FUMILI-based minimization with constraints using method of elimination of differentials

Kurbatov V. S.[1], Tokareva V. A.[2,*], Tsirkov D. A.[1]
[1] *DLNP, JINR, Dubna, Russia*
[2] *IKP, KIT, Karlsruhe, Germany*
[*] `tokareva.vict@gmail.com`

Competition for JINR Awards for young scientists and specialists

December 4, 2018

FUMILI minimizer

FUMILI is one of the first minimizers included into ROOT release. It has been showing it's reliability, stability and high convergence rate while it had been being used by scientific community for decades.

The greedy minimization algorithm which is employed in FUMILI was first proposed at JINR by S. Sokolov and implemented by I. N. Silin and V. S. Kurbatov.

FUMILI provides an optimal solution for χ^2-like functionals (1) employing linearization:

$$F(\mathbf{x}) = \sum_{k=1}^{K} f_k^2(\mathbf{x}) = \sum_{k=1}^{K} \left(\frac{Y_k - T_k(\mathbf{x})}{\sigma_k} \right)^2, \tag{1}$$

where Y_k are measured values with errors σ_k, $k \in [1, K]$, and $T_k(\mathbf{x})$ are the values predicted by the model, depending on some parameters $\mathbf{x} = \{x_1, \dots, x_n\}$.

Linearization method in minimizing χ^2-like functionals

The second derivative $\dfrac{\partial^2 F}{\partial x_i x_j}$ could be found the following way:

$$\begin{aligned}\frac{\partial^2 F}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}\sum_{k=1}^{K} f_k^2(\mathbf{x}) = \frac{\partial}{\partial x_i}\sum_{k=1}^{K} 2 f_k(\mathbf{x})\frac{\partial f_k(\mathbf{x})}{\partial x_j} = \\ &= 2\sum_{k=1}^{K}\left(\frac{\partial f_k(\mathbf{x})}{\partial x_i}\frac{\partial f_k(\mathbf{x})}{\partial x_j} + f_k(\mathbf{x})\frac{\partial^2 f_k(\mathbf{x})}{\partial x_i \partial x_j}\right)\end{aligned} \qquad (2)$$

Linearization means discarding the second term $f_k \dfrac{\partial^2 f_k}{\partial x_i \partial x_j}$ employing second derivatives, that is considered small in comparison to the first one $\dfrac{\partial f_k}{\partial x_i}\dfrac{\partial f_k}{\partial x_j}$.

Its main benefit is that the error matrix for a linearized functional is always positively defined, and thus each step leads to a minimum.

What are constraints

Constraints: additional restrictions on the minimization problem in form of equations

$$\Phi(\mathbf{x}) = \begin{cases} \phi_1(\mathbf{x}) = 0, \\ \cdots \\ \phi_m(\mathbf{x}) = 0. \end{cases} \tag{3}$$

$\mathbf{x} = \{x_1, \ldots, x_n\}$: a vector of parameters, usually $m < n$.

Simple cases: redundant parameters of functional (1) could be eliminated directly by solving the system (3).

Complicated cases: the constraint equations could be non-linear, thus it is impossible or impractical to solve (3).

What is kinematic fitting

The problem of minimizing functionals with constraints arises, for example, in the task of kinematic fitting.

Kinematic fitting

- Tracking detectors provide the coordinates of the triggered sensitive elements along with their errors;
- Track-finding involves fitting the particle trajectories to these coordinates;
- Sometimes, when the reaction channel is known, the additional information on kinematics could be utilized in terms of conservation laws: $\sum E_{\text{initial}} = \sum E_{\text{final}}$, $\sum \vec{P}_{\text{initial}} = \sum \vec{P}_{\text{final}}$;
  missing mass: $\left| \sum \mathcal{P}_{\text{initial}} - \sum \mathcal{P}_{\text{final}} \right|^2 = M_X^2$;
  This is called kinematic fitting.

