Kalman Filtering Update in traccc

Beomki Yeo

Kalman Filtering Update (tracc#264)

- CPU Kalman filtering is updated with detray tracking geometry
- Still in WIP status, but it will be completed soon after updating detray version and polishing the PR
- In this presentation, I will address the major features of KF implementation and its validation results

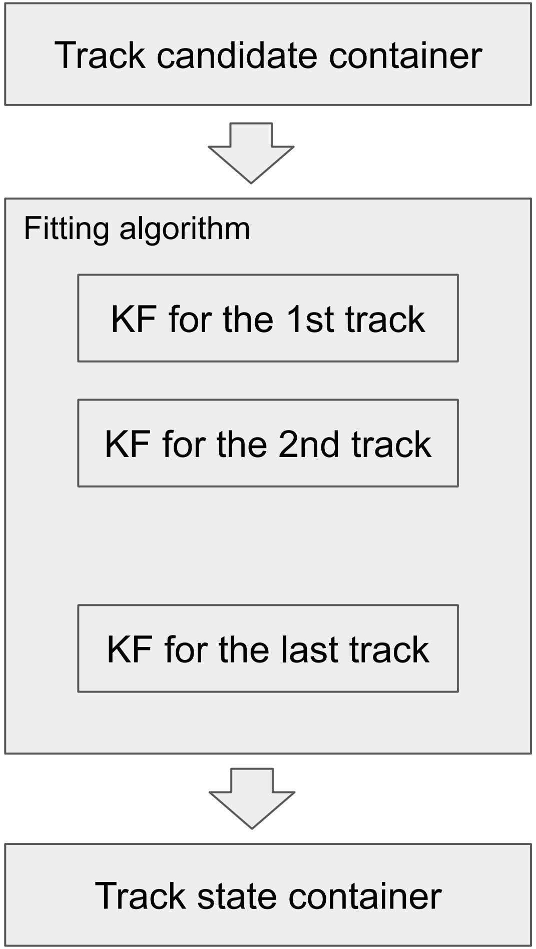
New EDMs

- Track candidates as inputs for kalman filtering
  - Header: seed track parameter of a track
  - Items: Candidate measurements per track

- Track states as outputs for kalman filtering
  - Header: Fitting information of a track
  - Item: Fitting information of measurements per track

Algorithm Procedure

- Fitting algorithm takes track candidate container as an input which represents a set of tracks
- Fitting algorithm runs the kalman filtering iterates over the tracks
- The results of kalman filtering is added to the track state container



Track candidate container

Fitting algorithm

KF for the 1st track

KF for the 2nd track

KF for the last track

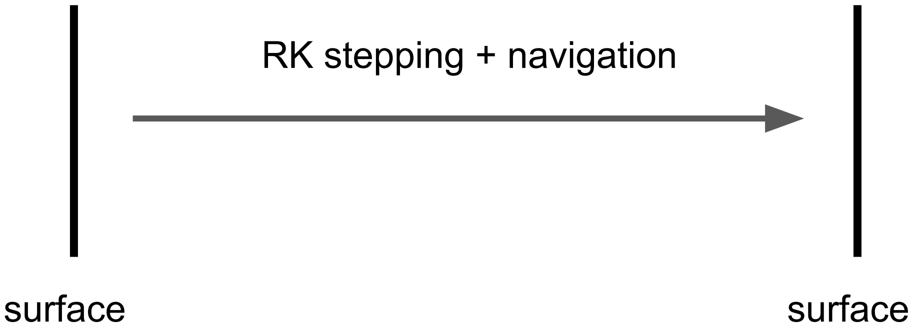
Track state container

Actor chain for kalman filtering

- The actor chain is a series of actions triggered for every surface intersection during the propagation

- Kalman fitting has four actors:
  - Parameter transporter to calculate the jacobian from the previous surface to the current one
  - Material interactor to calculate the energy loss and multiple scattering effect
  - Kalman actor to perform kalman filtering
  - Parameter resetter to reset the track parameters for the next surface intersection

- Smoothing of track parameters is done after the propagation is over



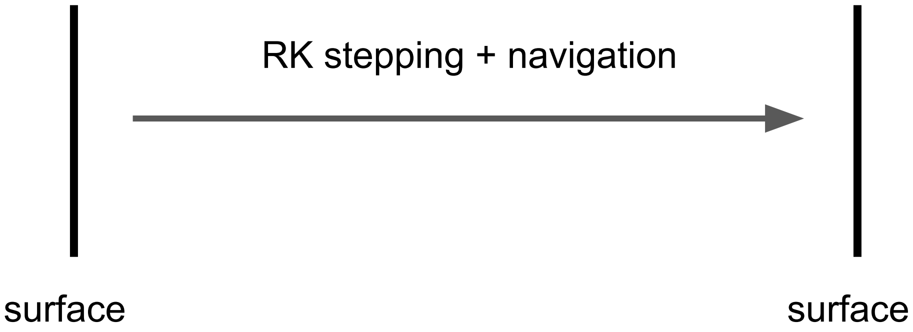
Actors:

