergence is achieved. There are three different convergence criteria: the difference in χ^2 of the Lagrange function between consecutive iterations, the Euclidean norm of the sum of all constraint equations and the Euclidean norm of the difference of all track parameters, normalized to the initial measurement, between two consecutive iterations. The default value for the convergence criteria is 10^{-4} but can be customized. If convergence is not reached before a specified number of iterations (default 20), the fitting procedure is exited.

Goodness of Fit

The performance of the fit is assessed by the final χ^2_{final} of the fit and the so-called pull distributions for all fitted variables. The value of χ^2_{final} should be small, on the order of the number of over-constraints, N_C. For an ensemble of events, N_C defines the shape of the χ^2_{final} distribution. There is a one-to-one relation between χ^2_{final} for a given N_C and the corresponding probability, as defined by the probability density function [16]. Ideally, the probability distribution should be uniform if the correct particle hypotheses have been used in the fit and if the covariance matrices accurately describe the measurement precision. However, if the particle hypotheses are incorrect, then a peak towards zero probability should be discernible. If elements of the covariance matrix are over- or underestimated, the distribution will decrease or increase towards larger values of probability.

The pull distributions are defined as:

$$pull = \frac{\eta - y}{\sqrt{\sigma^2(y) - \sigma^2(\eta)}}, \tag{13}$$

for each variable, where η/y are the fitted/measured variables, respectively, and $\sigma(\eta)/\sigma(y)$ are the corresponding uncertainties. The pull distribution should follow a normal distribution with a mean value of 0 and a standard deviation of 1.

Class Descriptions

The `KinFit` package is written in C++ and based on ROOT [14] (version 6) and uses CMake [17] (version 3.0 or newer) for the installation. It is available online at:

```
https://github.com/KinFit/
    KinFit.git
```

Documentation about the usage of the provided tools can be found in the README of the git repository and in the User's Guide in Appendix 7.

The properties of the particle candidates needed as input are the particle's track parameters, mass, and covariance matrix. Functions are provided to transform Cartesian track parameters to the R, Z track parameters.

KFitParticle

The track parameters of particle candidates are organized in `KFitParticle` objects that provide all information needed by the fitter class. It inherits from the ROOT class `TLorentzVector`. For each candidate, the values for the track parameters described in "Track Parametrization" section have to be set. Optionally, an arbitrary particle ID and track ID can be chosen by the user for later reference.

KFitDecayCandFinder

`KFitDecayCandFinder` calculates properties of an unmeasured candidate that decays in a displaced vertex to be used later in the fit. The angles, θ and ϕ are calculated from the line segment connecting the primary to the displaced decay vertex. The uncertainties for these angles are propagated from the uncertainties in the vertex positions in X, Y and Z using the matrix formalism for error propagation. The user needs to provide the uncertainties for the vertex positions.

KFitVertexFinder

`KFitVertexFinder` finds the vertex by calculating the point of closest approach between at least two tracks by a matrix formalism. For more than two tracks, the vertex is taken as the center of gravity, i.e., the point that is simultaneously closest to all tracks.. This is calculated from a least square method. The vertex finding code is based on a procedure in HYDRA [1], the HADES software.

KinFitter

`KinFitter` is the main class containing the fitting functions. It can perform a vertex, 4C, 3C, missing particle, missing mass, mass and a combined vertex+mass fit (see "Constraints" section). The constraint equations and Jacobi matrices are implemented here. This is where the iterative fitting procedure is carried out. The maximum number of iterations or convergence criteria can be changed to custom values.

[1] https://subversion.gsi.de/hades/hydra2/

KFitAnalyzer

`KFitAnalyzer` is a user interface class. It contains user settings and performs the event loop with a fit of choice. The input particles need to be stored as `KFitParticles` in a `TClonesArray`. For each event, particles are selected based on their particle ID (PID), which is also provided to the `KFitAnalyzer`. A `KFitDecayBuilder` object is created which takes the selected particles and the desired constraint as input. In the end, the fitted particles are retrieved from the `KFitDecayBuilder` and stored in an output file together with the fit probability. The `KFitAnalyzer` currently performs all fits except the 3C fit.

KFitDecayBuilder

This class is responsible for building all possible combinations of the particles within one event. Each combination is passed to the `KinFitter` which performs the selected fit. The `KFitDecayBuilder` selects the combination with the best fit probability.

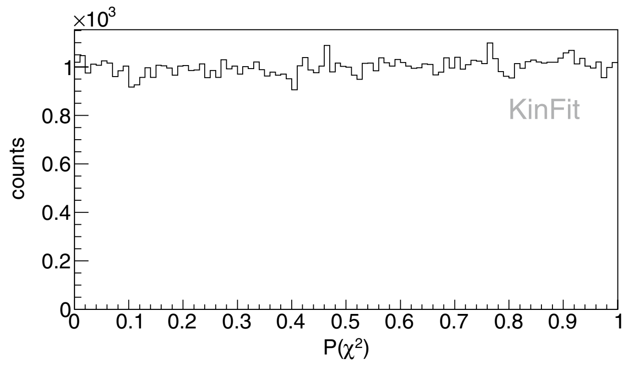
Performance Study

The performance of `KinFit` is benchmarked using Monte Carlo (MC) simulations. These represent an ideal scenario where a finite detector resolution is added by smearing, and thus known, whereas in real data, it might be difficult to estimate the covariance matrix exactly. This approach allows for a focused investigation of the quality of the `KinFit` tools. The reaction investigated is $pp \to pK^+\Lambda$, $\Lambda \to p\pi^-$, as illustrated in Fig. 1. This reaction provides an opportunity to examine all tools contained in the package. All fits performed in this section use the default settings of `KinFitter`, described in "Convergence" section.

The event generator Pluto [18] was used to generate 100 000 events of the $pp \to pK^+\Lambda$ $\Lambda \to p\pi^-$ reaction at a beam kinetic energy of T=4.5 GeV. In our simulations, the produced particles and their decay products are distributed isotropically across the available phase space. The primary vertex is generated at the origin, i.e., (0,0,0). A full 4π acceptance is assumed and no material effects are included. The Λ hyperons decay at a displaced decay vertex according to the mean Λ hyperon life time $c_\tau(\Lambda) = 7.98$ cm [16]. The track parameters of the final state particles are smeared according to a Gaussian distribution, simulating uncertainties from, e.g., detector resolution in a controlled way. The uncertainty for each track parameter is listed in Table 1. Since the results presented here do not depend on a specific experiment, the uncertainties are set to be constant, except the momentum uncertainty, which is momentum dependent.

Table 1 Gaussian standard deviations of the track parameters in the MC sample after smearing

Track parameter	$1/p$	θ	ϕ	R	Z
σ	$0.025 \cdot 1/p$	0.0009 rad	0.0009 rad	0.5 mm	1 mm



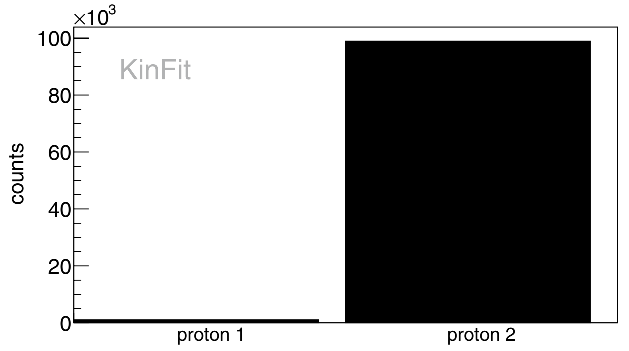


Fig. 2 Probability distribution of a Λ hyperon mass fit in the reaction $pp \to pK^+\Lambda$ $\Lambda \to p\pi^-$ using `KFitAnalyzer` (top) and the proton selected to originate from the Λ hyperon decay (bottom)

Mass Fit Using KFitAnalyzer

In each event, there are four measured particles: a proton (p_1) and a kaon from the interaction point and a pion and a proton (p_2) from the Λ hyperon decay. This means there are two possibilities to combine a pion with a proton, either $p_1\pi^-$ and $p_2\pi^-$, of which the latter is the correct Λ decay candidate. The `KFitAnalyzer` is used to find the pion–proton combination that comes from the Λ hyperon decay through the application of a mass fit.