History of constrained minimization

Method of Lagrange multipliers

- First proposed at early sixties, see e. g. J. P. Berge, F. T. Solmitz, H. D. Taft, Rev. Sci. Instr. 32 (1961) 538;
- Uses Lagrange multipliers λ_i, obtained from the equations

$$\frac{\partial\Psi}{\partial x_1} = \frac{\partial\Psi}{\partial x_2} = \cdots = \frac{\partial\Psi}{\partial x_n} = 0,$$

  where $\Psi(\mathbf{x}) = F(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \phi_i(\mathbf{x})$;
- Still the most widely used method for kinematic fitting, see e. g. KWFIT package `http://www.phys.ufl.edu/~avery/kwfit/`.

History of constrained minimization

Penalty-function method

- Proposed in JINR in mid-sixties, see V.I. Moroz, JINR communications R-1958 (1965);
- Adds a so-called “heavy term” to the minimized functional, designed in a way that values of constraint functions approach zero as this term approaches infinity:

$$\tilde{\Psi}(\mathbf{x}) = F(\mathbf{x}) + T \sum_{i=1}^{m} \phi_i^2(\mathbf{x}),\ T \to \infty.$$

- The method is very robust and almost always converges, which could be both a benefit (you won’t miss a minimum) and a drawback (you should carefully control that your minimum is reasonable).

Method of elimination of differentials

In the neighborhood of a point $\mathbf{x}_0$ the functional $F(\mathbf{x})$ could be expressed as

$$\begin{aligned} F(\mathbf{x}_0 + \Delta\mathbf{x}) &= F(\mathbf{x}_0) + \sum_{i=1}^{n} \frac{\partial F(\mathbf{x}_0)}{\partial x_i}\Delta x_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \Delta x_i \frac{\partial^2 F(\mathbf{x}_0)}{\partial x_i \partial x_j}\Delta x_j \\ &= F(\mathbf{x}_0) + G(\mathbf{x}_0)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T Z(\mathbf{x}_0)\Delta\mathbf{x}, \end{aligned} \tag{4}$$

and the constraints $\Phi(\mathbf{x})$ as

$$\Phi(\mathbf{x}_0 + \Delta\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}_0) + \sum_{i=1}^{n} \frac{\partial \phi_1(\mathbf{x}_0)}{\partial x_i}\Delta x_i \\ \cdots \\ \phi_m(\mathbf{x}_0) + \sum_{i=1}^{n} \frac{\partial \phi_m(\mathbf{x}_0)}{\partial x_i}\Delta x_i \end{bmatrix} = \Phi(\mathbf{x}_0) + D(\mathbf{x}_0)\Delta\mathbf{x}. \tag{5}$$

Here G, Z and D are the derivatives in matrix form.

Method of elimination of differentials

In the matrix equation $\Phi(\mathbf{x}) = \Phi(\mathbf{x}_0) + D(\mathbf{x}_0)\Delta\mathbf{x}$ the rectangular matrix D has m rows and n columns (we have n parameters and m constraints).

The vector $\Delta\mathbf{x}$ could be split into $\Delta\mathbf{x}_c$ that has m components, and $\Delta\mathbf{x}_f$ that has $n-m$ components; the same could be done with the matrix D. Then

$$\Phi(\mathbf{x}) = \Phi(\mathbf{x}_0) + D_c(\mathbf{x}_0)\Delta\mathbf{x}_c + D_f(\mathbf{x}_0)\Delta\mathbf{x}_f. \tag{6}$$

Since (6) is a linear equation, $\Delta\mathbf{x}_c$ could be expressed via $\Delta\mathbf{x}_f$, resulting in

