筑波大学大学院博士前期課程

数理物質科学研究科修士論文

リニアコライダー実験における
中央飛跡検出器の要求性能の検討

山口　敦史
(電子・物理工学専攻)

指導教官　浅野　侑三　印

Contents

1 序論 **1**

1.1 はじめに 1

1.2 リニアコライダー測定器の概要と飛跡検出器の役割 3

1.2.1 全体設計 3

1.2.2 測定器の構成 3

1.3 中央飛跡検出器への要請 6

1.3.1 物理からの要請 6

1.3.2 加速器からの要請 10

1.4 中央飛跡検出器の基本設計 11

1.4.1 中央飛跡検出器の幾何学的構造 11

1.4.2 要求される局所的性能 11

1.4.3 シミュレータの必要性と本研究の目標 13

2 Extended Kalman Filter **15**

2.1 概要 15

2.2 Kalman Filter の原理 16

2.2.1 問題の説明 16

2.2.2 Prediction: 次の測定点の推定 18

2.2.3 Filtering: 現在点での見積もりの最適化 19

2.2.4 Smoothing: 過ぎた点の再評価 22

2.2.5 Inverse Kalman Filter: 測定点の除外 24

2.3 Track Fitting への Kalman Filter の応用 25

2.3.1 Kalman Filter の基本的な式 25

2.3.2 系の方程式: $\boldsymbol{f}_{k-1}$、$\boldsymbol{F}_{k-1}$ と $\boldsymbol{Q}_{k-1}$ 26

2.3.3 Measurement Equation: $\boldsymbol{h}_k$, $\boldsymbol{H}_k$, and $\boldsymbol{V}_k$ 32

2.4 C++による実装 34

2.4.1 設計概念 34

2.4.2 KalLib: Kalman Filter のための基底クラス 34

2.4.3 Kaltracklib: Kalman-Filter ベースの Track Fitting ライブラリ 37

2.4.4 GeomLib: ジオメトリクラスライブラリ 43
2.5 hybrid パッケージの実装 51
2.5.1 TPC, IT, VTX (円筒) 51
2.5.2 Forward IT (平面) 52
2.6 hybrid パッケージを用いた解析 52
2.6.1 エネルギー損失補正の効果 53
2.6.2 運動量分解能 53
2.6.3 衝突点分解能 56

3 JUPITER と Satellites の枠組み **59**
3.1 シミュレータの構造と役割分担 59
3.2 Monte-Carlo Truth 生成プログラム JUPITER 60
3.2.1 特長 60
3.2.2 ベースクラス構造 60
3.2.3 TPC の実装 64
3.3 イベント再構成プログラム URANUS とそのシミュレータ Satellites 64
3.3.1 TPC におけるイベント再構成と URANUS 64
3.3.2 イベント再構成シミュレータ Satellites の全体構造 65
3.3.3 IO(Input/Output module set) 66
3.3.4 METIS(Monte-Carlo Exacthit To Intermediate Simulated output) 66
3.3.5 LEDA(Library Extension for Data Analysis) 67
3.3.6 解析 67

4 結論 **74**
4.1 本研究の成果 74
4.2 今後の方向 75

Chapter 1

序論

1.1 はじめに

高エネルギー物理学とは、物質の究極の構成要素を探求し、その間に働く相互作用を解明することを目的とする学問である。高エネルギー物理学発展の歴史は、その主要な研究手段である高エネルギー加速器の進展とともにあった。新しいより高いエネルギーの加速器での衝突実験は、より小さな領域の探索を可能とし、我々の極微の世界に対する理解を深めてきたのである。現在我々は、標準理論に集約される世界像、すなわち、「自然がクォークとレプトンと呼ばれる少数の物質粒子と、その間の相互作用を媒介するゲージ粒子からなり、それらがゲージ対称性で関係づけられている」とする世界像 (ゲージ原理) を手にするに至っている。この標準理論は、現在までの全ての実験事実を見事に説明し、その破れを示す確かな証拠はこれまでの所見つかっていない[1]。しかしながら、標準理論を形作るもう一つの重要な要素である自発的対称性の破れを引き起こす実体、つまり、この標準理論にあって、ゲージ粒子や物質粒子の質量を生み出すなくてはならない粒子、すなわちヒッグス粒子は未だ未発見のままである。昨今話題となっているニュートリノ振動や、B 中間子系での CP 非保存は、いずれも物質粒子 (クォークまたはレプトン) とヒッグス粒子との湯川相互作用に起因すると考えられるので、その本質的な理解は、自発的対称性の破れを引き起こす実体の解明を抜きにしてありえない。現在の高エネルギー物理学の最重要課題は、このヒッグス粒子の発見とその性質の詳細な研究、さらには標準理論の枠組みを本質的に越える理論 (超対称性、余次元、大統一など) の実験的な手がかりを得ることである。標準理論の構造を詳細に調べることにより、その手がかりが TeV 領域にあることが分かっている。これが、LHC を初めとする TeV 領域を調べるための将来加速器の主な建設動機である。

LHC では、広いエネルギー領域を調べる事ができるという大きな利点が有るが、構造を持った陽子・陽子の衝突であるため、反応が複雑で、素過程の精密な測定は難しい。一方、電子・陽電子線形衝突型加速器では、構造を持たない粒子同士の衝突なので、始状態が明快に分かっており、反応も素過程そのものなので、精密測定に理想的である。これまでの実験から、間接的ではあるが、標準理論のヒッグス粒子の質量について、95% の信頼度で、約 200 GeV 以下という上限値がえられており、重心系エネルギーで 400 GeV

[1]近年のニュートリノ振動の発見は、ニュートリノが有限の質量を持つ事を示したが、これは、標準理論の基本的枠組みを崩す事なく簡単に取り込む事ができる。標準理論の根幹をなす基本原理、すなわちゲージ原理はいささかもゆらいでいないのである。

程度の電子・陽電子線形衝突型加速器でその発見、研究ができると期待されるので、世界的に、建設に向けて気運が盛り上がっている。我が国でも、諸外国と共同で、将来計画として電子・陽電子線形衝突加速器である国際リニアコライダー計画 (ILC) を推進しつつあり、現在加速器、測定器の両方で勢力的に開発研究が行われている。

既に述べたように、リニアコライダー実験で最も重要な目標の一つに、ヒッグス粒子の発見と研究が挙げられる。特に、$e^+e^- \to H^0Z^0$ の反応で Z^0 の崩壊から生じるレプトン対の反跳質量分布の測定によるヒッグス粒子の測定では、その質量、崩壊巾の最も精度の高い測定を可能にするのみならず、ヒッグス粒子の崩壊モードによらない探索を可能とするので、生成断面積の崩壊分岐比によらない絶対測定が可能となる。また、ヒッグス粒子が検出不可能な粒子に崩壊した場合にも使える探索法を提供する。このレプトン対の反跳質量分布の測定で最も重要な役割を果たすのが中央飛跡検出器である。

レプトン対の反跳質量の分解能は、もともと加速中に生じるビームエネルギーの広がりと中央飛跡検出器によるレプトン対の運動量分解能とで決まる。加速器の性能を最大限に活かすためには、中央飛跡検出器によるレプトン対の運動量分解能の反跳質量分解能に対する寄与は、ビームのエネルギーの広がりの寄与に比べて無視できる程度でなくてはならない。後で詳しく述べるように、この要求から、$\sigma_{p_T}/p_T \sim 10^{-4} \times p_T(\mathrm{GeV})$ の運動量分解能を持った高性能の中央飛跡検出器が必要となる。

また、高エネルギー電子・陽電子線形衝突型加速器実験の著しい特徴として、複数のジェットに崩壊する弱い相互作用のゲージ粒子 (W/Z)、トップクォーク (t)、さらにはヒッグス (H) などの重いパートンを、ジェット不変質量法によって同定し、素過程を基本粒子のレベルで再構成する可能性 (ファインマン図が見えてくる) があげられるが、そのためには、ジェット中の近接する荷電粒子飛跡を高効率で分離し、各々の運動量を高精度で測定する事、また、こうして検出された荷電粒子飛跡をカロリメータのクラスターと一対一対応させる事が必要となる。

一方、将来の電子・陽電子線形衝突型加速器実験では、ビームをナノメータサイズにまで収束させ衝突させるので、衝突するビーム同士が強い電磁力を及ぼし合い、今までにない種類のバックグラウンドが発生する可能性もある。測定器はこうしたバックグラウンドに十分耐えうるものでなくてはならない。

こうした物理の要求を踏まえ、また、加速器からの要請を加味した、中央飛跡検出器の全体設計の完成が我々JLC-CDC グループの開発研究の最終目標である。JLC-CDC グループはこれまでジェットチェンバー型の中央飛跡検出器の開発を進めてきたが、近年、加速器技術が旧来の常伝導方式から超伝導方式へ変わったことで time-stamping 性能への要求がゆるくなったこと、また GEM、MicroMEGAS 等の MPGD の開発研究の進展で、これまでの MWPC 読み出し方式の TPC の問題点 (近接ヒット分離、セクター境界の問題) が大幅に改善する見通しが得られてきたことで、開発研究の中心をより多くの測定点の数を実現しやすい TPC に移すことにした。一般に、開発研究は、要素技術の開発と、それを組み合わせたシステム開発とに大別される。TPC の開発研究の場合、前者は、パッドあたりの位置分解能や、近接ヒット分離能といった測定器の局所的な性質にかかわる要求性能の実現可能性を見極めるためのハードウエア開発研究が中心となる。一方、後者のシステム開発は、TPC 全体としての性能評価を必要とするため、実機規模のプロトタイプ建設以前の段階においては、要素技術の開発研究の結果に基づいた、シミュレーション主体のものとなる。TPC の全体設計は、前者に対応する検出器要素 (例えば読み出しパッド) の設計、後者に対応するそれらの配置 (例えばレイヤー数、全体としての大きさなど) の設計からなり、各々、基本仕様 (検出器の幾何学的構造を決めるパラメータや、例えばパッドあたりの位置分解能など) を決める基本設計 (Conceptual Design)、またそれを実際どう具体化するかを決める技術設計 (Engineering Design) の段階を踏むことにな