The pion–proton pair giving the largest mass fit probability (see Fig. 2) are selected as Λ daughters. In 99 % of the events, the correct proton p_2 is chosen. The probability distribution is uniform, as expected.

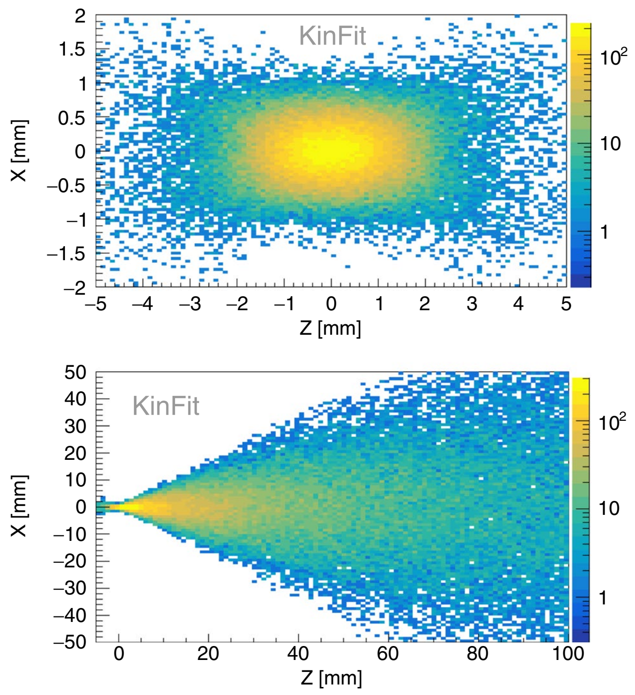


Fig. 3 Reconstructed production vertex from proton and K^+ (top) and location of reconstructed decay vertices of the Λ hyperon from the secondary proton and π^- (bottom) as reconstructed by the `KFitVertexFinder`. The positions are projected onto the x–z plane

Reconstruction of Λ Hyperon from Vertex Positions and a 3C Fit

The Λ hyperon reconstruction procedure includes three steps:

1. **Vertex finding** As a first step the position of the primary vertex, where the Λ hyperon was produced is determined from the point of closest approach between the other tracks coming from the production vertex. In this case, these are the proton (p_1) and kaon tracks. This task is performed by `KFitVertexFinder`. In a similar way, the Λ hyperon decay vertex is obtained from the point of closest approach between the proton (p_2) and pion tracks originating from the Λ hyperon decay. The vertex positions are shown in Fig. 3. All primary particles are produced at the origin which means that the distribution in the top panel reflects the finite resolution of the vertex estimation. The dominating effect in the Λ decay vertex distribution is the distance travelled by Λ before decaying.
2. **Reconstruction of the neutral candidate** The direction of the Λ candidate is given by the vector pointing from the primary to the decay vertex. The magnitude of the Λ hyperon momentum is estimated by the of the sum of the three-momenta of its daughter particles, see equation . The `KFitDecayCandFinder` carries out this task and calculates the uncertainties of the track parameters of the Λ candidate. The latter requires information on the vertex uncertainties which were estimated by Gaussian fits to the vertex resolution in x, y and z for each vertex.
3. **3C fit** A kinematic fit ensuring four-momentum conservation at the Λ hyperon decay vertex is the final step and is executed by `KinFitter`. This improves the resolution of the Λ hyperon track parameters significantly (Fig. 4) and can be used to reject false combinations of protons in the vertices.

Only the correct combination of particles was used in this example, i.e., correct assignment of the protons to the vertices. The pull distributions for the pion track are shown as an example in Fig. 5. These nearly follow a normal distribution, as expected. Figure 6 shows the probability distribution of the 3C fit. The probability deviates slightly from a uniform distribution and there are some events for which the fit converged, but gives a low probability. This effect and the slightly larger deviation of

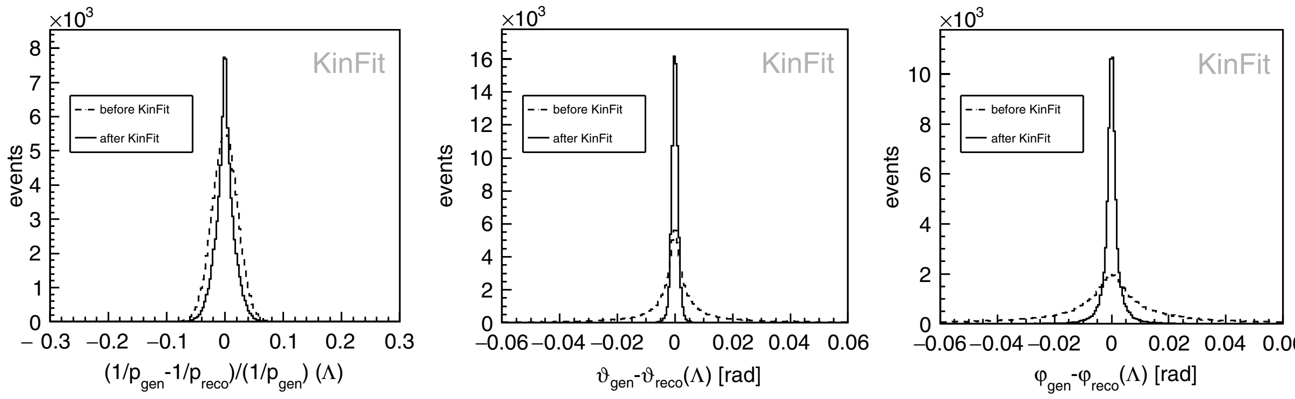


Fig. 4 Resolution of the Λ hyperon track parameters before and after the 3C fit

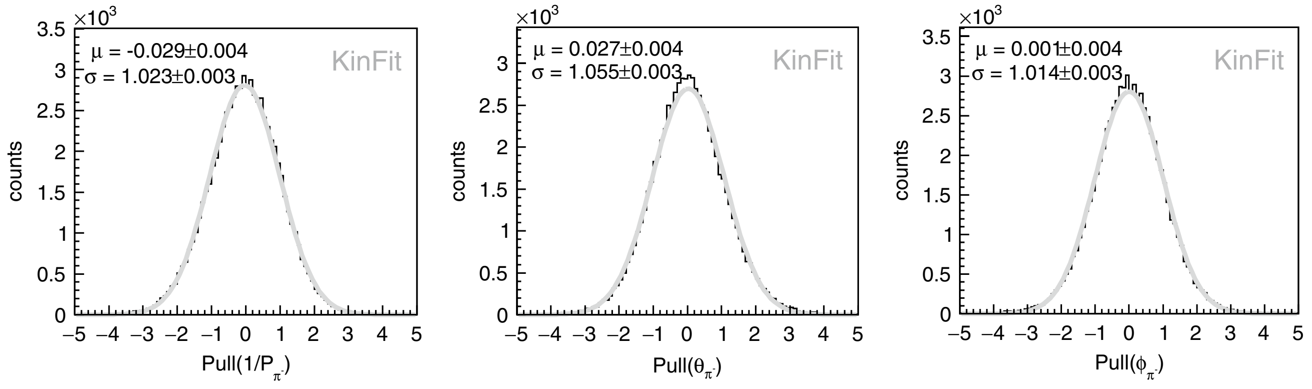


Fig. 5 Pull distributions for the π^- track parameters after the 3C fit with respective mean and standard deviation