$$\Delta\mathbf{x}_c = \mathbf{v} + M\Delta\mathbf{x}_f. \tag{7}$$

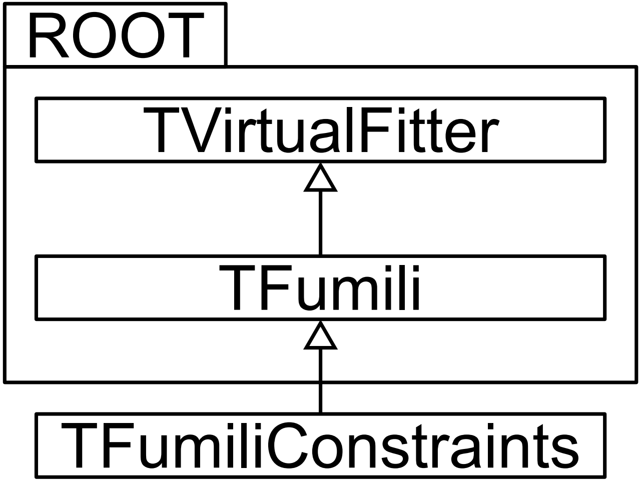
Returning to the initial functional (4)

$$F(\mathbf{x}) = F(\mathbf{x}_0) + G(\mathbf{x}_0)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T Z(\mathbf{x}_0)\Delta\mathbf{x},$$

we could employ (7) to eliminate the sub-vector $\Delta\mathbf{x}_c$, and obtain a similar functional, in contrast depending on only $n-m$ increments $\Delta\mathbf{x}_f$:

$$F(\mathbf{x}) = F'(\mathbf{x}_0) + G'(\mathbf{x}_0)\Delta\mathbf{x}_f + \frac{1}{2}\Delta\mathbf{x}_f^T Z'(\mathbf{x}_0)\Delta\mathbf{x}_f. \tag{8}$$

Integration with TVirtualFitter

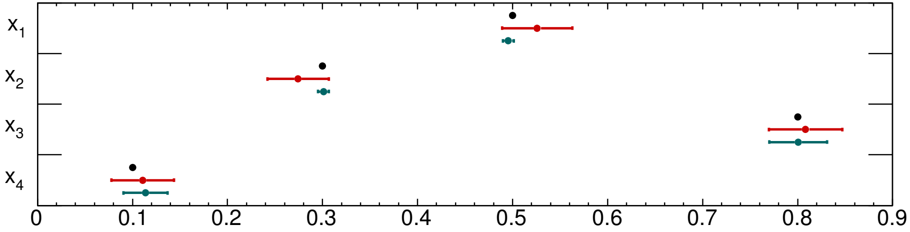


User API

```
void FCN(int & n_par, double * grad,
         double & val, double * par, int flag);
/* ... */
TFumiliConstraints * fum = new TFumiliConstraints;
// set objective function
fum->SetFCN(FCN);
// set parameters
fum->SetParNumber(2);
fum->SetParameter(0, "#alpha", .5, 0.01, 0, 0);
fum->SetParameter(1, "#beta", .0, 0.01, 0, 0);
// set constraints
fum->SetConstrNumber(1);
fum->SetConstraint(0, [](double * p){
  return p[0]*p[0] + .5*p[1] - 1.3;
});
fum->SetConstrDeriv(0, 0, [](double * p){ return 2*p[0]; });
fum->SetConstrDeriv(0, 1, .5);
// minimize
fum->Minimize();
```

Testing on a “toy” sample

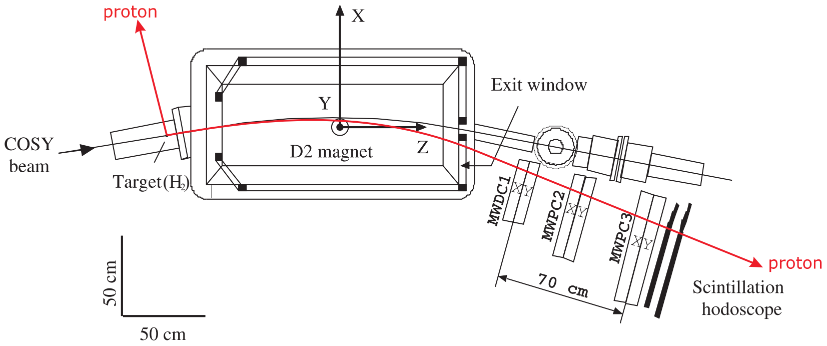
A set of 500 000 (a, b) events, Monte-Carlo generated according to the function $1 + x_1 a + x_2 a^2 + x_3 b + x_4 b^2$ with parameters $\{x_1 = 0.5,\ x_2 = 0.3,\ x_3 = 0.8,\ x_4 = 0.1\}$; and fitted using an event-by-event log. likelihood method with constraints $x_1^2 + x_1 x_4 - x_4^2 = 0.29$, $x_2^2/x_3 = 0.1125$.