る。そこで、要求性能を満たす TPC の基本設計を行うための測定器の応答及び事象再構成の過程をシミュレートするフルシミュレータを開発する必要が出てきた。

本研究は、このシミュレータ開発の一環として行われたものである。すでに述べたようにリニアコライダーでは、高運動量分解能、高近接飛跡分離能、高バーテックス分解能及び高検出率が必要となる。そのため、TPC から比較的密度の高い物質層を越えてバーテックス検出器のトラックと連結しトラックパラメータの測定制度を向上させること、複数のトラックを組み合わせ、崩壊点や衝突点の位置測定制度を向上させることが課題となる。Kalman Filter はこのような問題に有用な手法である。本研究では、Kalman Filter をこれらの問題に応用するための基本クラスを作成し、それをもとに飛跡再構成プログラムを開発し、解析を行う。また、Geant4 に基づく測定器シミュレーションの枠組み (JUPITER、URANUS/Satellites) について述べ、前述の飛跡再構成プログラムを振るシミュレータ上に再現し、解析結果の比較を行う。

1.2 リニアコライダー測定器の概要と飛跡検出器の役割

1.2.1 全体設計

リニアコライダーの衝突点は当面一ケ所であるため、そこに設置される測定器は、想定されるあらゆる物理だけでなく、予想外の物理に対しても十分対応可能な高性能かつ汎用の物でなければならない。具体的にはまず W 粒子や Z 粒子のジェットを用いた再構成が精度よく出来なければならない。また、ヒッグス粒子の発見及び研究のため、非常に良い質量分解能が要求される。またトップクォークの研究やヒッグス粒子に対するバックグラウンドを抑えるため、b ジェット識別が効率良く行える必要がある。さらには、超対称粒子探索のため十分広い立体角を隙間なく覆う測定器でなければならない。これらの条件をシミュレーションを行って検討した結果、「標準測定器」として図 1.1 に示すような構成の測定器を考え、表 1.1 に示すような性能を達成することを目標としている。

1.2.2 測定器の構成

バーテックス検出器

衝突点の極めて近傍で、荷電粒子の飛跡を精密に測定する検出器で、B 中間子や D 中間子の崩壊点を測定して、b クォークや c クォークの同定をする役割をしている。通常バーテックス検出器に使用されるシリコンストリップ型ではなく、2 次元的に読み出し可能なピクセル型の Si-CCD を使用する。多量のバックグランド中でも十分な機能を果たす為に、4 層の CCD 検出器より構成され、それぞれビーム軸から 2.4、3.6、4.8、6.0 cm の位置に設置される。総数 3.2×10^8 個の 25 μm $\times$ 25 μm のピクセルで構成され空間分解能は 4 μm である。

中央飛跡検出器

寿命の長い荷電粒子 (電子、ミュー粒子、荷電 π 中間子、荷電 K 中間子、陽子) の飛跡を検出する装置で、超電導磁石による磁場によって曲げられた荷電粒子の飛跡の曲率半径からその粒子の運動量を知ることができる。詳細は後述。

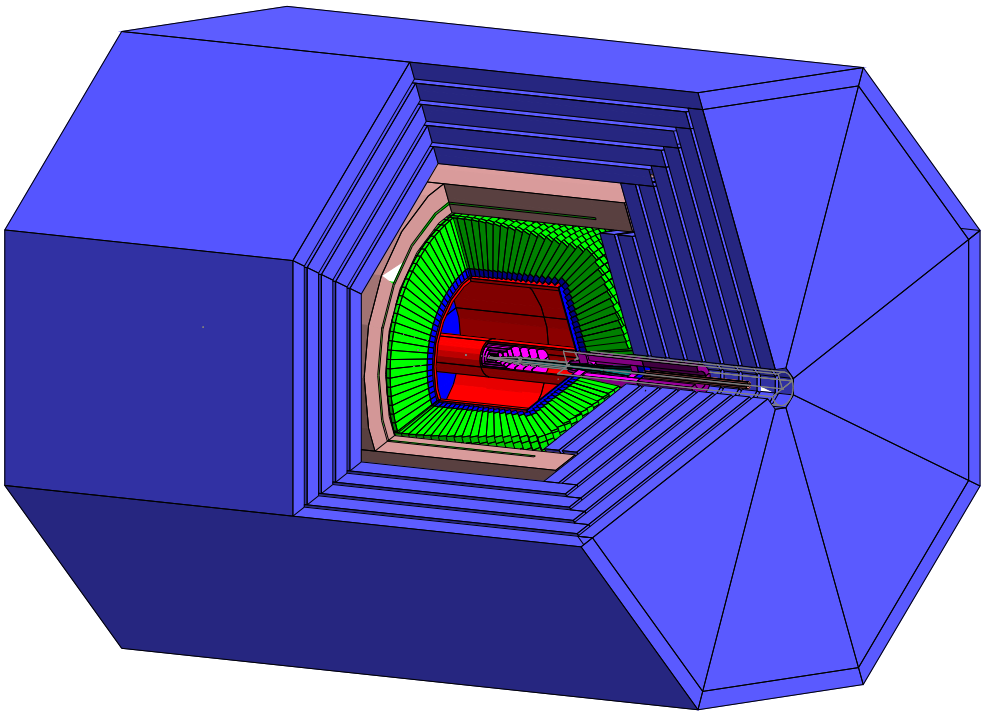

Figure 1.1: リニアコライダー測定器の概略図

Detector		Configuration	Performances	Channels and Data Size
PM (3D Active Pixel)		θ = 11 - 48mrad (r=2-8.5cm) 300μm-thick x 2 layers pixel size=100μm	Under Study	Number of pixels = 8.6M Readout channel = 156ch Data size = 12k bytes/sec
LM (W/Si)		θ = 50-150mrad 43Xo x 16samplings Nr = 32, $N\phi$ = 16	Under Study	Number of pads = 16.4k Readout channel = 128ch Data size = 3.3k bytes/train
AM (W/Si)		θ = 150-200mrad 23Xo x 8samplings Nr = 10, $N\phi$ = 32	Under Study	Number of pads = 5.1k Readout channel = 16 Data size = 5.1k bytes/train
FT		TBD	Unknown	
VTX (CCD)		cosθ < 0.90 pixel size=25μm, thickness=300μm 4 layers at r = 2.4, 3.6, 4.8, 6.0cm	σ = 4.0μm $\delta^2=7^2+(20/p)^2/\sin^3\theta$ [μm] ε_b=50% @ purity=93%	Number of pixels = 320M Readout channel = 2.4k Data size = 1.4M bytes/train
IT (Si-strip)		cosθ < 0.90 strip width=100μm, thickness=300μm 5 layers at r = 9, 16, 23, 30, 37cm	σ = 40μm Tracking Performance Under Study	Number of strips = 522k Readout channel = 1.0k Data size = under study
CDC (Mini-jet)	common	cosθ < 0.70 (full sample) cosθ < 0.95 (1/5 samples)	σ_z = 1mm 2-track separation = 2mm	200MHz FADC depth = 1k words
	2Tesla	r = 45 - 230cm, L = 460cm N_{sample} = 80	σ_x =100μm $\sigma_{Pt}/Pt = 1\times10^{-4}Pt + 0.1\%$	Readout channel = 13k Data size = 5.2M bytes/train
	3Tesla	r = 45 - 155cm, L = 310cm N_{sample} = 50	σ_x =85μm $\sigma_{Pt}/Pt = 3\times10^{-4}Pt + 0.1\%$	Readout channel = 8.1k Data size = 3.3M bytes/train
Trackers Combined			$\sigma_{Pt}/Pt = 1\times10^{-4}Pt + 0.1\%$	
CAL (Pb/Sci)	common	EM = 27Xo (3sections) HAD = 6.5λo (4sections) $\Delta\theta,\phi$ = 24mrad (EM), 72mrad (HAD)	$\sigma/E=15\%/\sqrt{E}+1\%$ (EM) $\sigma/E=40\%/\sqrt{E}+2\%$ (Had) e/π ID = 1/1000	Number of cells = 144k Readout channel = 5k Data size = 3k bytes/train
	2Tesla	cosθ < 0.985 (full thickness) r = 250 - 400 cm, z = +- 290cm		
	3Tesla	cosθ < 0.966 (full thickness) r = 160 - 340 cm, z = +- 190cm		
SHmax		scin.strip (1cm-wide) or Si-pad (1cm x 1cm)	σ = 3mm/$\sqrt{E}$	Readout channel = 5k Data size = 40k bytes/train
MU (SWDC/RPC/TGC)		cosθ <0.998 6 SuperLayers	σ = 0.5mm Muon ID under study	Readout channel = 10k
Yoke	2Tesla	r = 5.5m - 7.5m, Z = 5.0m - 7.9m		
	3Tesla	r = 4.5m - 7.0m, Z = 3.9m - 6.5m		

Table 1.1: リニアコライダー測定器の諸性能

カロリメータ

飛跡を残さない中性粒子 (光子、中性子、K_L^0 など) のエネルギーを測定する役割をもつ。カロリメータはビーム軸の周りの円筒状のバレル部とそれに蓋をするような形の端部より成り $|\cos\theta| < 0.99$ の領域を覆っている。検出部分は鉛とプラスチックシンチレーターの多層サンドイッチ構造をしていて 1GeV の電子や光子に対するエネルギー分解能は 15%、1GeV のハドロン粒子に対しては 40%である。

ミュー粒子検出器

カロリメータで吸収されることなく全ての測定器を通り抜けるミューオンを検出する役割をもつ。このミュー粒子検出器は運動量の測定ではなくミュー粒子の識別に用いられるので空間分解能は 500 μm 程度でよい。この検出器は、6 層 (Supre-Layer) からなり、1 つの層 (Supre-Layer) には単線のワイヤーチェンバーから成る 4 つの層構造になっていて飛跡の方向を求めることができる。またカロリメータ及び鉄中でのエネルギー損失のため識別可能なミュー粒子の運動量は 3.5GeV 以上である。