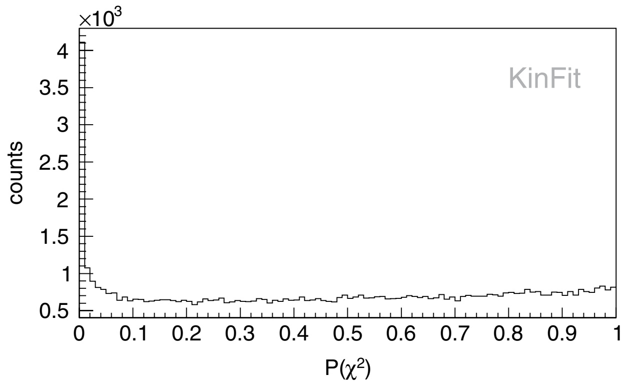


Fig. 6 Probability distribution from the 3C fit

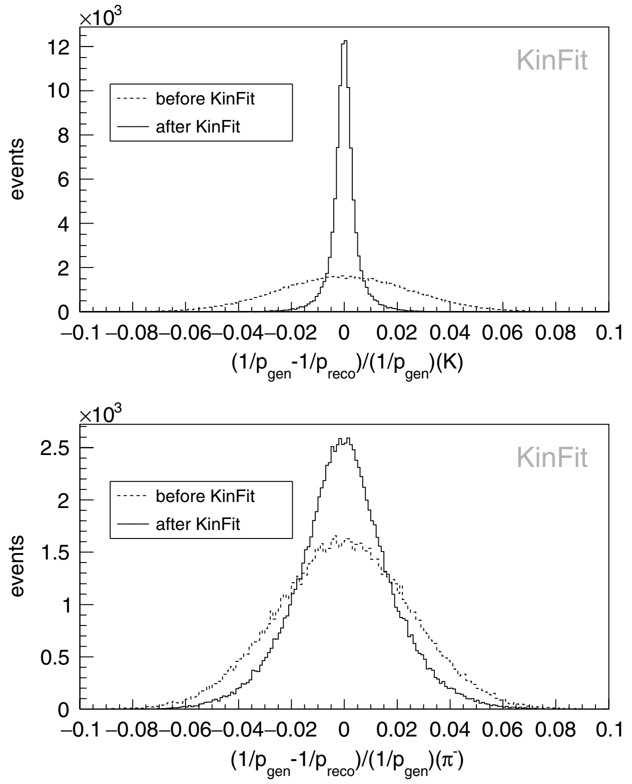


Fig. 7 Momentum resolution for the K^+ (top) and π^- (bottom) before and after the 4C fit

the pull distributions from a normal distribution are only seen in the 3C fit and originate from the estimation of the vertex resolutions, which are not exactly described by a Gaussian.

4C Fit of the Reaction $pp \rightarrow pK^+\Lambda$

The 4C fit is used to constrain the four-momenta of all final state particles to that of the initial beam–target system. In this example, the final state particles are a proton and a kaon from the primary vertex and a proton and a pion from the Λ hyperon decay vertex. Figure 7 shows the momentum resolution for the kaon and the pion before and after the fit. The maximum improvement in resolution is achieved for the kaon momentum, with $\frac{\sigma_{\text{pre-fit}}-\sigma_{\text{post-fit}}}{\sigma_{\text{pre-fit}}} = 89\,\%$. This improvement is more substantial for the kaon momentum resolution compared to the other particles. This is due to the larger uncertainty of the kaon associated with its larger total momentum. The probability distribution is uniform and the pull distributions follow a normal distribution, as shown in Fig. 8.

Missing Particle Fit of K^+ in the Reaction $pp \rightarrow pK^+\Lambda$.

In this scenario, it is assumed that the kaon is not detected. The missing particle fit is employed to constrain the four-momenta of the detected particles along with the undetected particle to match the beam–target system, as well as to estimate the momentum of the missing particle. Figure 9 presents the probability distribution of the fit and compares the kaon momentum resolution from the initial guess and after

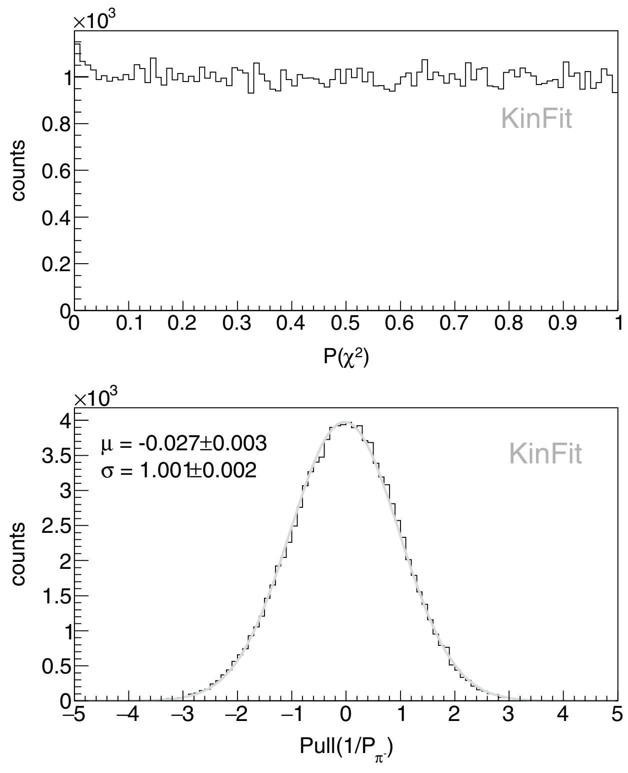


Fig. 8 4C fit. Probability distribution (top) and example pull distribution of the π^- momentum with mean and standard deviation (bottom)