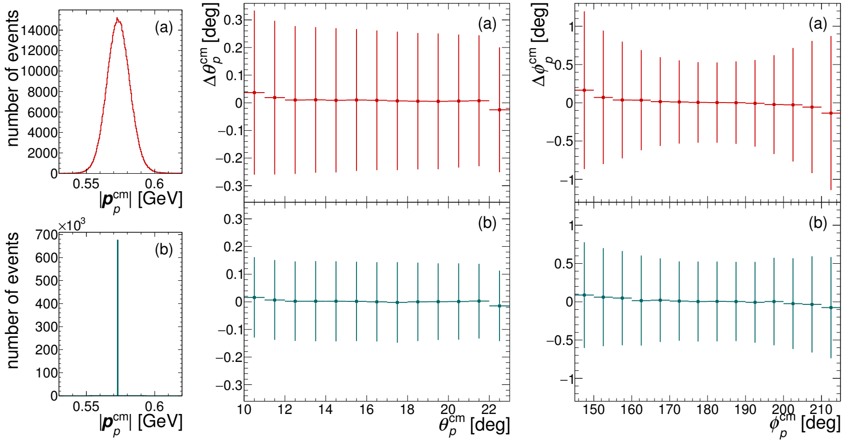
Parameter	True values	Unconstrained fit	Constrained fit
x_1	0.5	0.526 ± 0.037	0.496 ± 0.006
x_2	0.3	0.274 ± 0.032	0.301 ± 0.006
x_3	0.8	0.808 ± 0.039	0.801 ± 0.030
x_4	0.1	0.111 ± 0.033	0.114 ± 0.023

Kinematic fitting at ANKE, $pp \to pp$

- Reaction $pp \to pp$ at ANKE;
- Undetected proton;
- A constraint $|\mathcal{P}_{\text{beam}} + \mathcal{P}_{\text{targ}} - \mathcal{P}_p|^2 = m_p^2$.



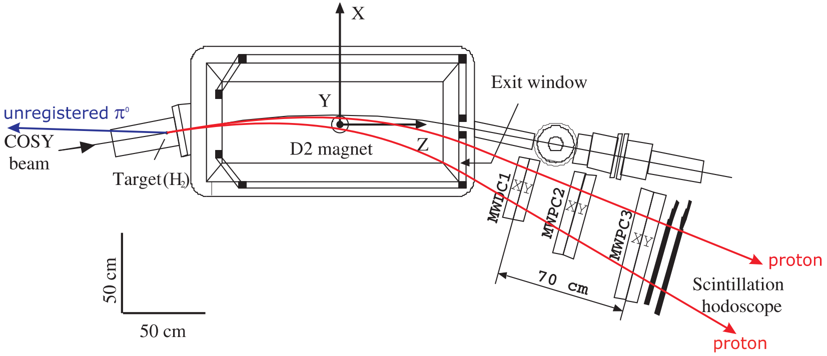
Kinematic fitting at ANKE, $pp \to pp$



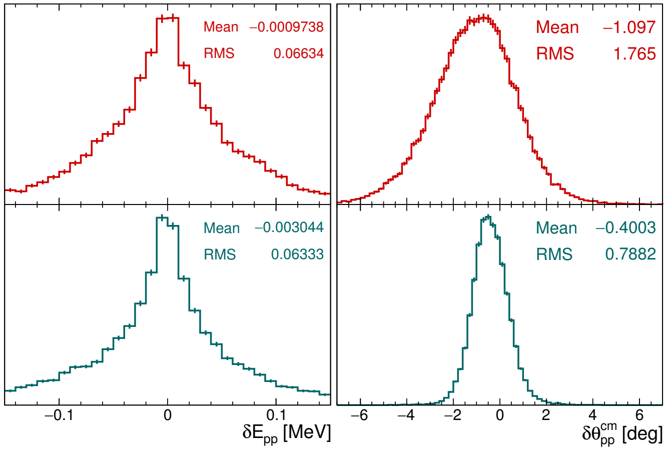
Errors of reconstructed proton momentum in polar coordinates $(|P_p^{\rm cm}|, \theta_p^{\rm cm}, \phi_p^{\rm cm})$ for the $pp \to pp$ reaction at ANKE (used for luminosity estimation), simulated for $T_{\rm beam} = 700$ MeV with and without kinematic fitting.