1. Parameter transporter
2. Material interactor
3. Kalman filter
4. Parameter resetter

Actor chain for simulation

- Simulation can be done by replacing some actors:
  - Random scatterer to deflect the track based on the covariance calculated in material interactor
  - Event writer to record the measurements and hits at each surface

- Currently, only muon-like charged particles can be simulated

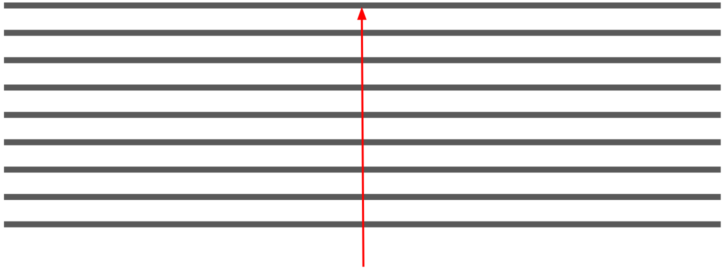


Actors:

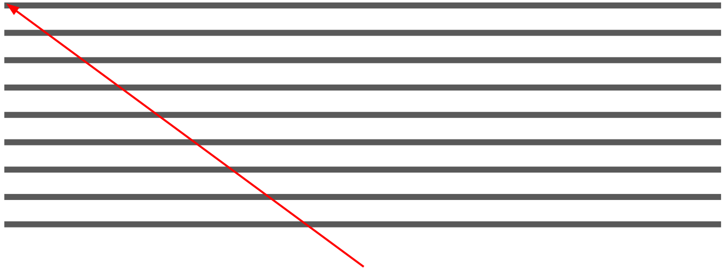
1. Parameter transporter
2. Material interactor
3. Random scatterer
4. Parameter resetter
5. Event writer

Simulation setups for KF validation

- **Double** precision
- Telescope geometry where the 9 planes aligned along the x-axis (2 cm gap)
- 2 Tesla B field in the x-axis
- 50 um measurement resolution
- 1 GeV/c momentum
- Tested for two data sets:
  A. 5e4 events with zero incidence angle
  B. 5e4 events with PI/3 (60 degree) incidence angle

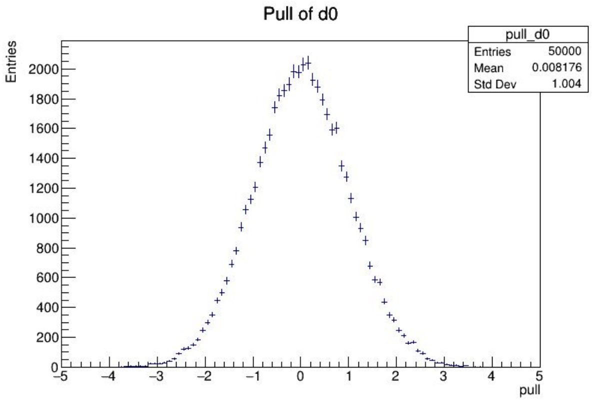


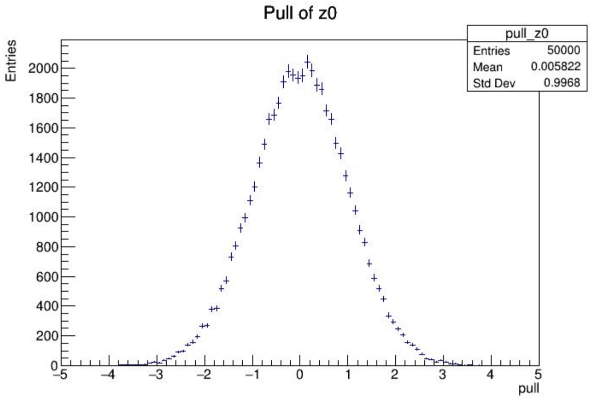
(A)