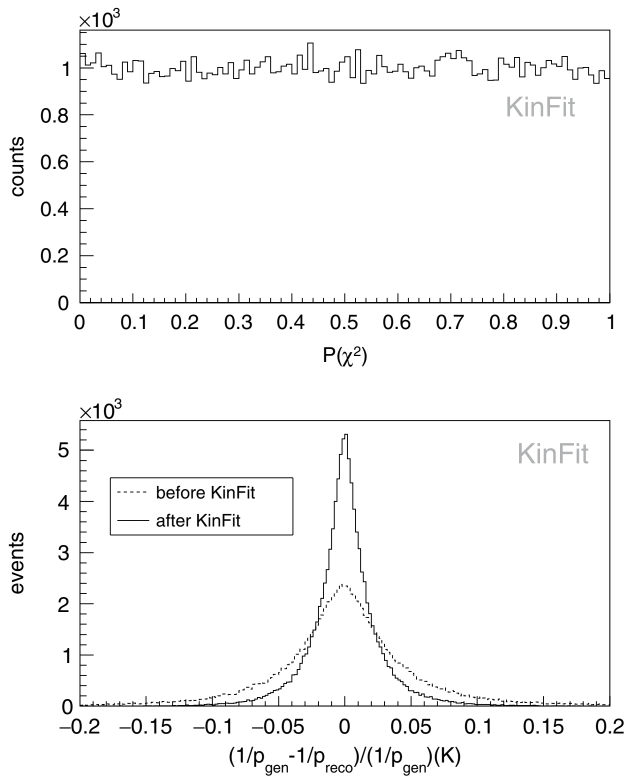


Fig. 9 Missing particle fit. Probability distribution (top) and resolution of the estimated and fitted K^+ momentum (bottom)

the fit. The initial guess for the kaon momentum is calculated from three-momentum conservation in the interaction point. The momentum resolution before the fit is worse than in the example in "4C Fit of the Reaction pp → pK+Λ" section, where the momentum was measured directly. A noticeable improvement in the momentum resolution can be observed after the fit. However, the resolution is still worse than in the case of the 4C fit, as the missing particle fit has fewer over-constraints. The resolutions of the other track parameters and the pull distributions appear quite similar to those of the 4C fit.

Vertex Fit

The vertex fit aims to constrain the track parameters of final state particles that originate from the same single point in space. To perform the vertex fit, at least two outgoing particles from the same vertex must be measured. In this example, these particles are the kaon and proton (p_1) produced at the beam–target interaction point. Figure 10 illustrates the resolution of the estimated R-parameter, defined in "Track Parametrization" section, for the proton. The probability distribution for this vertex fit, along with an example pull distribution of the R-parameter of p_1, is depicted in Fig. 11. As anticipated, the probability is uniformly distributed across

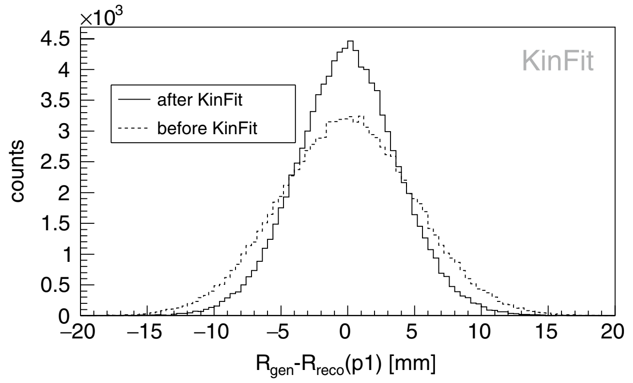


Fig. 10 Vertex fit. Resolution of the R-parameter of the primary proton (p_1) before and after the fit

its range, while the pull follows a normal distribution. The `KinFitAnalyzer` was used to choose the proton track that has the largest probability to come from the same vertex as the kaon track. The correct proton (p_1) is chosen in 85 % of the events, which is lower than the fraction of correctly identified combinations by the mass fit presented in "Mass Fit using KFitAnalyzer" section. This shows that the mass constraint is more powerful than the vertex fit.

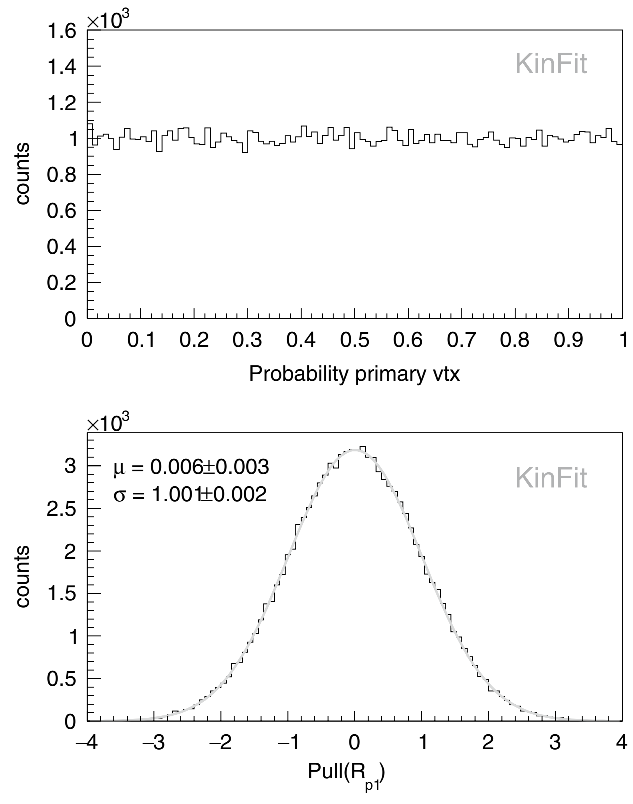


Fig. 11 Vertex fit. Probability distribution (top) and example pull distribution of the R-parameter for the primary proton ($p1$) (bottom)

Table 2 Runtime per event for different parts of `KinFit`

	Time / event [μs]
Initialization	2.39 ± 0.08
Vertex Finding	3.04 ± 0.07
Mother Candidate Finding	1.92 ± 0.06
Fitting (1 iteration)	16.48 ± 0.20
Fitting (2 iterations)	20.65 ± 0.26
Fitting (3 iterations)	24.79 ± 0.32
Fitting (4 iteration)	28.93 ± 0.33
Fitting (5 iterations)	33.06 ± 0.48
Fitting (6 iterations)	37.12 ± 0.37

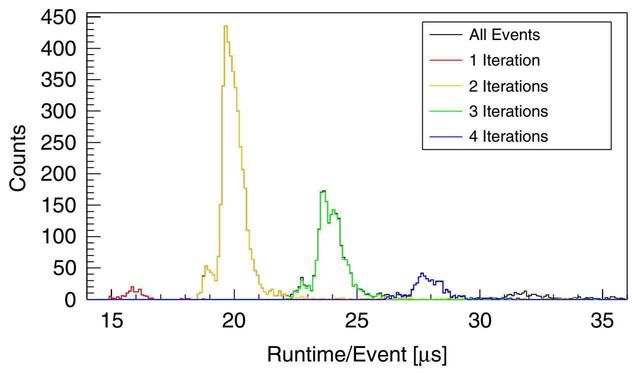


Fig. 12 The runtime for all converged events after the 3C fit. The peaks correspond to different number of iterations before the fit converges and the first four iterations are highlighted with different colors

Runtime Performance

`KinFit` currently runs sequentially on CPU architectures. The runtime performance of various components of `KinFit` was evaluated on a laptop with the following hardware specifications:

- Processor: Intel Core i7-1185G7, 3.00 GHz
- Memory: 32 GB RAM

The results are displayed in Table 2. The average runtime per 1000 events for the 3C constraint (see "3C: Four-momentum conservation in a displaced vertex" section) was calculated for the cases in which the fit converged. There are two tracks per event used in each fit. An overhead on the order of a microsecond for initialization, vertex finding and mother candidate finding each. The runtime for the iterative fitting depends on the number of iterations and the time for each event is shown in Fig. 12. Peaks are visible that correspond to the number of iterations. There is roughly a linear increase in runtime with an increase of about 4 μs per additional iteration.

Conclusions

The `KinFit` package is a versatile kinematic fitting package that has been developed to provide tools for reconstruction of vertices, momenta and masses of particles in a generic hadron physics experiment. In particular, it is capable of fitting an unknown production vertex from an extended beam–target interaction volume, and of combining kinematic and geometric constraints.