Kinematic fitting at ANKE, $pp \to \{pp\}_s\pi^0$

- Reaction $pp \to \{pp\}_s\pi^0$ at ANKE;
- Undetected pion;
- A constraint $|\mathcal{P}_{\text{beam}} + \mathcal{P}_{\text{targ}} - \mathcal{P}_{p1} - \mathcal{P}_{p2}|^2 = m^2_{\pi^0}$.

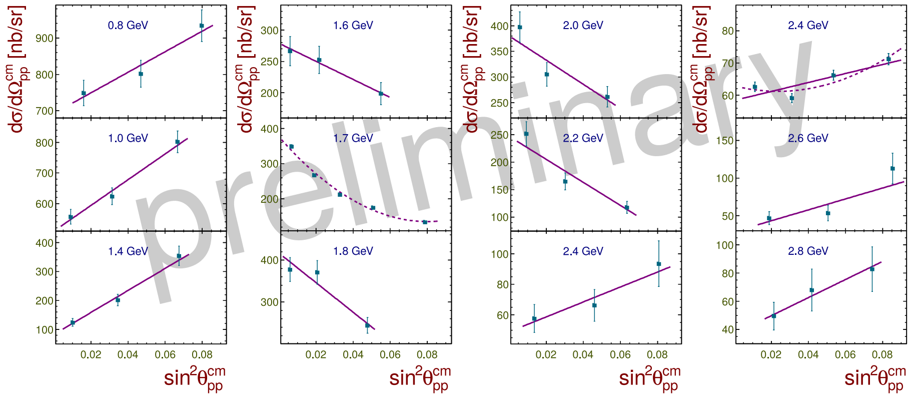


Kinematic fitting at ANKE, $pp \to \{pp\}_s\pi^0$



Errors of reconstruction with and without kinematic fitting for two main kinematic variables: excitation energy of the proton pair E_{pp} and its polar angle in the center-of-mass system θ_{pp}^{cm}. The error is calculated as a difference between the reconstructed value and the true value used as an input by the simulation software.

Measured $pp \to \{pp\}_s\pi^0$ differential cross sections



Differential cross sections of the $pp \to \{pp\}_s\pi^0$ reaction at all analyzed energies. The solid lines correspond to the linear fit, the dashed ones to the quadratic one.

Presented on September 21, 2018, at the XXIV Baldin Seminar by V. Kurbatov; article in preparation.

Kinematic fitting at ANKE, $pp \to \{pp\}_s\pi^0$

Slope parameters with their statistical and systematic errors estimated with and without kinematic fitting for three beam energies.

T_{beam}	slope $\pm\, \sigma_{\mathrm{stat}} \pm \sigma_{\mathrm{syst}}$	slope $\pm\, \sigma_{\mathrm{stat}} \pm \sigma_{\mathrm{syst}}$
500 MeV	$3.35 \pm 0.56 \pm 1.18$	$1.96 \pm 0.56 \pm 0.04$
550 MeV	$2.21 \pm 0.36 \pm 0.71$	$0.82 \pm 0.36 \pm 0.09$
700 MeV	$1.21 \pm 0.33 \pm 0.36$	$2.10 \pm 0.33 \pm 0.06$

Outlook

- Refactoring the code and covering it with tests;
- Adding the method of Lagrange multipliers to the code;
- Adding the penalty-function method to the code;
- Open-source release, preview:
  `https://bitbucket.org/ParallelMe/fumili_constraints`.

Thank you
for your attention!

Any questions?