1.3 中央飛跡検出器への要請

1.3.1 物理からの要請

多くの実験による精力的な探索にも係わらず、標準模型の要であるヒッグス粒子は未だ見つかっていない。その探索と性質の研究はリニアコライダーにおける物理の最も重要な課題の一つである。標準模型では、ヒッグス粒子の質量はパラメーターにすぎない。従って、その予言は標準模型を越える理論でのみ可能である。軽い ($\lesssim$ 200 GeV) ヒッグス粒子の存在は大統一、大砂漠を基礎とする模型の一般的な帰結である。これに対し、テクニカラーなどの複合ヒッグス模型ではヒッグス粒子は重くて良い。前者のシナリオでは、自然さの問題を考えれば、TeV 以下に多くの超対称粒子が存在し得る。一方、後者のシナリオでは、その背後にある新しい力学の全容を明らかにするには、TeV を越えたエネルギーが必要となる。この意味で、リニアコライダーにおける軽いヒッグス粒子の探索は、高エネルギー物理の今後の方向を決定する分岐点となるであろう。既に述べたように、標準理論の枠組みの中で、これまでに得られたデータをフィットすると、ヒッグス粒子の質量に対して約 200 GeV の上限値がえられる。これは、データが軽いヒッグスのシナリオを指示していると見る事ができるが、重いヒッグスの可能性はまだ残されており、最終的な決着は、直接探索の結果を待たねばならない。以下にヒッグス粒子探索からの中央飛跡検出器の性能に対する要求をまとめる。

ヒッグス粒子の探索法

高エネルギー電子陽電子衝突反応で、標準模型のヒッグス粒子を作る反応としては、(1) $e^+e^- \to H^0_{SM}Z^0$、(2) $e^+e^- \to \nu\bar{\nu}H^0_{SM}$、(3) $e^+e^- \to e^+e^-H^0_{SM}$ 反応などがある。このうち、(2) と (3) の反応は 1 TeV 以上の高エネルギーで断面積が大きくなるので特に重いヒッグスの探索の場合に適している。一方、(1) の反応は比較的軽いヒッグスの場合に適しており、LEP での探索などで利用されている。第一期のエネルギー領域でも、主に (1) の反応を利用して、ヒッグスの探索を行なうことになる。

一方、ヒッグス粒子とフェルミオン、ウィークボソンとの結合は質量に比例し、よって、その崩壊の部分巾は質量の自乗に比例する。従って、ヒッグス粒子は、質量が半分以下の粒子 ($< \frac{1}{2}M_{H^0_{SM}}$) のうち最も重い粒子への崩壊巾が最も大きい。実際 140 GeV 以下では確かに $H^0_{SM} \to b\bar{b}$ モードの崩壊の分岐比が最大である。しかしながら、ウイークボソンへの崩壊巾と b クォークへの崩壊巾は 3 桁近く異なるので、140 GeV 以上では、$H^0_{SM} \to W^*W$ モードの崩壊の分岐比が最も高くなる。超対称性理論が予言するような軽いヒッグス (140 GeV 以下) の場合は、主として $H^0_{SM} \to b\bar{b}$ モードの崩壊を探索することになる。

以上のことから、$e^+e^- \to H^0_{SM}Z^0$ 反応によるヒッグス粒子生成事象の検出は、終状態として、Z^0 の崩壊モードにより (1)$\nu\bar{\nu}b\bar{b}$、(2)$l^+l^-b\bar{b}$、および (3)$q\bar{q}b\bar{b}$ の三つの型にに分類出来る。

典型的な事象の例を図 1.2 に示す。

このようなヒッグス粒子は次の二つの方法で探索できる。まず、上記三つの場合によって、2 ジェット、レプトン対あるいは四次元運動量欠損の不変質量が Z 粒子のそれと一致するという要求をすると、ヒッグス粒子は残りの b クォークによる 2 ジェット系の不変質量分布にピークとなって現れる。このピークを探すことにより発見出来る、2 ジェット不変質量分布を用いた探索。もう一つは、$l^+l^-b\bar{b}$ モードを使い、検出した l^+l^- から l^+l^- 以外の系の質量を求めることにより、ヒッグスの崩壊モードに無関係に探索を行なうことが出来る、レプトン対に対する質量欠損を用いた探索である。

飛跡検出器への性能要請

2 ジェット不変質量分布を用いた探索からの要請

電子陽電子衝突過程は全重心系エネルギーが反応の素過程に使用されるために、終状態の識別が容易であり、確実な新粒子探索や精密実験ができるという特徴がある。これに加えて、リニアコライダーのエネルギー領域ではジェットのエネルギー集中がますます顕著になり、また Calorimeter のエネルギー分解能も良くなるので、トップ以外のクォークが鋭いジェットとして見えるようになるのみならず、ジェット不変質量法によるゲージボソンやトップクォークの再構成が可能となる。つまり、リニアコライダーでは、ファイマン図を見るが如く、反応の終状態を基本粒子すなわちレプトン、クォーク、ゲージボソンの単位で捉えることができるようになるのである。これは、全く新しい加速器実験の幕開けである。この特筆すべき可能性を現実のものとし、加速器の潜在能力を 100% 引き出すためには、終状態に生成されるニュートリノを除く全ての粒子を精度よく検出する、高性能の測定器が必要である。

そこで、W ボソンと Z ボソンは、主要な崩壊モードであるクォークジェットへの崩壊において識別可能であることを要求する。Z ボソンは W ボソンより 10 GeV 程度重く、それぞれ 2.5 GeV と 2.0 GeV の崩壊巾をもつ。従って、W と Z が 2 ジェット不変質量で分離可能であるためには、その分解能はこれらの崩壊巾と同程度でなければならない。

測定器の性能を最大限に生かしてジェット不変質量の分解能を出来る限り改善するためには、高分解能のハドロン Calorimeter を建設するのみならず、中央飛跡検出器から得られる運動量情報を利用することも重要である。特に、前節で述べたようなヒッグス粒子のレプトン対質量欠損法による測定から要求されるような高い分解能の中央飛跡検出器がある場合には、荷電ハドロン粒子のエネルギーとして Calorimeter でなく中央飛跡検出器の情報を用いた方が測定精度が向上する。すなわち、荷電粒子に関しては中央飛跡検出器、中性粒子に関しては Calorimeter と言うように、役割を分担できるのが理想である。この場合、Calorimeter のクラスターと中央飛跡検出器で検出された荷電粒子の飛跡とを対応 (クラスター・トラックマッチング)

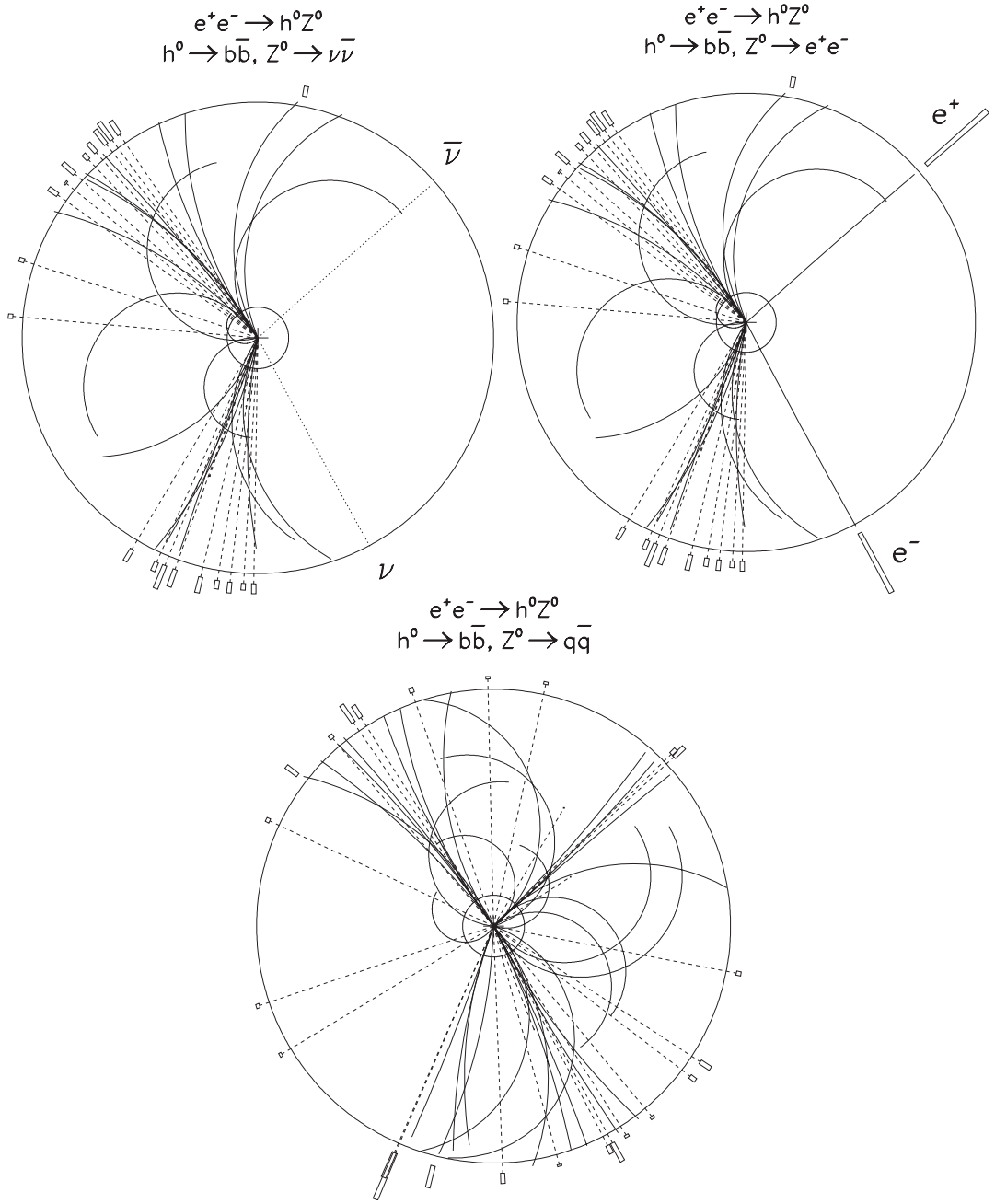

Figure 1.2: 典型的な $e^+e^- \to H^0_{SM}Z^0$ 事象の例。$m_{H_{SM}} = 120$ GeV、$\sqrt{s} = 300$ GeV とした。(a) $H^0_{SM} \to b\bar{b}$ 、 $Z^0 \to \nu\bar{\nu}$、(b) $H^0_{SM} \to b\bar{b}$ 、 $Z^0 \to e^+e^-$、(c) $H^0_{SM} \to b\bar{b}$ 、 $Z^0 \to q\bar{q}$。図 (a) および (c) では、実線は 2 テスラの磁場中におかれた内径 0.3 m、外径 2.3 m の飛跡検出器により測定される荷電粒子の軌跡を、点線は光子を示す。外側の箱は電磁 Calorimeter を表し箱の大きさはエネルギーに対応している。