The tools provided by `KinFit` have been evaluated using toy MC simulations, showing significant improvements in track parameter resolutions and the capability to accurately select the correct particle hypotheses and combinations. It's essential to note that while our results are promising, a comparison with other kinematic fitting packages will further validate the efficacy of `KinFit` in a broader context.

While the focus has been on providing suitable tools for hadronic interactions and specifically hyperon decays, the package can be applied for other types of reactions as well. Though `KinFit` uses the same track parametrization as HADES, but it is generally considered experiment-independent.

The minimal overhead introduced by adding KinFit to an analysis suggests that the package is suitable for running on a local machine.

Outlook

Although the majority of particle physics analyses are based on C++, the demand for similar Python tools is currently growing due to the rapid expansion of the Python ecosystem. Therefore, the KinFit package is planned to be integrated into the SciKit-HEP project [19], which is a community-driven Python ecosystem for data analysis.

Appendix

A: User's Guide

The source code is available online in the KinFit git repository. It can be downloaded via

```
git clone https://github.com/KinFit/KinFit.git
```

and installed using CMake. A ROOT 6 installation is required.

There are two options how the tools provided by this package may be applied by the user. One of them is to use the provided user functions through KFitAnalyzer. These make it straight-forward to apply the fitter using basic constraints. However since the type of analysis where kinematic fitting can yield large improvements often requires a careful and customized event selection, the provided classes can also be applied directly by the user, which makes a fine tuning of the fitting process possible but requires a deeper understanding of the procedure.

A1: Automated Application of a Kinematic Fit

In order to use KinFit in an automated way, the user needs to provide a root file with a TTree called "data" that contains a TClonesArray "KFitParticle" of KFitParticles as input. The macro analysis_user.C illustrates how to set up the KFitAnalyzer.

```
#include "KFitAnalyzer.h"

Int_t analysis_user(TString infile, TString outfile, Int_t evts){

    KFitAnalyzer RootAnalyzer(infile, outfile, evts);
    std::vector<int> pids;
    pids.push_back(14); pids.push_back(9);
    Double_t mass = 1.11568;

    RootAnalyzer.doFitterTask("Mass", pids, mass);

    return 0;
}
```

A KFitAnalyzer object is created. It takes the input file(s), the output file name and the number of events to be processed as input. The only command that needs to be executed to start the analysis is doFitterTask. Here the type of fit, the PIDs of the particles to be fit and additional information that is required for the specific fit is requested. In this example, an invariant mass fit of a proton and a pion is performed, assuming that they originate from a Λ decay. After the analysis procedure the fitted tracks of the best particle combination and the fit probability are written to the output file. This requires that the kinematic fit converges and that that combination yield the highest fit probability of the event.

Other available fits are

```
doFitterTask("4C", pids, -1, ppSystem);
doFitterTask("MassVtx", pids, mLambda);  doFitterTask("MissingMass", pids,
    mass, ppSystem);
doFitterTask("MissingParticle", pids, mass, ppSystem);
doFitterTask("Vertex", pids, -1);
```

A2: Using the Individual Classes

In an event-based analysis, the kinematic fitting tools are typically applied as one of the first analysis steps inside the ”event loop”, i.e., while iterating through all events, selecting the most suitable particle candidates from each.

Creating a KFitParticle

After a set of particle candidates are selected whose track parameters are to be fit, a KFitParticle object is created for each candidate. In addition, the covariance matrix has to be assigned to each candidate. This is done either by calling the constructor KFitParticle(TLorentzVector cand, double R, double Z) or KFitParticle(TLorentzVector cand, double X, double Y, double Z) and setting the covariance matrix using the setCovariance() function directly, or by using a suitable FillData function, useful when the covariance was estimated and is not known for each particle individually. An example of how a FillData function can look like to set the attributes of the KFitParticle is shown below.

```
void FillData(TLorentzVector *cand, KFitParticle *outcand, double R,
   double Z, double arr[], double mass, int trackID, int pid)

    /** Cov(5,5)Covariance matrix of the KFitParticle
    * Diagonal entries correspond to the covariances
    * in the parameters in the following order
    *
    * ----------------------------
    * | 1/p                      |
    * |     theta                |
    * |            phi           |
    * |                   R      |
    * |                        Z |
    * ----------------------------
    *
    * Off diagonal elements corresponds to the
    * correlations between the parameters
    * arr[] can be adjusted acordingly to set the
    * off diagonal elements
    */

    TMatrixD cov(5, 5);
    cov(0, 0) = std::pow(arr[0], 2);
    cov(1, 1) = std::pow(arr[1], 2);
    cov(2, 2) = std::pow(arr[2], 2);
    cov(3, 3) = std::pow(arr[3], 2);
    cov(4, 4) = std::pow(arr[4], 2);


    outcand->SetXYZM(
        cand->P() * std::sin(cand->Theta()) * std::cos(cand->Phi()),
        cand->P() * std::sin(cand->Theta()) * std::sin(cand->Phi()),
        cand->P() * std::cos(cand->Theta()), mass);
    outcand->setThetaRad(cand->Theta());
    outcand->setPhiRad(cand->Phi());
    // outcand->setThetaDeg(cand->Theta()); // Depending on the unit
       of the angles from cand the functions setThetaRad(double val)
       and setThetaDeg(double val) can be used iterchangably
    // outcand->setPhiDeg(cand->Phi()); // Depending on the unit of
       the angles from cand the functions setPhiRad(double val) and
       setPhiDeg(double val) can be used iterchangably
    outcand->setR(R);
    outcand->setZ(Z);
    outcand->setCovariance(cov);

    //optional
    outcand->setTrackId(trackID);
    outcand->setPid(pid);
```

In this example, R and Z are given as an input by the user. Alternatively the creation point of the particle candidate can be given in cartesian coordinates which are internally converted to R and Z. `SetXYZM()` sets the attributes of the `TLorentzVector` that the `KFitParticle` inherits from whereas `setThetaRad()`, `setPhiRad()`, `setR()`, `setZ()` and `setCovariance()` operate directly on the `KFitParticle` object. The covariance matrix needs to be provided by the user as well as a mass hypothesis. A PID and track-ID can be set optionally.

Handling the `KinFitter`

To use fitting functions, the `KFitParticle` objects that will be used need to be placed in a vector of the type `std::vector` from the C++ STD library. This vector needs to be passed to the constructor of `KinFitter`. How this is done is illustrated in the examples below. For most fit options, an arbitrary number of particles can be added. However, for the vertex fit, exactly two particles need to be added. Following this, the user must choose one of the constraints, either; the 3C, the 4C, mass, missing mass, the vertex or the missing particle. Adjusting the number of iterations as well as the convergence criteria is possible but optional; if not set, the default criteria are applied. Then the `fit()` function is called. This performs the actual fitting and returns *true* if the fit converged. Some information can be obtained independently of the individual fit function that is used, e.g., `getChi2()` returns the χ^2 of the fit and `getProb()` returns the corresponding probability. `getPull(int val)` returns the pull of variable v for a daughter particle p, $val = 5 \cdot d + v$. The function `isConverged()` returns a `boolean`; *true* if the fit has converged and *false* otherwise, this can be used for event or sample selections. The initialization of `KinFitter` settings that can be used before calling `fit()` are summarized below.

```
KinFitter(const std::vector<KFitParticle> &cands);
setNumberOfIterations(int val); // default 20
setConvergenceCriteria(double val, double val2, double val3);
// default 1e-4
setVerbosity(int val); //between 0 and 2, default 0
```