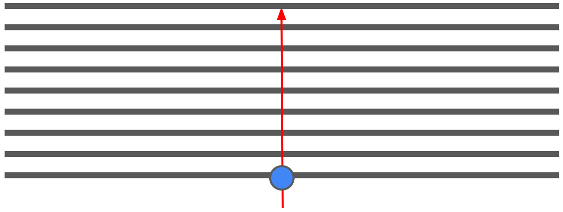


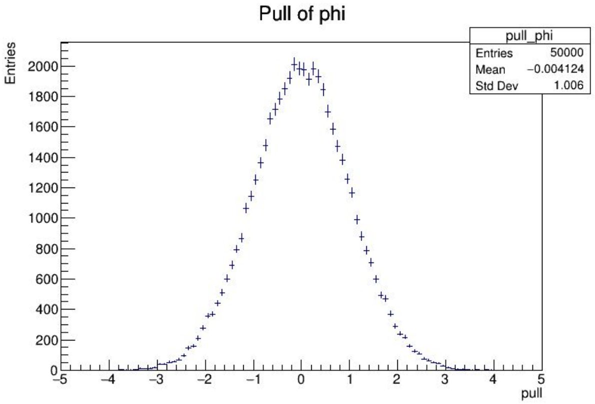
(B)

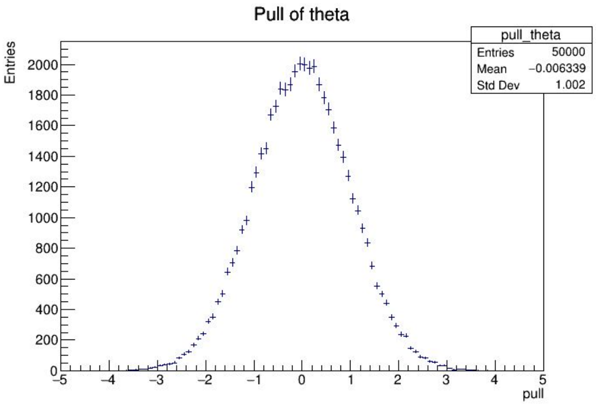
(A) Pull value distributions at the **FIRST** surface