させ、対応しないものに関してだけ中性粒子として Calorimeter 情報を使うことになる。Calorimeter には位置測定用のシリコンパッド (1cm × 1cm) が装着されているが、中央飛跡検出器で検出された荷電粒子の飛跡をこのパッドの位置まで延長した際に間違いなく対応できることが必要となる。

また、ジェット中の荷電粒子に対する高分解能の実現には、複数の飛跡が重なり合って分離できない場合が生ずるという困難がある。特に、飛跡の一部の測定点が使いものにならなくなると、実質的な測定点の数と測定領域の大きさが減少し、運動量分解能を悪化させる。近接した 2 本の飛跡の分離性能に対する要求は、ジェットの混み具合がエネルギーによるので、エネルギーとともに変わるが、リニアコライダーでは、2 mm 程度の距離であれば分離できることが望ましい。

<u>レプトン対に対する質量欠損法を用いた探索からの要請</u>

レプトン対に対する質量欠損法とは、初期状態の重心エネルギー (E_{CM}) が良く分かっているとき、終状態 2 体のうちの一方 (今の場合 Z ボソン) のエネルギーや運動量 (E_Z, p_Z) から、他方 (ヒッグス粒子) の質量 (M_h) をエネルギー運動量保存則より計算する方法である。すなわち、

$$\begin{aligned} M_h^2 &= (E_{CM} - E_Z)^2 - \vec{p}_Z^{\,2} \\ &= E_{CM}^2 - 2E_{CM}(|\vec{p}_{\ell_1}| + |\vec{p}_{\ell_2}|) + 2|\vec{p}_{\ell_1}||\vec{p}_{\ell_2}|(1 - \cos\theta) \end{aligned} \tag{1.1}$$

ここで、

$$\vec{p}_z = \vec{p}_{\ell_1} + \vec{p}_{\ell_2}$$

で、θ は実験室系でのレプトン対の運動量間の角度である。従って、ヒッグス質量の分解能 (ΔM_h) は、(1.1) 式より角度の誤差を無視すれば

$$(\Delta M_h^2)^2 = \left(\frac{\partial M_h^2}{\partial |\vec{p}_{\ell_1}|}\right)^2 (\Delta|\vec{p}_{\ell_1}|)^2 + \left(\frac{\partial M_h^2}{\partial |\vec{p}_{\ell_2}|}\right)^2 (\Delta|\vec{p}_{\ell_2}|)^2$$

ただし、

$$\begin{aligned} \frac{\partial M_h^2}{\partial |\vec{p}_{\ell_1}|} &= -2\left(E_{CM} - 2|\vec{p}_{\ell_2}|\sin^2\frac{\theta}{2}\right) \\ \frac{\partial M_h^2}{\partial |\vec{p}_{\ell_2}|} &= -2\left(E_{CM} - 2|\vec{p}_{\ell_1}|\sin^2\frac{\theta}{2}\right) \end{aligned}$$

となる。

ここで、しきい値近くでは $\theta \simeq 180^\circ$ かつ $|\vec{p}_{\ell_1}| \simeq |\vec{p}_{\ell_2}|$ である事を考慮すれば、ヒッグス質量の分解能 (ΔM_h) はレプトンの運動量分解能 ($\Delta|\vec{p}_\ell|$) に比例し、

$$\Delta M_h \simeq \sqrt{2}\frac{E_{CM} - 2|\vec{p}_\ell|}{M_h} \cdot \Delta|\vec{p}_\ell| \tag{1.2}$$

と表すことができる。

一方、ビームエネルギーの広がり ($\Delta E = \Delta E_{CM}/2$) は、0.2% 程度まで小さくできると期待でき、この広がりのヒッグス質量の分解能への寄与は

$$\Delta M_h = \frac{E_{CM} - (|\vec{p}_{\ell_1}| + |\vec{p}_{\ell_2}|)}{M_h} \cdot \Delta E_{CM}$$

$$\simeq \frac{E_{CM} - 2|\vec{p}_\ell|}{M_h} \cdot \Delta E_{CM}$$

で近似できる。これは、$M_h = 100$ GeV、$E_{CM} = 250$ GeV とすると、$\Delta M_h \simeq 0.75$ GeV の寄与である。運動量分解能のから来る誤差が無視できるためには、(1.2) 式で得られる誤差がその 1/2 程度以下でなくてはならない。すなわち、50 GeV に対して 0.4% の運動量分解能が必要となる。

1.3.2 加速器からの要請

加速器

円型加速器では、電子と陽電子は円型の軌道を何周も回る。そのため、比較的弱い加速装置でも、粒子が軌道を何周もする間に、少しずつエネルギーを大きくすることができる。また、同じビームが何度も衝突するので、比較的小さなビーム強度でも反応確率を高くすることが出来る。例えばつくばのトリスタンや欧州の LEP などは、この方式の加速器である。しかしこの方式では、高エネルギーの電子が曲げられるとき放射光を発生してエネルギーを失うため、到達できるエネルギーには限界があり、LEP-II 以上のエネルギー、すなわちで 200 GeV を超える重心系エネルギーを実現するのは困難である。一方、線形加速器は、前段部のごく低エネルギーの部分をのぞいては、曲線部を持たない。このため、放射光によるエネルギー損失は原理的にない。従って、これまで円型加速器では、到達できなかった高エネルギーを実現できる。しかし、直線である為、電子や陽電子は加速部分を 1 度しか通らないし、それらは 1 回の衝突にしか使えない。そのため

- これまでの加速器の約 10 倍、強く加速すること、
- 非常に多数のバンチ (電子、陽電子の塊) を次々と発生させ、それを加速すること、
- ビームを衝突させるときに非常に小さく (240 nm × 3 nm) 圧縮すること、

が必要である。現在リニアコライダーでは、図 1.3 で示されているような、加速器が考えられている。また、パラメーターは、表 1.2 に示す。

Figure 1.3: リニアコライダーの加速器の概略図

衝突エネルギー	500 GeV	1 TeV
ルミノシティー	$2.0 \times 10^{34} cm^{-2} s^{-1}$	$2.8 \times 10^{34} cm^{-2} s^{-1}$
バンチあたりの粒子数	2.0×10^{10} 個	2.0×10^{10} 個
バンチ数	2820 バンチ	2820 バンチ
バンチ間隔	295.4 ns	295.4 ns
加速繰り返し	5 Hz	5 Hz
主線形加速器加速勾配	35 MV/m	35 MV/m
衝突点ビームサイズ	655 nm×5.7 nm	544 nm×3.5 nm
ビームパワー	11.3 MW/beam	22.6 MW/beam

Table 1.2: リニアコライダー加速器のビーム主要諸元

加速器からの要請

以上に述べたように、電子・陽電子線形衝突型加速器には、今までの円形加速器にない特徴があり、これが中央飛跡検出器に今までにない制約を与えることになる。特に、ビームをナノメータ程度まで絞り込むための最終収束電磁石系は、衝突点を囲む測定器システムの中に入り込み、その一部として扱われる。ナノメータのビームを安定に衝突させるためには、最終収束系にもナノメータの精度の安定性が要求される。特に電子側の電磁石と陽電子側の電磁石の相対位置がナノメータの精度で制御されていなくてはならない。そのため、現在の リニアコライダーの最終収束系の設計では、電子側の最終収束電磁石と陽電子側の最終収束電磁石は同一の CFRP 製の円筒 (サポートチューブ) の中に納められることになっている。このサポートチューブの半径は約 40 cm なので、中央飛跡検出器の内径はそれ以上となる。

1.4 中央飛跡検出器の基本設計

1.4.1 中央飛跡検出器の幾何学的構造

1.4.2 要求される局所的性能

チェンバーの全体構造が決まったので、次にこの飛跡検出器全体への性能要求を、達成すべき局所的性能に焼き直す。

検出器の性能の下限は物理的要請から決まってしまうので、ここで考慮すべき問題は、その性能を保証するための測定精度を達成出来るか否か、である。計算の結果、その測定精度が、現在の技術力からみて、明らかに達成不可能なものであれば、設定された基礎デザインに問題がある。ただちに再度物理要請に立ち戻り、基礎デザインを練り直さなければならない[2]。

一方、要求される測定精度が達成可能と見積もられる場合には、その測定精度を実機で達成出来るか否かを実験によって確かめなくてはならない。ここに至って、テストチェンバー等を使用した R&D 実験が開

[2]実際には、基礎デザインを練る段階で当然このことは考慮されている。基礎デザインの設定は物理的要求と実現可能な測定精度を天秤にかけながら行われるものであり、基礎デザインが完成した時点で、測定精度に対する境界条件は満たされているべきである。

始される。R&D の結果、必要な測定精度が満たせないことが判明した場合には、基礎デザインにたち戻って再度デザインの検討を行わなければならない。

R&D 項目及び R&D 現状については次節に述べることとし、ここでは基礎デザインから予想される飛跡検出器の性能と、その性能を達成するために必要な測定精度について列挙する。

運動量分解能

運動量分解能に対する最も厳しい要求は、$e^+e^- \to ZH$ の反応でつくられた Z がレプトン対に崩壊した時の、レプトン対から計算される質量欠損の分解能に対する要求から決まる。標準模型で期待されるヒッグスの崩壊幅は MeV のオーダーであるので、検出器側も出来る限りその大きさに近付きたいところである。実際には、ビームエネルギーの幅が 200MeV 程度存在するので、検出器側では少なくともこのビームエネルギーの広がりと同程度の運動量分解能を持たねばならない。ヒッグス質量の分解能 ΔM_h はレプトンの運動量分解能 ΔP_ℓ に比例し、$\Delta M_h \simeq 2P_\ell/M_h \cdot \Delta P_\ell$ と表すことができるため、M_h =100 GeV に対する ΔM_h =200 MeV の要求は、P_ℓ ≃50 GeV として、ΔP_ℓ =200 MeV すなわち 50 GeV に対して 0.4% の運動量分解能の要求となる。この運動量分解能が達成されれば、5TeV の荷電粒子に対してその電荷を判別出来ることになる。

横方向の運動量 P_t の逆数 $\kappa = 1/P_t$ に対してその分解能 σ_κ は

$$\sigma_\kappa^2 = (\sigma_\kappa^{meas})^2 + \left(\sigma_\kappa^{MS}\right)^2 \tag{1.3}$$