After the fitting procedure called by `fit()`, the fit result can be assessed by the following functions. Updated track parameters are obtained by the `KFitParticle` objects returned by the first function

```
KFitParticle getDaughter(int val); // Fitted daughter particle #val
double getChi2(); // Returns chi2
double getProb(); // Returns fit probability
double getPull(int val); // Returns pull of variable val
//* The pulls are returned in the following order:
* val = i + 0 : 1/ p of particle i
* val = i + 1 : theta of particle i
* val = i + 2 : phi of particle i
* val = i + 3 : R of particle i
* val = i + 4 : Z of particle i
* Where the particles appear in the same order they
* were entered into the fit
*/
bool isConverged(); // Returns true if fit converged
int getIteration(); // Returns number of iterations until convergence
    or maximal number of iterations
```

An example of how to perform the vertex fit is performed is shown below.

```
#include "KinFitter.h"

KinFitter fitter(cand_vector);
fitter.addVertexConstraint(); // Choose vertex constraint
fitter.fit();
```

To perform the missing particle fit, the mass of the missing particle and a TLorentzVector corresponding to the initial beam–target system must be given as input to the fitter in addition to a mass hypothesis. An example of how the TLorentzVector can be constructed is in the following way:

```
TLorentzVector ppSystem(p1,p2,p3,E);
```

where the first three entries correspond to the momentum in each Cartesian direction and the last entry is the total energy of the system. After the fit has been performed, the missing daughter can be retrieved as a TLorentzVector

```
#include "TLorentzVector.h"
#include "KinFitter.h"

Double_t mass;
TLorentzVector ppSystem(p1,p2,p3,E);
KinFitter fitter(cand_vector);
void addMissingParticleConstraint(ppSystem, mass); // Choose momentum
    constraint for missing particle
fitter.fit();
TLorentzVector getMissingDaughter(); // Retrieve the missing daughter
    after the fit
```

To perform the 4C fit, in addition to the vector of final state particles, a TLorentzVector corresponding to the initial beam–target system must be passed to the fitter.

```
#include "TLorentzVector.h"
#include "KinFitter.h"

TLorentzVector ppSystem(p1,p2,p3,E);
KinFitter fitter(cand_vector);
fitter.add4Constraint(ppSystem); // Choose 4C fit
fitter.fit();
```

To perform a mass fit, in addition to the vector of final state particles, the mass of the particle must be passed to the fitter.

```
#include "KinFitter.h"

Double_t mass;
KinFitter fitter(cand_vector);
fitter.addMassConstraint(mass); // Choose mass fit
fitter.fit();
```

To perform a missing mass fit, in addition to the vector of final state particles, the mass of the missing particle and a TLorentzVector corresponding to the initial beam–target system must be passed to the fitter.

```
#include "TLorentzVector.h"
#include "KinFitter.h"

Double_t mass;
TLorentzVector ppSystem(p1,p2,p3,E);
KinFitter fitter(cand_vector);
fitter.addMissingMassConstraint(ppSystem, mass); // Choose missing
    mass fit
fitter.fit();
```

Since a mother particle needs to be constructed in order to perform the 3C fit, this is a bit more involved and an example of how to perform this fit and run the 3C fit is given below

```
#include "KinFitter.h"
#include "KFitVertexFinder.h"
#include "KFitDecayCandFinder.h"

#include "TLorentzVector.h"

#include <vector>
#include <algorithm>
#include <map>
#include <iostream>
#include <iomanip>
#include <math.h>

// Create a vector for particle pairs from the interaction point and
   decay vertex, respectively
std::vector<KFitParticle> cands1, cands2;
cands1.push_back(proton1_fit);
cands1.push_back(kaon_fit);
cands2.push_back(proton2_fit);
cands2.push_back(pion_fit);

// Initialize a KFitVertexFinder for each vertex
KFitVertexFinder vtx1finder(cands1);
KFitVertexFinder vtx2finder(cands2);

// Find vertex and retrieve it
TVector3 vtx1 = vtx1finder.getVertex();
TVector3 vtx2 = vtx2finder.getVertex();

// Vertex resolutions
Double_t vtx1_xres, vtx1_yres, vtx1_zres, vtx2_xres, vtx2_yres,
   vtx2_zres;

// Find the decaying candidate
KFitDecayCandFinder lambdafinder(cands2, 1.115683, vtx1, vtx2,
   vtx1_xres, vtx1_yres, vtx1_zres, vtx2_xres, vtx2_yres, vtx2_zres);
KFitParticle lambda_cand =
    lambdafinder.getDecayCand();

// Do 3C fit in decay vertex
KinFitter fitter(cands2);
fitter.add3Constraint(lambda_cand);
fitter.fit();

// Get fit result
KFitParticle fcand1 = fitter.getDaughter(0);
    // proton
KFitParticle fcand2 = fitter.getDaughter(1); // pion
KFitParticle lambda_fit = fitter.getMother();
    // lambda
TMatrixD test = lambda_fit.getCovariance();
    // lambda covariance
```

B: QA Plots

B1: Pull Distributions of 4C Fit

See Figs. 13, 14, 15 and 16.