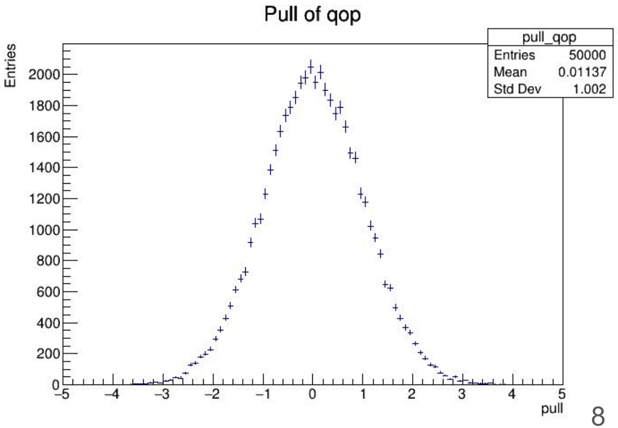




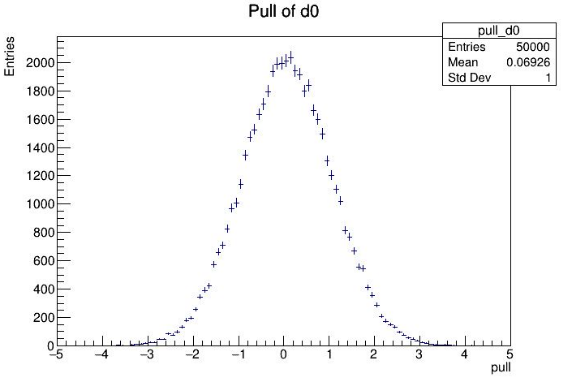


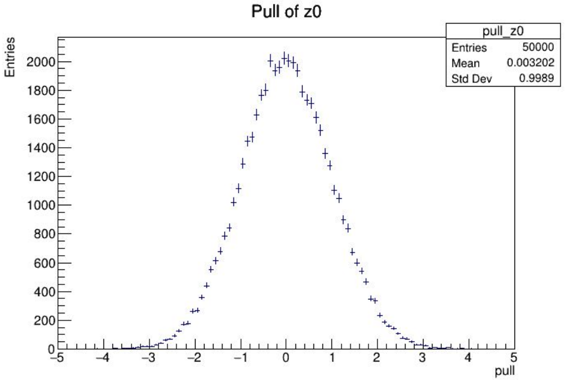


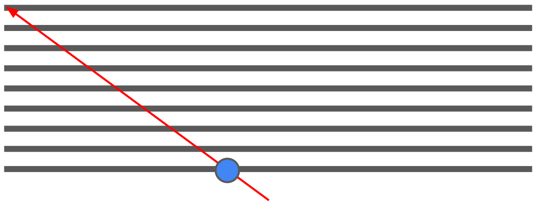


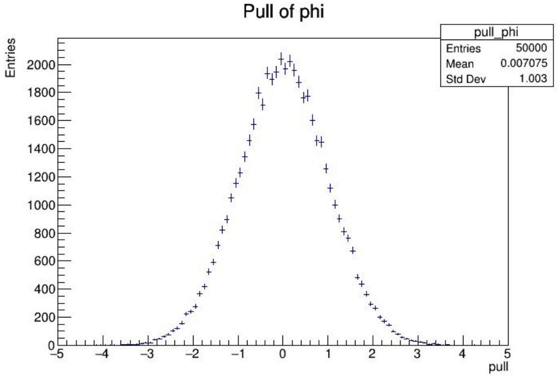


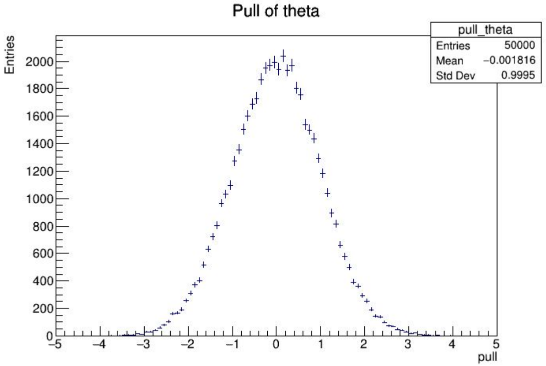
(B) Pull value distributions at the **FIRST** surface