で与えられる。ここで右辺第一項は位置測定の誤差、第二項はチェンバー ガスでの多重散乱によるものであり、各々

$$\begin{aligned} \sigma_\kappa^{meas} &\simeq \left(\tfrac{\alpha\sigma_x}{Bl^2}\right)\sqrt{\tfrac{720}{n+4}} \\ \sigma_\kappa^{MS} &\simeq \left(\tfrac{\alpha C}{Bl}\right)\sqrt{\tfrac{10}{7}\left(\tfrac{X}{X_0}\right)}\cdot\kappa \end{aligned} \tag{1.4}$$

と書ける。ここで $\alpha = 333.56$ ($\mathrm{cm}\cdot\mathrm{T}\cdot\mathrm{GeV}^{-1}$)、$C = 0.0141$ (GeV)、 L は測定される飛跡の長さ (cm)、B は磁場の強さ (T)、$\left(\frac{X}{X_0}\right)$ は輻射長で表したチェンバー ガスの厚さ、n は測定点の数である。

ここで、B=3Tesla、$\sigma_x = 100\mu$m、n=200、X/X_0=1.1%、l=250cm を選ぶ。すると、σ_κ^2 及び $\left(\frac{\sigma_{p_T}}{p_T}\right)^2$ は、

$$\sigma_\kappa^2 = (3.1\times10^{-5})^2 + (6.6\times10^{-4}\cdot\kappa)^2 \tag{1.5}$$

$$\left(\frac{\sigma_{p_T}}{p_T}\right)^2 = (3.1\times10^{-5}\cdot p_T[GeV])^2 + (6.6\times10^{-4})^2 \tag{1.6}$$

となる。

以上より、R&D 項目としては、ドリフト領域全域にわたって、平均 $\sigma_x = 100\mu$m の空間分解能が保証されれば、運動量分解能に対する飛跡検出器への要求を満たすことができる。

近接飛跡分離能

質の良い飛跡再構成を行うためには 2 本の近接した飛跡を分離出来る分解能が重要である。この性能を 250GeV の W 粒子が崩壊して出来る粒子を用いて調べた。これによるともし 2mm 以下しか離れていない

２個のヒットは区別できないとすれば、最内層においては 5%、最外層においては 1%のヒットが２個のヒットを分離することに失敗することにより失われる。この程度の損失は飛跡再構成にほとんど影響を及ぼさないので、2mm の近接ヒット分離能が達成できればよい。TPC の場合 x-y 平面での近接飛跡分離能は読み出し用パッドでの電荷の広がりで決まる。旧来の MWPC 読み出しでは電荷の広がりがワイヤとパッドの間の距離で決まり (静電気学の問題)、分離能として 1cm 以下の値を達成するのは困難であったが、MPGD 読み出しでは、電荷の広がりが主としてドリフト中の拡散で決まり 2mm は可能と考えられている。

Background

ビーム・ビーム相互作用によるコヒーレント及び非コヒーレントな QED 過程により、一次バックグラウンドとして電子・陽電子が発生する。これらはQ電磁石の前面やマスク等に衝突して二次バックグラウンド光子を発生する。これらの光子のエネルギースペクトルは約 100keV に幅広いピークを持つとともに、500keV には対消滅による鋭いピークが存在する。

バックグラウンドは、磁場、ビーム強度、マスクの形状によるが、現在の設計では $\sqrt{s}$=500GeV において１衝突当たり約 10^4 個の光子が TPC に飛び込むことになる。飛跡再構成に影響を及ぼす可能性がある。

1.4.3 シミュレータの必要性と本研究の目標

以上のように基礎デザインから要求される各測定量への測定精度が決まったところで、これらの測定精度が実際に達成できるか否かを見極めるための R&D が開始された。R&D ではまず、TPC の全体構造に依存しないローカルパラメータを設定し、その後 TPC 全体の構造に直接依存するグローバルパラメータの最適化を行う。ここで、ローカルパラメータの設定は TPC の一部を取り出したテストチェンバーを作成して行うことができるが、グローバルパラメータの最適化のために実機サイズのプロトタイプを作成することは、時間、費用共両面において現実的でない。そのため、ローカルパラメータの設定からグローバルパラメータの最適化に移行する前に、シミュレータによる最適化作業が必要不可欠となるのである。シミュレータの開発段階では、同時にオフラインの解析プログラムの開発が行われる。グローバルパラメータの最適化や、測定精度を求めるためには、TPC の詳細な構造を組み込んだシミュレータが必要である。我々が進めているシミュレータの構成を以下に示す。

1. イベント生成部分 (粒子以前のパートンを粒子に変換)
2. Monte-Carlo Truth 生成部分 (検出器シミュレーション部)
3. イベント解析プログラム (イベント再構成部)
4. 物理解析プログラム (物理解析部)

このうち、1. と 4. は既存のプログラムが存在するので、それを用いる。

2. の部分に関しては、検出器と粒子の反応に関するモンテカルロ・シミュレータである Geant4[4] をベースに開発する。ただし、Geant4 はライブラリ群であり、TPC で開発したシミュレータをやがてバーテックスやカロリーメータにも拡張することを考えると、ライブラリを生のまま用いるのは好ましくない。そこで、リニアコライダー検出器全体を統合して扱う JUPITER と名付けられたシミュレーション・フレー

ムワークを開発した上で、そのフレームワークを用いて TPC 部分の開発を行う。また、Geant4 ライブラリに含まれていないオブジェクトに関しては、そのつど新規開発する。

3. の部分に関しては、リニアコライダーの解析フレームワークである JSF[5] の枠組みを用いて、まず JSF の下位に URANUS/Satellites と名付けられたシミュレータ用のフレームワークを作成する。これは、2. と同様に、他の検出器を将来統合する方向を考慮しているためである。実際の解析部分は、ローカルパラメータの測定値の解析に際して開発された解析プログラムを組み込みながら、TPC 全体のトラッキングメソッドや、シミュレータ特有の解析ルーチン (Monte-Carlo Truth の情報を利用したカンニングルーチン) を開発する。

また、これとは独立に 2. と 3. を併せ持った簡易型のシミュレータを開発する。これは解析に URANUS/Satellites と共通した Kalman-filter ライブラリを用いており、イベントの生成が解析ライブラリと直結しているものである。

Chapter 2

Extended Kalman Filter

2.1 概要

飛跡の再構成する過程は、1 イベント中の全荷電粒子によって生成された Hit 点の集合から特定のトラックに属する Hit 点を選び出す Track Finding と、そうして選び出された Hit 点を Fit してトラックパラメータを決定する Track Fitting に分けられる。ここでは、Track Fitting、すなわち衝突点での正確なトラックパラメータを決定する方法について考察する。

一様磁場中では、検出器内に多重散乱やエネルギー損失がないときには、荷電粒子はトラックパラメータ $(d_\rho, \phi_0, \kappa, d_z, \tan\lambda)$ で規定される Helix で表される。トラックパラメータを決定する最も単純な方法は、多重散乱やエネルギー損失がなく、飛跡が完全な Helix を描くと仮定して、χ^2 fit を行うものである。しかしながら、この方法では 1 つの Hit 点の集合に対して 1 つのトラックパラメータしか持つことができない。低運動量域では、多重散乱や電離によるエネルギー損失が重要となってくるために、途中で比較的大きな外乱を受け飛跡が変化し、その後の Hit 点が衝突点付近の Fit の精度を悪化させてしまう。(図 2.1-a))。

そこで、この問題を解決するために、高エネルギー物理の分野では 1987 年ごろからトラックの再構成に Kalman filter が用いられはじめた。 [13, 14, 15] Kalman filter のアイデアは、1960 年 [12] にランダムな外乱の影響下での線形系を扱うために考え出されたもので、位置の決まった複数の測定系に対から効果的に測定を行えるようにデザインされている。この方法は、

- 運動方程式が (近似的に) 線形であり [1]、
- 離散的な測定点が多数あり、
- それぞれの測定点間の距離が、k 番目の測定点における状態が、ランダムな外乱中における運動方程式を用いて、$k-1$ 番目の測定点における状態を $k-1$ 番目から k 番目の測定点へ外挿したものにほぼ近似できる程度に短い場合

[1] オリジナルの Kalman filter は、線形系に対するものであり、磁場中の荷電粒子の飛跡のような非線形系の扱いは、それを拡張したものである。その意味で、厳密に言えば Extended Kalman filter と呼ぶべきであるが、ここでは単に Kalman filter と呼ぶ。

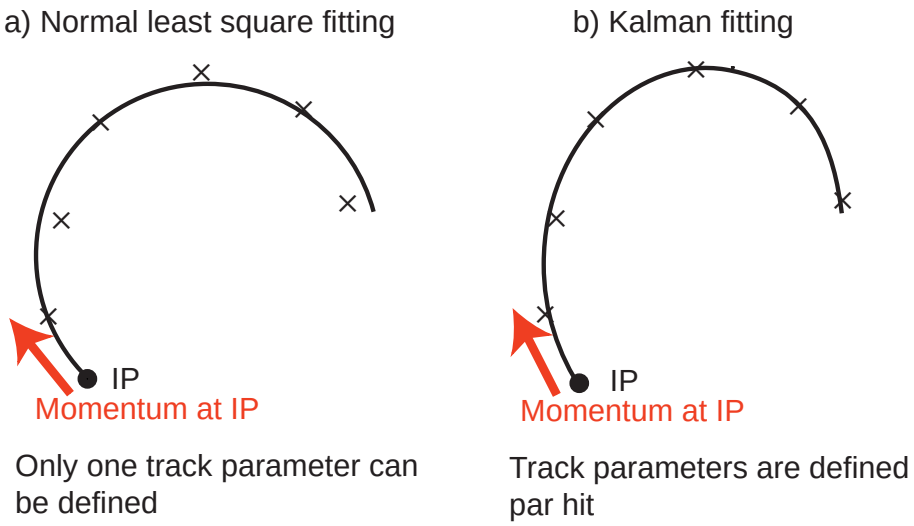

Figure 2.1: Simple χ^2 fitting versus Kalman filter based fitting.