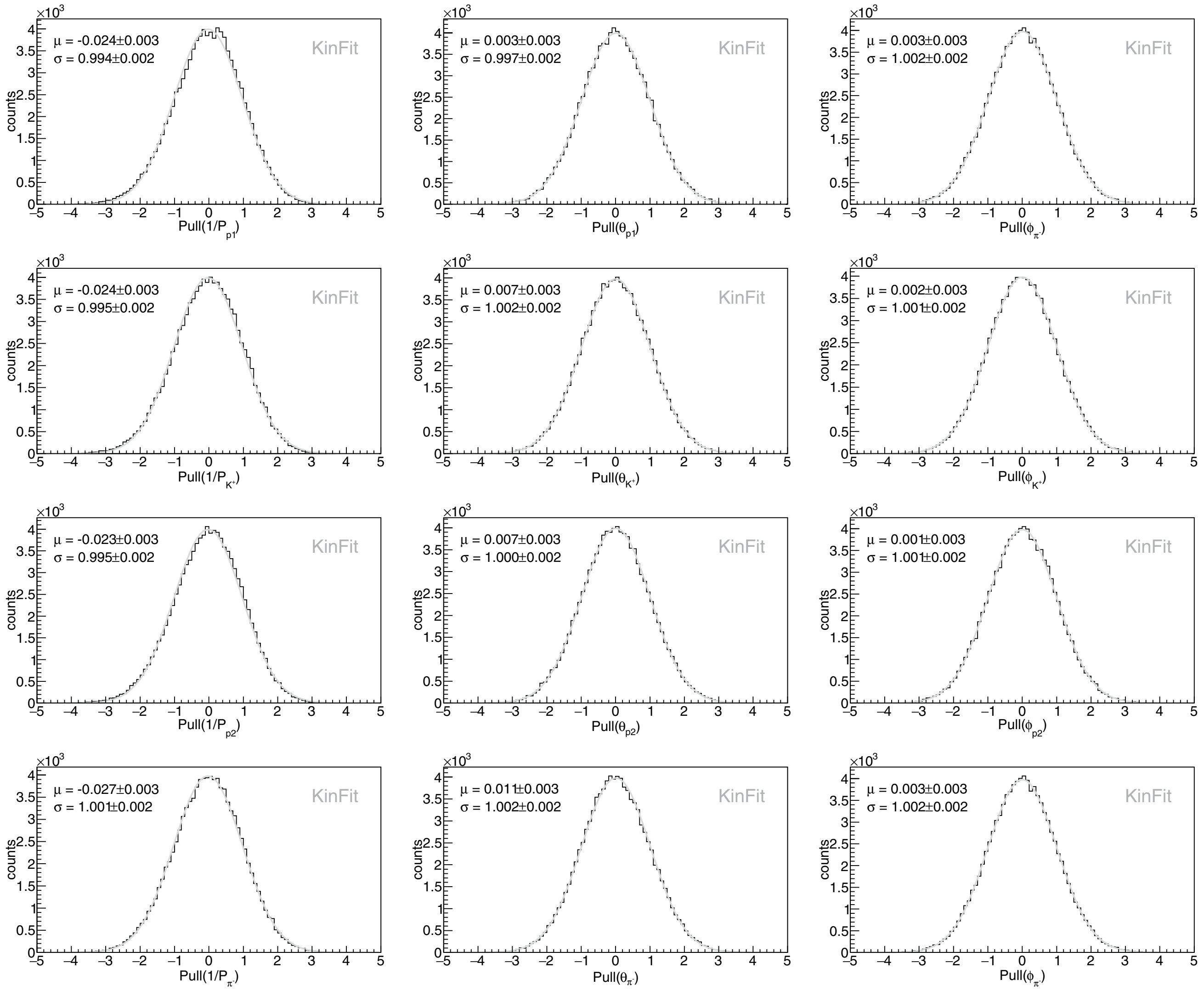


Fig. 13 Pull distributions for the track parameters of all particles after the 4C fit with respective mean and standard deviation

B2: Pull Distributions of Missing Particle Fit

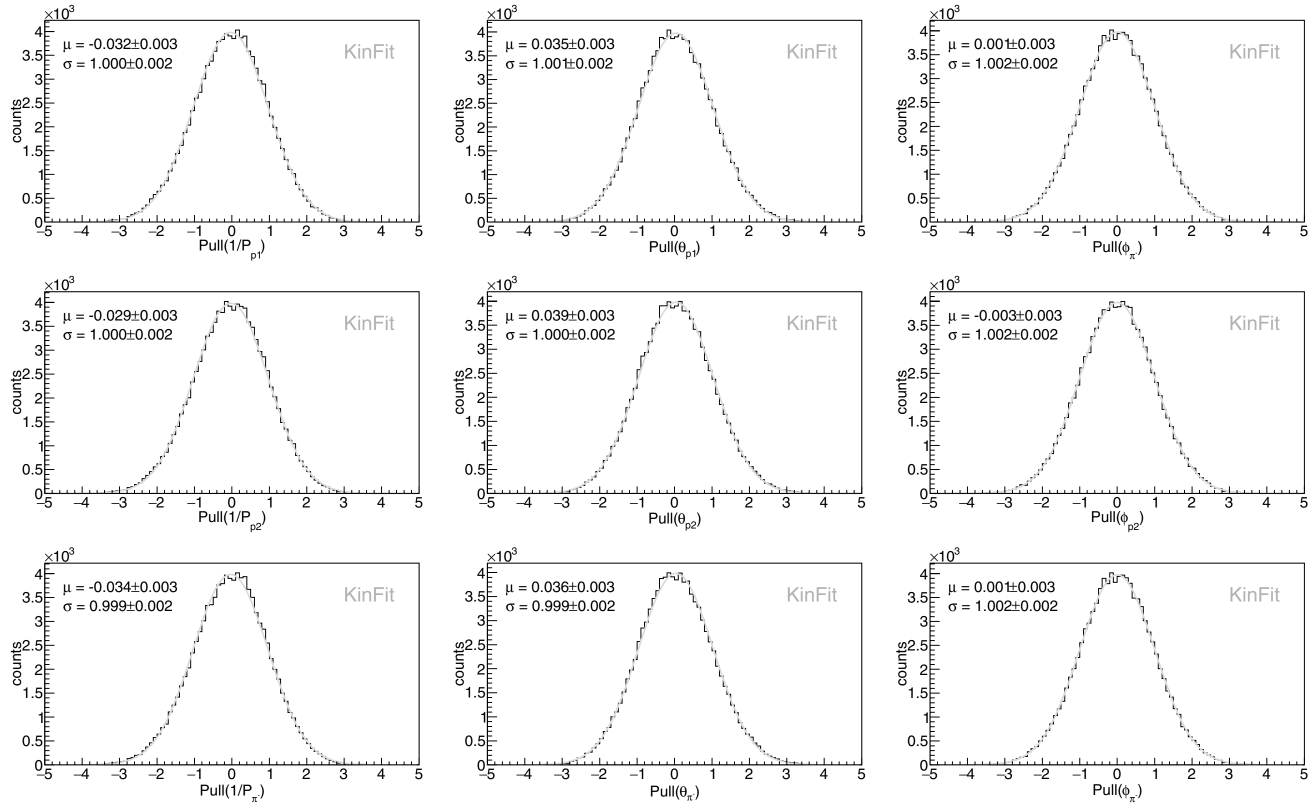


Fig. 14 Pull distributions for the track parameters of all particles after the missing K^+ fit with respective mean and standard deviation

B3: Pull Distributions of 3C Fit in Λ Decay Vertex

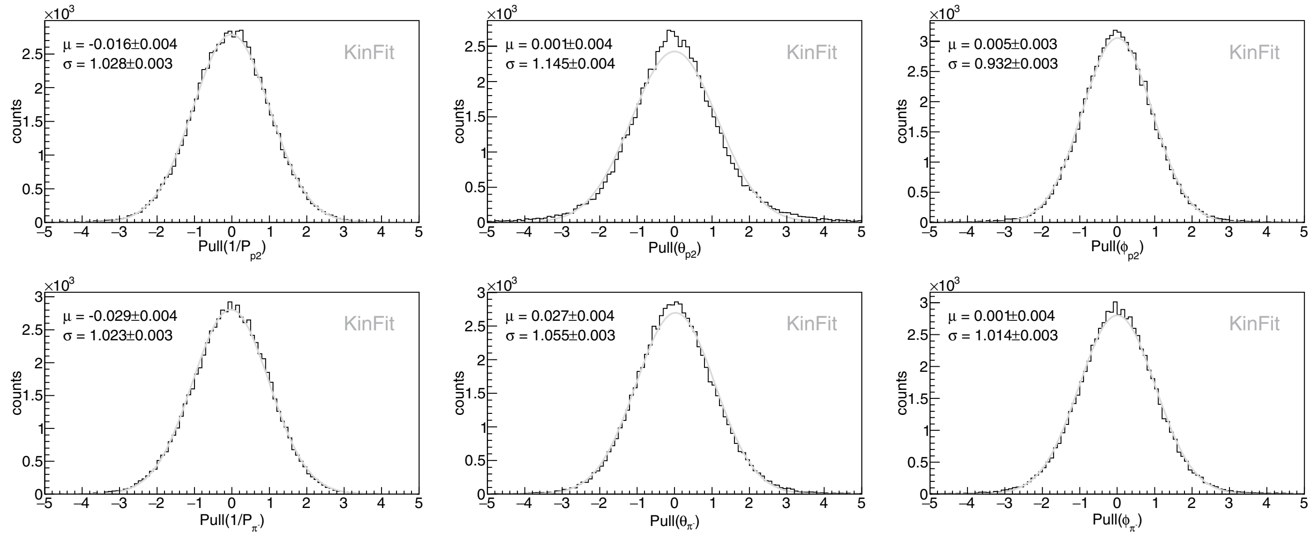


Fig. 15 Pull distributions for the track parameters of all particles after the 3C fit with respective mean and standard deviation