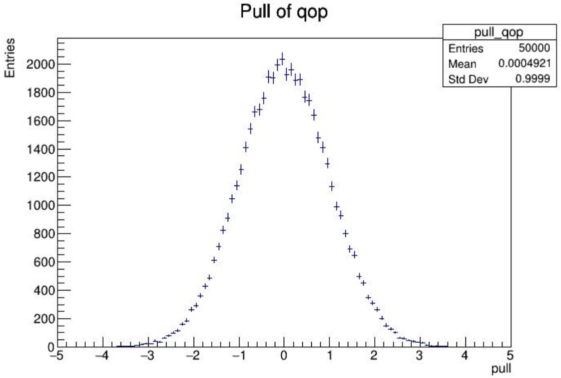




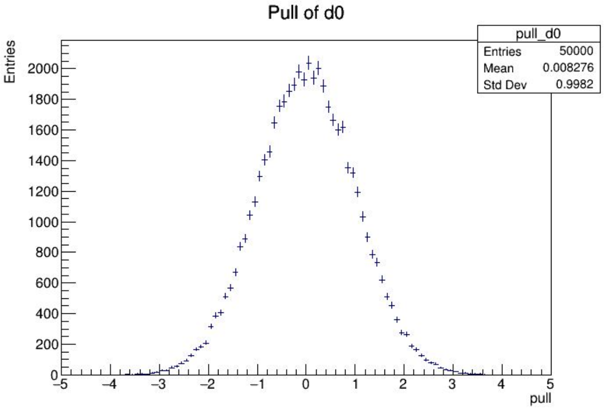


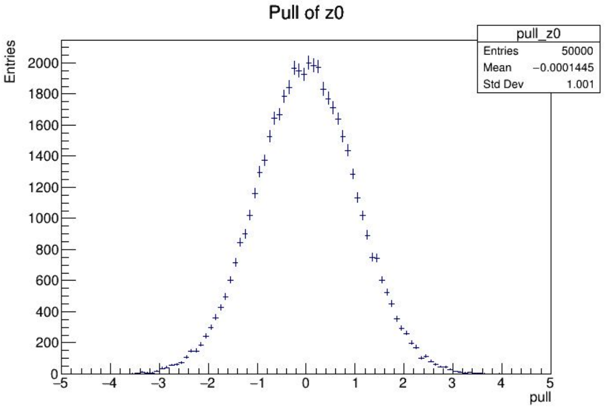


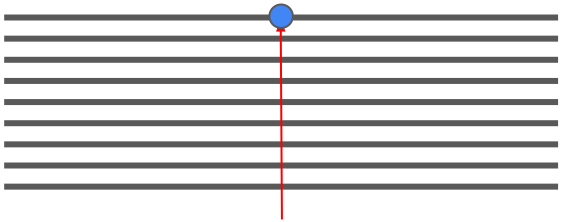


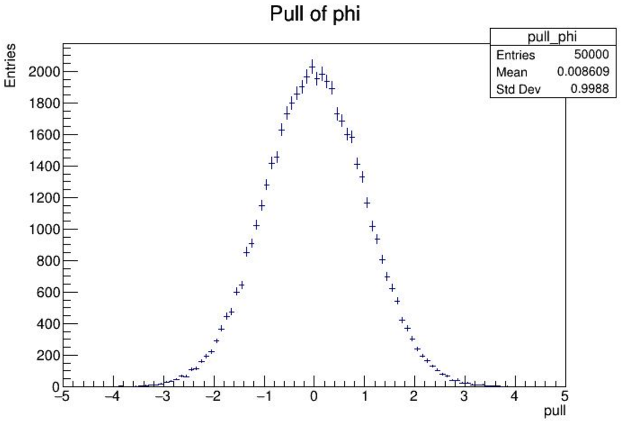


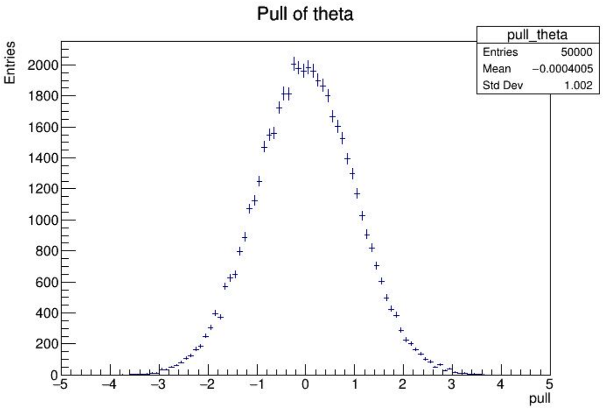
(A) Pull value distributions at the **LAST** surface