に特に効果的な方法である。Helix に対する χ^2 fit との最大の違いは、測定点が増えるごとにトラックパラメータを更新するため、測定点ごとに異なるトラックパラメータを与えることができる点にある。これによりエネルギー損失、多重散乱や $K \to \mu\nu$ or $\pi \to \mu\nu$ 等の崩壊を考慮できるようになるので、これによって衝突点付近での fit が改善し、その結果粒子の運動量の推定値が改善する。(図 2.1-b)).

2.2 Kalman Filter の原理

2.2.1 問題の説明

一般的に Kalman filter を用いる系は、運動方程式 (*system equation*) に従う過程の間でのランダムな外乱 (*process noise*) の影響を受ける。複数の測定点 (*measurement sites*) で測定された情報を元に、与えられた点における、この系の状態の最も確からしい推定値を導き出すことが目標となる。系の状態が p 次元の列ベクトル (*state vector*) で規定できると仮定し、$\bar{\boldsymbol{a}}_k$ を k 番目の点における真の値とする。$\bar{\boldsymbol{a}}_{k-1}$ と $\bar{\boldsymbol{a}}_k$ の間の関係を表す系の方程式は以下のように書ける。

$$\bar{\boldsymbol{a}}_k = \boldsymbol{f}_{k-1}(\bar{\boldsymbol{a}}_{k-1}) + \boldsymbol{w}_{k-1} \tag{2.1}$$

ここで、$\boldsymbol{f}_{k-1}(\bar{\boldsymbol{a}}_{k-1})$ は一般的に非線形であり、*process noise* である $\boldsymbol{w}_{k-1}$ が無視できる場合に期待される決定論的運動に対応する *state propagator* である。process noise は偏りがなく ($< \boldsymbol{w}_{k-1} >= 0$)、共変行列が以下の式で与えられるものと仮定する。

$$\boldsymbol{Q}_{k-1} \equiv < \boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T > \tag{2.2}$$

各測定点においては、系に関して観測可能な量だけが測定される。m 次元の列ベクトルで表される、k 番目の点における観測可能な量の観測値の集合は、*measurement vector*(測定ベクトル) と呼ばれ、$\boldsymbol{m}_k$ で表すこととする。よって測定ベクトルと状態ベクトルとの関係は *measurement equation*:

$$\boldsymbol{m}_k = \boldsymbol{h}_k(\bar{\boldsymbol{a}}_k) + \boldsymbol{\epsilon}_k \tag{2.3}$$

のように表される。ここで $\boldsymbol{h}_k(\bar{\boldsymbol{a}}_k)$ は *measurement noise* に起因するランダムな誤差 $\boldsymbol{\epsilon}_k$ が無視できる場合の measument vector である。また、$\boldsymbol{h}_k(\bar{\boldsymbol{a}}_k)$ は一般に線形ではなく、measument vector の成分を推定する *projector* としての役割を果たす。$\boldsymbol{\epsilon}_k$ に偏りがなく[2]($<\boldsymbol{\epsilon}_k>=0$)、共変行列が

$$\boldsymbol{V}_k \equiv (\boldsymbol{G})^{-1} \equiv <\boldsymbol{\epsilon}_k \boldsymbol{\epsilon}_k^T> \tag{2.4}$$

で与えられるものと仮定する。

例 1: 弾道ミサイル

Kalman filter は元来弾道ミサイルを追跡し、その飛跡を予測するために発明された。この場合、状態ベクトル $\boldsymbol{a}_k$ は、k 番目の測定点を点 $k-1$ から点 k までの間の乱気流に対応する process noise($\boldsymbol{w}_{k-1}$) を用いて、ハミルトン方程式に従って展開した運動量と位置

$$\boldsymbol{a}_k = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{p} \end{pmatrix}_k \tag{2.5}$$

から成り立つ。
レーダーで検出された点 k におけるミサイルの位置と速度はレーダーの有限測定誤差 $\boldsymbol{\epsilon}_k$ をもった測定ベクトル $\boldsymbol{m}_k$ を形成する。

例 2: 高エネルギー実験でのトラッキング

我々の主な関心は磁場中での荷電粒子のトラッキングにある。もし磁場が、少なくとも点 k の近辺において一様ならば、Helix によってこの粒子のトラックを近似することができる [16]。

$$\begin{cases} x &= x_0 &+ d_\rho \cos\phi_0 &+ \frac{\alpha}{\kappa}(\cos\phi_0 - \cos(\phi_0+\phi)) \\ y &= y_0 &+ d_\rho \sin\phi_0 &+ \frac{\alpha}{\kappa}(\sin\phi_0 - \sin(\phi_0+\phi)) \\ z &= z_0 &+ d_z &- \frac{\alpha}{\kappa}\tan\lambda\cdot\phi \end{cases} \tag{2.6}$$

ここで、$\boldsymbol{x}_0 = (x_0, y_0, z_0)_k^T$ は点 k を表す基準点 (*pivot*) である。測定点が与えられると、Helix は以下の 5 つのパラメータで規定される。

$$\boldsymbol{a}_k = \begin{pmatrix} d_\rho \\ \phi_0 \\ \kappa \\ d_z \\ \tan\lambda \end{pmatrix}_k \tag{2.7}$$

この場合、このパラメータは状態ベクトルとしての役割を果たす。よって基準点を変更すると、Helix パラメータは変化することになる。点 $k-1$ から点 k へ測定点を移すと、状態ベクトルは基準点の変更に伴って $\boldsymbol{a}_{k-1}$ から $\boldsymbol{a}_k$ へと移るので、基準点の変化は、系の方程式中の $\boldsymbol{f}_{k-1}$ を決めることになる。また、この場合には process noise $\boldsymbol{w}_{k-1}$ は、点 $k-1$ から点 k の測定点の間での多重散乱やエネルギー損失から発生する。この点は後に詳しく述べる。measument vector $\boldsymbol{m}_k$ は、検出器のノイズ $\boldsymbol{\epsilon}_k$ の影響を受けた、点 k で測定された座標の集合である。

[2]tracking の場合、測定器のアライメントが正しく行われ、測定座標の系統誤差が無視できると仮定することに対応する

必要事項

measument vector の蓄積は系の状態の見積もりを改善していく。それに伴って、以下の漸化式が必要となる。

- *Prediction* : k 番目までの測定結果を基にして $k''(>k)$ 番目の状態ベクトルを予測する。
$$\{\boldsymbol{m}'_k; k' \leq k\} \mapsto \boldsymbol{a}^k_{k''>k} \qquad : \text{future}$$
- *Filtering* : k 番目での測定結果を加えることで、$k-1$ 番目までの測定結果を基にして、prediction した k 番目での状態ベクトルを更新する。
$$\{\boldsymbol{m}'_k; k' \leq k\} \mapsto \boldsymbol{a}^k_{k''=k} \qquad : \text{present}$$
- *Smoothing*, k 番目までの測定結果を全て用いることで $(k'' < k)$ 番目の filtering された状態ベクトルを改善する。
$$\{\boldsymbol{m}'_k; k' \leq k\} \mapsto \boldsymbol{a}^k_{k''<k} \qquad : \text{past}$$

この章の最後に、これら 3 つの漸化式を導く[3]。

2.2.2 Prediction: 次の測定点の推定

predition は、点 $k-1$ までの測定結果から点 k での状態ベクトルを予測する。式 2.1 は点 $k-1$ と点 k の測定点とを関係付けるものであった。系を点 $k-1$ から点 k へ測定点を移す際のプロセスノイズ (式 2.1 の右辺第二項) を、観測前に知ることはできないので、式 2.1 の右辺第一項のみを用いるか、state propagator

$$\boldsymbol{a}^{k-1}_k = \boldsymbol{f}_{k-1}(\boldsymbol{a}^{k-1}_{k-1}) = \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1}) \tag{2.9}$$

を用いて、点 $k-1$ での状態ベクトルを外挿することが、最善な方法となる。定義より、$\boldsymbol{a}_{k-1} \equiv \boldsymbol{a}^{k-1}_{k-1}$ の共変行列 (誤差行列) は、

$$\begin{aligned} \boldsymbol{C}_{k-1} &\equiv \boldsymbol{C}^{k-1}_{k-1} \\ &\equiv \left\langle (\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1}) (\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1})^T \right\rangle \end{aligned} \tag{2.10}$$

で与えられる。同様にして、prediction した状態ベクトル $\boldsymbol{a}^{k-1}_k$ に対する共変行列を定義でき、$\boldsymbol{C}_{k-1}$ の項を式 2.9、式 2.1、式 2.10 を用いて、

$$\boldsymbol{C}^{k-1}_k \equiv \left\langle (\boldsymbol{a}^{k-1}_k - \bar{\boldsymbol{a}}_k) (\boldsymbol{a}^{k-1}_k - \bar{\boldsymbol{a}}_k)^T \right\rangle \tag{2.11}$$

[3] 今回用いる記法を以下にまとめる。

$$\left\{ \begin{array}{lcl} \boldsymbol{a}^i_k & : & i \text{ 番目までの測定結果を用いた } \bar{\boldsymbol{a}}_k \text{ の見積もり} \\ & & \text{(記法の簡略化のため } \boldsymbol{a}^k_k \equiv \boldsymbol{a}_k \text{ とする)} \\ \boldsymbol{C}^i_k & : & \boldsymbol{a}^i_k \text{ の共変行列} \\ & & \boldsymbol{C}^i_k \equiv \left\langle (\boldsymbol{a}^i_k - \bar{\boldsymbol{a}}_k)(\boldsymbol{a}^i_k - \bar{\boldsymbol{a}}_k)^T \right\rangle \\ \boldsymbol{r}^i_k & : & \text{residual} \\ & & \boldsymbol{r}^i_k \equiv \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}^i_k) \\ \boldsymbol{R}^i_k & : & \boldsymbol{r}^i_k \text{ の共変行列} \\ & & \boldsymbol{R}^i_k \equiv \left\langle \boldsymbol{r}^i_k \boldsymbol{r}^{iT}_k \right\rangle \end{array} \right. \tag{2.8}$$