B4: Pull Distributions of the Vertex Fit in the Interaction Point

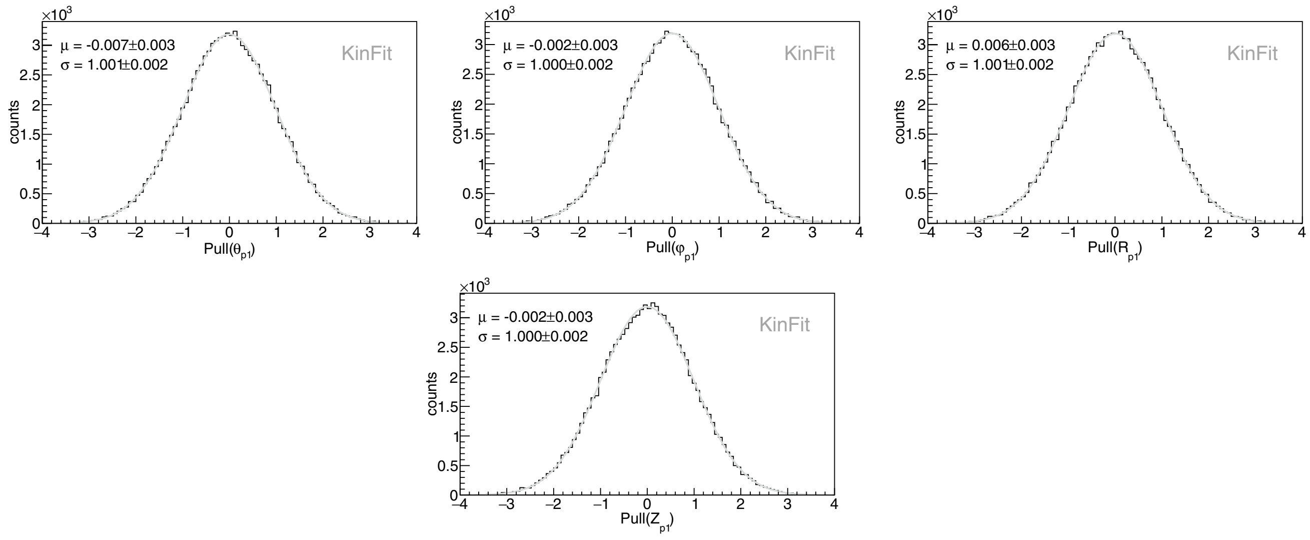


Fig. 16 Pull distributions for the track parameters of the primary proton tracks after the vertex fit with respective mean and standard deviation

Acknowledgements We would like to thank Tord Johansson for useful discussions on kinematic fitting of Λ hyperons. Furthermore, we are grateful to Anar Rustamonov for the suggestion to make KinFit available to a larger community as an independent package. Finally, we would like to express our appreciation of the fruitful discussions and the encouragements we received from James Ritman and Piotr Salabura. The publication is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—491382106, and by the Open Access Publishing Fund of GSI Helmholtzzentrum fuer Schwerionenforschung. This project has received funding from the Knut and Alice Wallenberg Foundation, Contract No. 2016.0157 and 2021.0299 (Sweden), and the Swedish Research Council, Contract No. 2019–04594 (Sweden).

Author Contributions This work is conceptualized by WE and KS. The development and implementation of the C++ `KinFit` package was done by WE, JR and JT. The first paper draft preparation, was done by WE, JR, and JT and the drafting was coordinated by JR. The project was supervised by KS. Benchmark studies and bug fixes were performed by JR, JT and MB, while all authors contributed to the review of the paper's intellectual content and the editing.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The simulated data used in this article are available in the github repository:

```
https://github.com/KinFit/
    KinFit.git
```

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Bock RK (1960) Application of a Generalized Method of Least Squares for Kinematical Analysis of Tracks in Bubble Chambers. CERN Yellow Reports: Monographs. CERN, Geneva https://doi.org/10.5170/CERN-1960-030

2. Moser F, Waltenberger W, Regler M, Mitaroff W (2009) Implementation and application of kinematic vertex fitting in the software environment of ILD. arXiv. https://doi.org/10.48550/ARXIV.0901.4020
3. Krohn J-F et al (2020) Global decay chain vertex fitting at belle II. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 976:164269. https://doi.org/10.1016/j.nima.2020.164269
4. PANDA Collaboration (2021) Panda phase one. Eur. Phys. J. A **184**(57) https://doi.org/10.1140/epja/s10050-021-00475-y
5. Radkhorrami Y, List J (2021) Kinematic Fitting for ParticleFlow Detectors at Future Higgs Factories. arXiv. https://doi.org/10.48550/ARXIV.2110.13731
6. Ai X, Allaire C, Calace N, et al (2022) A common tracking software project. Computing and Software for Big Science 6, 8 https://doi.org/10.1007/s41781-021-00078-8
7. Kiesel, (2018) I for the CBM Collaboration: Event topology reconstruction in the cbm experiment. J. Phys.: Conf. Ser. 1070, 012015
8. Fisyak Y, Ivanov V, Ke H et al (2021) Application of the missing mass method in the fixed-target program of the star experiment. EPJ Web Conf. 251:04029. https://doi.org/10.1051/epjconf/202125104029
9. HADES Collaboration (2009) The high-acceptance dielectron spectrometer HADES. The European Physical Journal A 41(2), 243–277 https://doi.org/10.1140/epja/i2009-10807-5
10. The HADES Collaboration and PANDA@HADES Collaboration (2021) Production and electromagnetic decay of hyperons: a feasibility study with HADES as a phase-0 experiment at FAIR. Eur. Phys. J. A 57(4), 138 https://doi.org/10.1140/epja/s10050-021-00388-warXiv:2010.06961
11. Ohnishi H, Sakuma F, Takahashi T (2020) Hadron physics at j-parc. Progress in Particle and Nuclear Physics 113:103773. https://doi.org/10.1016/j.ppnp.2020.103773
12. JLab Experiments. https://www.jlab.org/physics/experiments. Accessed: 2023-05-05
13. Abbon P et al (2015) The COMPASS setup for physics with hadron beams. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 779:69–115. https://doi.org/10.1016/j.nima.2015.01.035
14. Brun R, Rademakers F (1997) ROOT: An object oriented data analysis framework. Nucl. Instrum. Meth. A 389:81–86. https://doi.org/10.1016/S0168-9002(97)00048-X
15. Frodesen AG, Skjeggestad O (1979) Probability and Statistics in Particle Physics. Universitetsforlaget, Bergen, Norway
16. Zyla PA et al (2020) Review of Particle Physics. PTEP 2020(8):083–01. https://doi.org/10.1093/ptep/ptaa104
17. CMake. https://cmake.org/. Accessed: 2022-12-15
18. Fröhlich I et al (2010) Design of the pluto event generator. Journal of Physics: Conference Series 219(3):032039. https://doi.org/10.1088/1742-6596/219/3/032039
19. Rodrigues E et al (2020) The Scikit HEP Project - overview and prospects. EPJ Web Conf. 245:06028. https://doi.org/10.1051/epjconf/202024506028. arXiv:2007.03577

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.