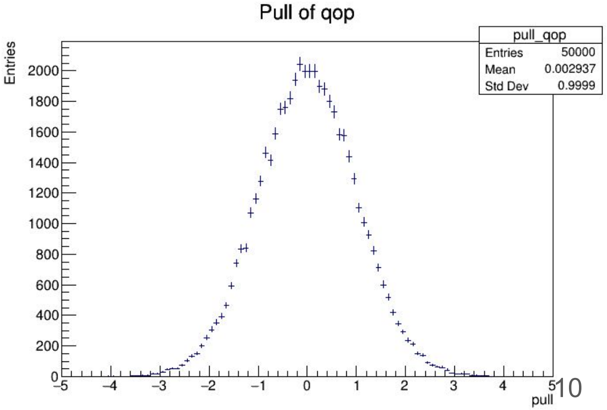




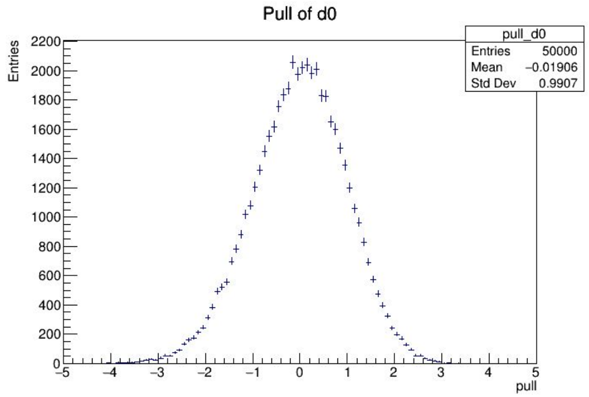


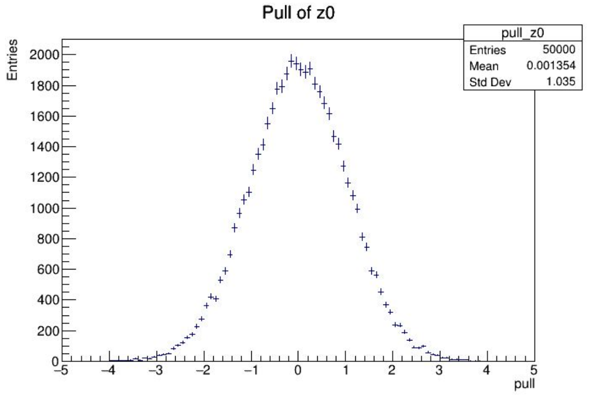


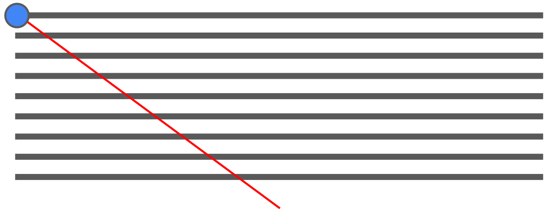


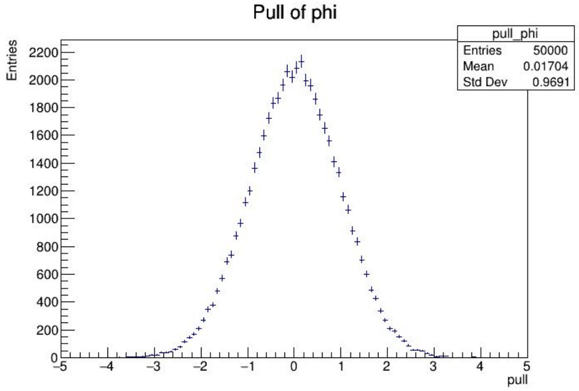


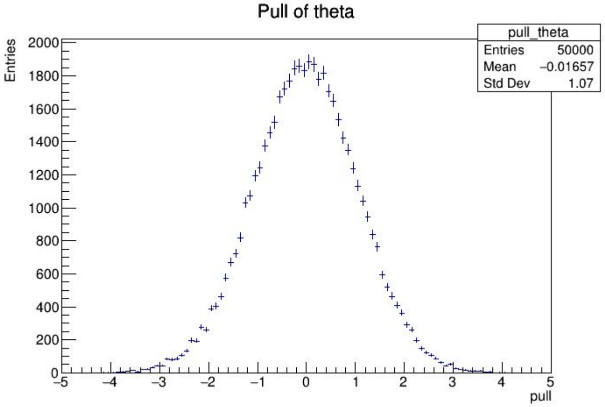
(B) Pull value distributions at the **LAST** surface

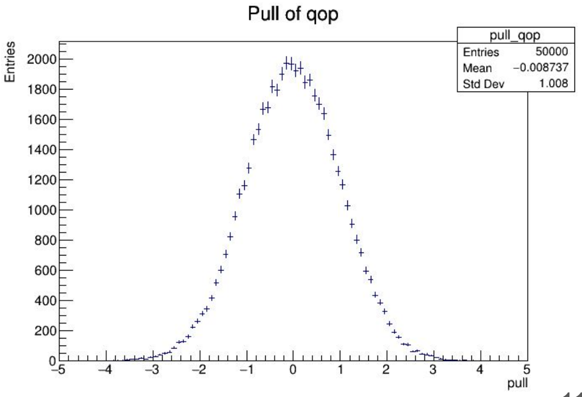












Summary

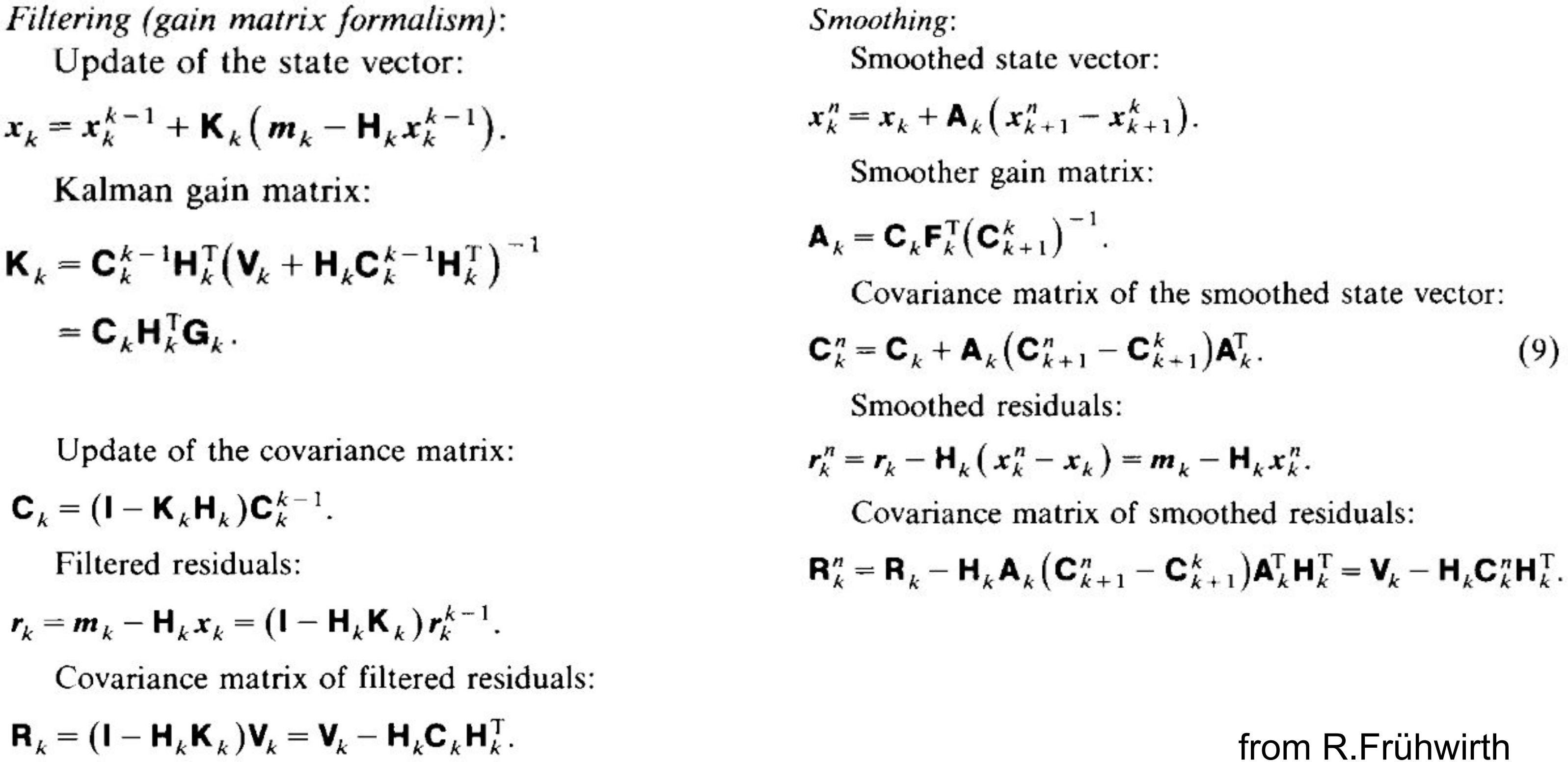
- The Kalman filtering in traccc was validated with the detray telescope detector