特に断りがなければ、記号 k は k 番目の測定点を表し、記号 i は i 番目までの測定結果を表す。i, j のような記号であれば、i から j 番目までの測定結果を表す。また、上付き文字と下付き文字が同じならば、特に混乱の恐れがなければ上付き文字を省略する。

$$
\begin{aligned}
&= \left\langle \left(\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1}) - \boldsymbol{f}_{k-1}(\bar{\boldsymbol{a}}_k) - \boldsymbol{w}_{k-1}\right)\left(\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1}) - \boldsymbol{f}_{k-1}(\bar{\boldsymbol{a}}_k) - \boldsymbol{w}_{k-1}\right)^T \right\rangle \\
&\simeq \left\langle \left(\boldsymbol{F}_{k-1}(\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1}) - \boldsymbol{w}_{k-1}\right)\left(\boldsymbol{F}_{k-1}(\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1}) - \boldsymbol{w}_{k-1}\right)^T \right\rangle \\
&= \boldsymbol{F}_{k-1}\left\langle (\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1})(\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1})^T \right\rangle \boldsymbol{F}_{k-1}^T + \left\langle \boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T \right\rangle \\
&\quad +\text{交差項} \\
&= \boldsymbol{F}_{k-1}\boldsymbol{C}_{k-1}\boldsymbol{F}_{k-1}^T + \left\langle \boldsymbol{w}_{k-1}\boldsymbol{w}_{k-1}^T \right\rangle \\
&\quad +\text{交差項}
\end{aligned} \tag{2.12}
$$

と表すことができる。ここで、

$$
\begin{aligned}
\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1}) - \boldsymbol{f}_{k-1}(\bar{\boldsymbol{a}}_k) &\simeq \left(\frac{\partial \boldsymbol{f}_{k-1}}{\partial \boldsymbol{a}_{k-1}}\right)(\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1}) \\
&= \boldsymbol{F}_{k-1}(\boldsymbol{a}_{k-1} - \bar{\boldsymbol{a}}_{k-1})
\end{aligned}
$$

である。点 $k-1$ での、プロセスノイズと、見積もられた状態ベクトルのふらつきは統計的に独立しているので、上記の方程式の交差項は無視でき、よって式 2.2 は

$$
\boldsymbol{C}_k^{k-1} = \boldsymbol{F}_{k-1}\boldsymbol{C}_{k-1}\boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1} \tag{2.13}
$$

となる。ここで、

$$
\boldsymbol{F}_{k-1} \equiv \left(\frac{\partial \boldsymbol{f}_{k-1}}{\partial \boldsymbol{a}_{k-1}}\right) \tag{2.14}
$$

を *propagator matrix* と呼ぶことにする。

2.2.3 Filtering: 現在点での見積もりの最適化

上記で述べたように、filtering とは、点 $k-1$ までの測定点に点 k での結果を加えることで点 k での prediction した state vectr を更新し、点 k までの全ての情報を用いて、点 k での状態ベクトルの最適値を求めることである。漸化式として定式化することがここでの目標となる。

式 2.13 で与えられる、prediction した状態ベクトル $(\boldsymbol{a}_k^{k-1})$ に対する共変行列は、点 $k-1$ までの結果から得られる点 k における状態ベクトルに対する我々の知識の全てが、次の χ^2 に集約できることを示している:

$$
(\chi^2)_k^{k-1} = (\chi^2)_{k-1}^{k-1} + (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})^T \left(\boldsymbol{C}_k^{k-1}\right)^{-1} (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})
$$

ここで $\boldsymbol{a}_k^*$ は下記の点 k での測定情報を加えることで最適化された、点 k における状態ベクトルの新たな見積もりである。また $(\chi^2)_{k-1}^{k-1} \equiv (\chi^2)_{k-1}$ は点 $k-1$ までの χ^2 であり、$\boldsymbol{a}_k^*$ と独立である。これに、点 k での新たな測定情報:

$$
(\chi^2)_k^{k,k} = (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^*))^T \boldsymbol{G} (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^*))
$$

を加えると、$(\chi^2)_{k-1}^{k-1}$ の差分として、

$$
\begin{aligned}
\chi_+^2 &= (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})^T \left(\boldsymbol{C}_k^{k-1}\right)^{-1} (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1}) \\
&\quad + (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^*))^T \boldsymbol{G} (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^*)) \\
&= (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})^T \left(\boldsymbol{C}_k^{k-1}\right)^{-1} (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})
\end{aligned} \tag{2.15}
$$

$$
\begin{aligned}
& + \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) - (\boldsymbol{h}_k(\boldsymbol{a}_k^*) - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}))\right)^T \boldsymbol{G} \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) - (\boldsymbol{h}_k(\boldsymbol{a}_k^*) - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}))\right) \\
\simeq\ & (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})^T \left(\boldsymbol{C}_k^{k-1}\right)^{-1} (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1}) \\
& + \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) - \boldsymbol{H}_k(\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})\right)^T \boldsymbol{G} \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) - \boldsymbol{H}_k(\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})\right) \qquad (2.16)
\end{aligned}
$$

が得られる。ここで、$\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1}$ が小さいと仮定し、Taylor 展開の一時の項までを残し、

$$
\boldsymbol{h}_k(\boldsymbol{a}_k^*) \simeq \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) + \boldsymbol{H}_k\ (\boldsymbol{a}_k^* - \boldsymbol{a}_k^{k-1})
$$

とした。ここで projector の微分:

$$
\boldsymbol{H}_k = \left(\frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{a}_k^{k-1}}\right) \qquad (2.17)
$$

を *projector matrix* と呼ぶことにする。$(\chi^2)_{k-1}^{k-1}$ は、$\boldsymbol{a}_{k-1}$ で最小値をとり、$\boldsymbol{a}_k^*$ とは独立である[4]。よって、この差分 (χ_+^2) を最小にするような $\boldsymbol{a}_k^*$ から、点 k までの測定情報を用いて最適化された点 k での状態ベクトル $(\boldsymbol{a}_k \equiv \boldsymbol{a}_k^k)$ を得ることができる。極値条件

$$
\frac{\partial \chi_+^2}{\partial \boldsymbol{a}_k^*} = 0
$$

を解くことにより、

$$
\boldsymbol{a}_k = \boldsymbol{a}_k^{k-1} + \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k\right]^{-1} \boldsymbol{H}_k^T \boldsymbol{G}_k \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})\right)
$$

が得られ、

$$
\boldsymbol{a}_k = \boldsymbol{a}_k^{k-1} + \boldsymbol{K}_k \left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})\right) \qquad (2.18)
$$

となる。ただし、

$$
\boldsymbol{K}_k = \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k\right]^{-1} \boldsymbol{H}_k^T \boldsymbol{G}_k \qquad (2.19)
$$

とした。ここで定義された行列 $\boldsymbol{K}_k$ は *Kalman Gain Matrix* と呼ばれ、点 k での新たら測定が、点 k における prediction した状態ベクトルをどのように改善するかを表している。(式 2.18 の第二項は測定ベクトル $\boldsymbol{m}_k$ に対する新たな特性に対応する)

Kalman gain matrix に対する上述の表現は別の形に書き換えることができる。すなわち、式 2.19 より、

$$
\begin{aligned}
\boldsymbol{K}_k \left(\boldsymbol{H}_k \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T + (\boldsymbol{G}_k)^{-1}\right) &= \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k\right]^{-1} \left(\boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T + \boldsymbol{H}_k^T\right) \\
&= \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k\right]^{-1} \\
&\quad \times \left[\left\{\boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k + \left(\boldsymbol{C}_k^{k-1}\right)^{-1} - \left(\boldsymbol{C}_k^{k-1}\right)^{-1}\right\} \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T + \boldsymbol{H}_k^T\right] \\
&= \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T
\end{aligned}
$$

となり、これより、

$$
\boldsymbol{K}_k = \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T \left(\boldsymbol{V}_k + \boldsymbol{H}_k \boldsymbol{C}_k^{k-1} \boldsymbol{H}_k^T\right)^{-1}, \qquad (2.20)
$$

[4]途中の点における状態ベクトルがどう改善されるかについては後に検討する

となる。ここで式 2.4 を用いた。

式 2.19 や 2.20 に加えて、式 2.18 より、点 k での新たな測定情報を加えることで prediction した状態ベクトル $\boldsymbol{a}_k^{k-1}$ を更新でき、filtering された状態ベクトル $\boldsymbol{a}_k \equiv \boldsymbol{a}_k^k$ を得ることができる。

$\boldsymbol{a}_k$ に対する共変行列を得るために共変行列 $\boldsymbol{C}_k^{k-1}$ を更新するための漸化式を求めることを考える。式 2.18 の共変行列の定義より、

$$\begin{aligned}
\boldsymbol{C}_k &\equiv \left\langle (\boldsymbol{a}_k - \bar{\boldsymbol{a}}_k)(\boldsymbol{a}_k - \bar{\boldsymbol{a}}_k)^T \right\rangle \\
&= \left\langle \left(\boldsymbol{a}_k^{k-1} + \boldsymbol{K}_k(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})) - \bar{\boldsymbol{a}}_k\right) \left(\boldsymbol{a}_k^{k-1} + \boldsymbol{K}_k(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})) - \bar{\boldsymbol{a}}_k\right)^T \right\rangle \\
&= \left\langle \left[(\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k) + \boldsymbol{K}_k \left\{\boldsymbol{m}_k - \boldsymbol{h}_k(\bar{\boldsymbol{a}}_k) - \left(\boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) - \boldsymbol{h}_k(\bar{\boldsymbol{a}}_k)\right)\right\}\right] [\cdots]^T \right\rangle
\end{aligned}$$

となる。これより、式 2.3 を展開することによって、

$$\begin{aligned}
\boldsymbol{C}_k &= \left\langle \left[(\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k) + \boldsymbol{K}_k \left\{\boldsymbol{\epsilon}_k - \left(\boldsymbol{h}_k(\boldsymbol{a}_k^{k-1}) - \boldsymbol{h}_k(\bar{\boldsymbol{a}}_k)\right)\right\}\right] [\cdots]^T \right\rangle \\
&\simeq \left\langle \left[(\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k) + \boldsymbol{K}_k \left\{\boldsymbol{\epsilon}_k - \boldsymbol{H}_k(\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k)\right\}\right] [\cdots]^T \right\rangle \\
&= \left\langle \left[(1 - \boldsymbol{K}_k\boldsymbol{H}_k)(\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k) + \boldsymbol{K}_k\boldsymbol{\epsilon}_k\right] [\cdots]^T \right\rangle
\end{aligned}$$