Future works

- Implement the GPU version
- Need to test again with detray toy geometry (trackML detector)
- Make the single precision work: Currently, the chi square blows up already in the first surface

BACK UP

Kalman actor (Kalman filtering) and smoothing



Filtering (gain matrix formalism):

Update of the state vector:

$$\boldsymbol{x}_k = \boldsymbol{x}_k^{k-1} + \mathbf{K}_k\left(\boldsymbol{m}_k - \mathbf{H}_k\boldsymbol{x}_k^{k-1}\right).$$

Kalman gain matrix:

$$\mathbf{K}_k = \mathbf{C}_k^{k-1}\mathbf{H}_k^{\mathrm{T}}\left(\mathbf{V}_k + \mathbf{H}_k\mathbf{C}_k^{k-1}\mathbf{H}_k^{\mathrm{T}}\right)^{-1}$$
$$= \mathbf{C}_k\mathbf{H}_k^{\mathrm{T}}\mathbf{G}_k .$$

Update of the covariance matrix:

$$\mathbf{C}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{C}_k^{k-1}.$$

Filtered residuals:

$$\boldsymbol{r}_k = \boldsymbol{m}_k - \mathbf{H}_k\boldsymbol{x}_k = (\mathbf{I} - \mathbf{H}_k\mathbf{K}_k)\boldsymbol{r}_k^{k-1}.$$

Covariance matrix of filtered residuals:

$$\mathbf{R}_k = (\mathbf{I} - \mathbf{H}_k\mathbf{K}_k)\mathbf{V}_k = \mathbf{V}_k - \mathbf{H}_k\mathbf{C}_k\mathbf{H}_k^{\mathrm{T}}.$$

Smoothing:

Smoothed state vector:

$$\boldsymbol{x}_k^n = \boldsymbol{x}_k + \mathbf{A}_k\left(\boldsymbol{x}_{k+1}^n - \boldsymbol{x}_{k+1}^k\right).$$

Smoother gain matrix:

$$\mathbf{A}_k = \mathbf{C}_k\mathbf{F}_k^{\mathrm{T}}\left(\mathbf{C}_{k+1}^k\right)^{-1}.$$

Covariance matrix of the smoothed state vector:

$$\mathbf{C}_k^n = \mathbf{C}_k + \mathbf{A}_k\left(\mathbf{C}_{k+1}^n - \mathbf{C}_{k+1}^k\right)\mathbf{A}_k^{\mathrm{T}}. \tag{9}$$

Smoothed residuals:

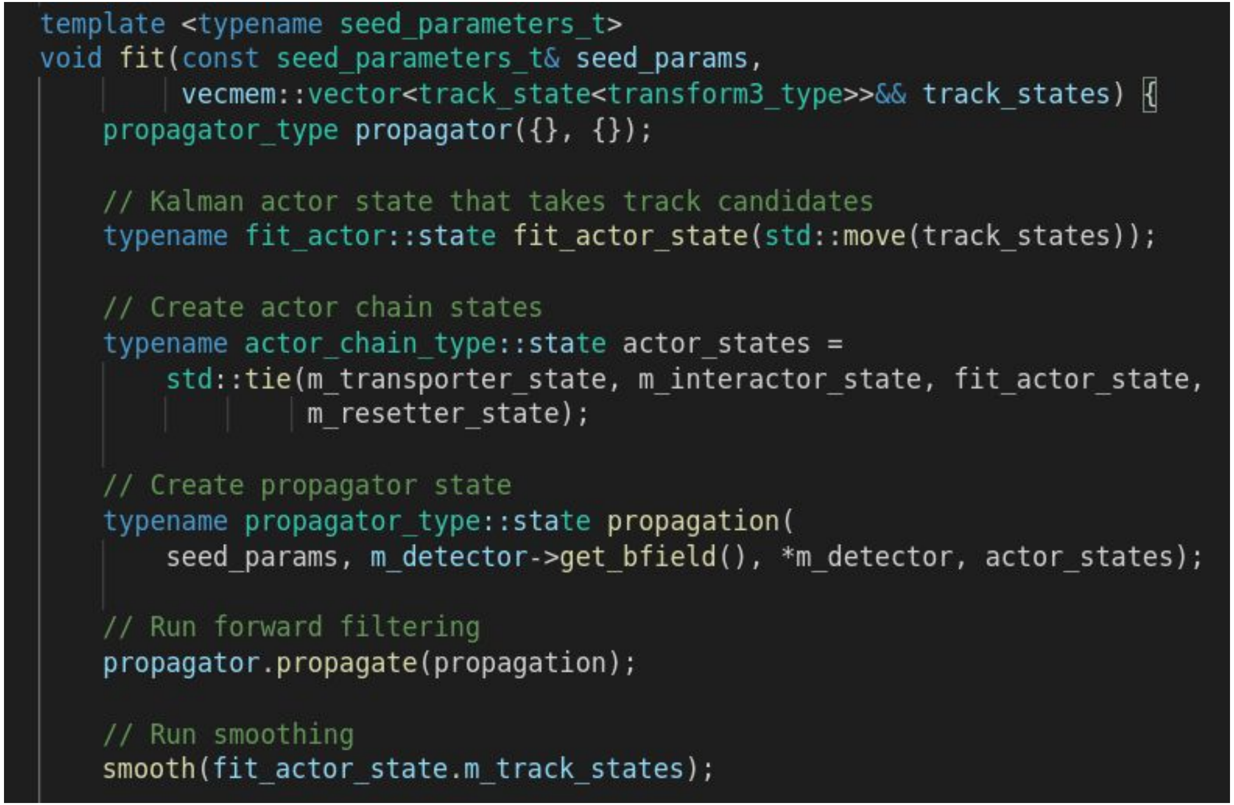
$$\boldsymbol{r}_k^n = \boldsymbol{r}_k - \mathbf{H}_k\left(\boldsymbol{x}_k^n - \boldsymbol{x}_k\right) = \boldsymbol{m}_k - \mathbf{H}_k\boldsymbol{x}_k^n.$$

Covariance matrix of smoothed residuals:

$$\mathbf{R}_k^n = \mathbf{R}_k - \mathbf{H}_k\mathbf{A}_k\left(\mathbf{C}_{k+1}^n - \mathbf{C}_{k+1}^k\right)\mathbf{A}_k^{\mathrm{T}}\mathbf{H}_k^{\mathrm{T}} = \mathbf{V}_k - \mathbf{H}_k\mathbf{C}_k^n\mathbf{H}_k^{\mathrm{T}}.$$

from R.Frühwirth

Fitting function implementation



```
template <typename seed_parameters_t>
void fit(const seed_parameters_t& seed_params,
         vecmem::vector<track_state<transform3_type>>&& track_states) {
    propagator_type propagator({}, {});

    // Kalman actor state that takes track candidates
    typename fit_actor::state fit_actor_state(std::move(track_states));

    // Create actor chain states
    typename actor_chain_type::state actor_states =
        std::tie(m_transporter_state, m_interactor_state, fit_actor_state,
                 m_resetter_state);

    // Create propagator state
    typename propagator_type::state propagation(
        seed_params, m_detector->get_bfield(), *m_detector, actor_states);

    // Run forward filtering
    propagator.propagate(propagation);

    // Run smoothing
    smooth(fit_actor_state.m_track_states);
```