を得る。ここで、$\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k$ は小さいと仮定し、Taylor 展開の一次まで取った。点 K での filtering した状態ベクトルと、実際の状態ベクトルとの差、$\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k$ は統計的に点 k の測定誤差 $\boldsymbol{\epsilon}_k$ と独立しているので、交差項は消えて、

$$\begin{aligned}
\boldsymbol{C}_k &= (1 - \boldsymbol{K}_k\boldsymbol{H}_k) \left\langle (\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k)(\boldsymbol{a}_k^{k-1} - \bar{\boldsymbol{a}}_k)^T \right\rangle (1 - \boldsymbol{K}_k\boldsymbol{H}_k)^T + \boldsymbol{K}_k \left\langle \boldsymbol{\epsilon}_k\boldsymbol{\epsilon}_k^T \right\rangle \boldsymbol{K}_k^T \\
&= (1 - \boldsymbol{K}_k\boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1} (1 - \boldsymbol{K}_k\boldsymbol{H}_k)^T + \boldsymbol{K}_k\boldsymbol{V}_k\boldsymbol{K}_k^T \qquad (2.21) \\
&= (1 - \boldsymbol{K}_k\boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1} - \left[(1 - \boldsymbol{K}_k\boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1}\boldsymbol{H}_k^T - \boldsymbol{K}_k\boldsymbol{V}_k\right] \boldsymbol{K}_k^T \\
&= (1 - \boldsymbol{K}_k\boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1} - \left[\boldsymbol{C}_k^{k-1}\boldsymbol{H}_k^T - \boldsymbol{K}_k(\boldsymbol{H}_k\boldsymbol{C}_k^{k-1}\boldsymbol{H}_k^T + \boldsymbol{V}_k)\right] \boldsymbol{K}_k^T \\
&= (1 - \boldsymbol{K}_k\boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1} \qquad (2.22)
\end{aligned}$$

を得る。ここで式 2.11、式 2.4、式 2.20 を用いた。式 2.19 を用いて、$\boldsymbol{C}_k$ に対する上記の方程式をさらに下記のように書き直すことができる。

$$\begin{aligned}
\boldsymbol{C}_k &= (1 - \boldsymbol{K}_k\boldsymbol{H}_k)\, \boldsymbol{C}_k^{k-1} \\
&= \left(1 - \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k\right]^{-1} \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k\right) \boldsymbol{C}_k^{k-1} \\
&= \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k\right]^{-1} \left\{\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k - \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k\right\} \boldsymbol{C}_k^{k-1} \\
&= \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k\right]^{-1}
\end{aligned}$$

これより、

$$\boldsymbol{C}_k = \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T\boldsymbol{G}_k\boldsymbol{H}_k\right]^{-1} \qquad (2.23)$$

となり、加重平均した filtering した共変行列を得る。この式は、新たな共変行列 $\boldsymbol{C}_k$ が、実際には prediction した共変行列 $\boldsymbol{C}_k^{k-1}$ よりも小さいことを示している。この式と式 2.19 とを合わせて、Kalman Gain Matrix に対するもう一つの式

$$\boldsymbol{K}_k = \boldsymbol{C}_k\boldsymbol{H}_k^T\boldsymbol{G}_k \qquad (2.24)$$

を得る。

2.2.4 Smoothing: 過ぎた点の再評価

一般に、filtering の過程において、点 k で得られた状態ベクトルはその点において最適化されている。しかし、点 k における状態ベクトルは、その後に続く、点 $k+1$ から点 n までの情報を加えて再評価することでより精度を高めることができる。この過程を *Smoothing* と呼ぶ。明らかに、末尾の点 n の filtering の結果は、smoothing の結果と一致する。故に、点 n から出発して点 1 に向かい、逐次 filtering した状態ベクトルをこれより、点 $k+1$ での smoothing した状態ベクトルと点 k と $k+1$ での filtering した状態ベクトルから点 k における smoothing した状態ベクトルを与える漸化式を表現することが目標である。

この目標を達成するために、点 $k+1$ における smoothing を考える。n 個の点全ての情報を用いた点 $k+1$ における状態ベクトルの最適値 $(\boldsymbol{a}_{k+1}^n)$ は、点 1 から k までの結果を用いた点 $k+1$ での prediction した状態ベクトル $(\boldsymbol{a}_{k+1}^k)$ と、点 n から点 $k+1$ までの結果を基にした点 $k+1$ における逆行して filtering した状態ベクトル $(\boldsymbol{a}_{k+1}^{n,k+1})$ との加重平均として与えられるはずである。この加重平均は、次の χ^2 を最小化することで得られる。

$$\begin{aligned}\chi^2 &= (\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k)^T \left(\boldsymbol{C}_{k+1}^k\right)^{-1} (\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k) \\ &\quad +(\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^{n,k+1})^T \left(\boldsymbol{C}_{k+1}^{n,k+1}\right)^{-1} (\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^{n,k+1})\end{aligned}$$

この χ 2 を $\boldsymbol{a}_{k+1}^n$ で微分して 0 とおくことにより、ただちに点 $k+1$ における smoothing した state vecetor

$$\boldsymbol{a}_{k+1}^n = \left[\left(\boldsymbol{C}_{k+1}^k\right)^{-1} + \left(\boldsymbol{C}_{k+1}^{n,k+1}\right)^{-1}\right]^{-1} \left[\left(\boldsymbol{C}_{k+1}^k\right)^{-1} \boldsymbol{a}_{k+1}^k + \left(\boldsymbol{C}_{k+1}^{n,k+1}\right)^{-1} \boldsymbol{a}_{k+1}^{n,k+1}\right]$$

が得られる。また、共変行列は

$$(\boldsymbol{C}_{k+1}^n)^{-1} = \left(\boldsymbol{C}_{k+1}^k\right)^{-1} + \left(\boldsymbol{C}_{k+1}^{n,k+1}\right)^{-1}$$

で与えられる。上記の 2 式を $\boldsymbol{a}_{k+1}^{n,k+1}$ と $\left(\boldsymbol{C}_{k+1}^{n,k+1}\right)^{-1}$ について解くと、

$$\boldsymbol{a}_{k+1}^{n,k+1} = \boldsymbol{a}_{k+1}^n + \boldsymbol{C}_{k+1}^n \left[\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right]^{-1} (\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k) \tag{2.25}$$

$$\left(\boldsymbol{C}_{k+1}^{n,k+1}\right)^{-1} = (\boldsymbol{C}_{k+1}^n)^{-1} - \left(\boldsymbol{C}_{k+1}^k\right)^{-1} \tag{2.26}$$

となる。同様に点 k における smooting した状態ベクトルについても

$$\boldsymbol{a}_k^n = \left[(\boldsymbol{C}_k)^{-1} + \left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\right]^{-1} \left[(\boldsymbol{C}_k)^{-1} \boldsymbol{a}_k + \left(\boldsymbol{C}_k^{n,k+1}\right)^{-1} \boldsymbol{a}_k^{n,k+1}\right] \tag{2.27}$$

$$(\boldsymbol{C}_k^n)^{-1} = (\boldsymbol{C}_k)^{-1} + \left(\boldsymbol{C}_k^{n,k+1}\right)^{-1} \tag{2.28}$$

が得られる。式 2.25 と式 2.26 を用いて、上記の二つの方程式から上添字 n,k+1 を持つ項をとり省くことが目標である。

共変行列の定義より、

$$\boldsymbol{C}_k^{n,k+1} \equiv \left\langle (\boldsymbol{a}_k^{n,k+1} - \bar{\boldsymbol{a}}_k)(\boldsymbol{a}_k^{n,k+1} - \bar{\boldsymbol{a}}_k)^T \right\rangle$$

であり、これと式 2.1 から、

$$\begin{aligned}
\boldsymbol{C}_k^{n,k+1} &= \left\langle \left(\boldsymbol{f}_k^{-1}(\boldsymbol{a}_{k+1}^{n,k+1}) - \boldsymbol{f}_k^{-1}(\bar{\boldsymbol{a}}_{k+1} - \boldsymbol{w}_k)\right)\left(\boldsymbol{f}_k^{-1}(\boldsymbol{a}_{k+1}^{n,k+1}) - \boldsymbol{f}_k^{-1}(\bar{\boldsymbol{a}}_{k+1} - \boldsymbol{w}_k)\right)^T \right\rangle \\
&\simeq \left\langle \left(\boldsymbol{F}_k^{-1}(\boldsymbol{a}_{k+1}^{n,k+1} - \bar{\boldsymbol{a}}_{k+1} + \boldsymbol{w}_k)\right)\left(\boldsymbol{F}_k^{-1}(\boldsymbol{a}_{k+1}^{n,k+1} - \bar{\boldsymbol{a}}_{k+1} + \boldsymbol{w}_k)\right)^T \right\rangle \\
&= \boldsymbol{F}_k^{-1} \left\langle \left((\boldsymbol{a}_{k+1}^{n,k+1} - \bar{\boldsymbol{a}}_{k+1}) + \boldsymbol{w}_k\right)\left((\boldsymbol{a}_{k+1}^{n,k+1} - \bar{\boldsymbol{a}}_{k+1}) + \boldsymbol{w}_k\right)^T \right\rangle \boldsymbol{F}_k^{-1^T}
\end{aligned}$$

が得られる。ここで $\boldsymbol{a}_{k+1}^{n,k+1}$ と $\bar{\boldsymbol{a}}_{k+1} - \boldsymbol{w}_k$ は近接していて、$\boldsymbol{f}_k^{-1}$ が Taylor 展開の一次まででよく近似できるものと仮定した。点 k から点 $k+1$ の間のプロセスノイズが統計的に $\boldsymbol{a}_{k+1}^{n,k+1} - \bar{\boldsymbol{a}}_{k+1}$ の差と独立していることに注意すると、式 2.2 と共変行列の定義から

$$\boldsymbol{C}_k^{n,k+1} = \boldsymbol{F}_k^{-1}\left(\boldsymbol{C}_{k+1}^{n,k+1} + \boldsymbol{Q}_k\right)\boldsymbol{F}_k^{-1^T} \tag{2.29}$$

を得る。$\boldsymbol{C}_{k+1}^{n,k+1}$ は式 2.26 で与えられるので、これで上添字 $n, k+1$ を持つ項を取り省いたことになる。

一方、式 2.27 と式 2.28 より

$$\begin{aligned}
\boldsymbol{a}_k^n &= \boldsymbol{C}_k^n\left\{(\boldsymbol{C}_k^n)^{-1}\boldsymbol{a}_k + \left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}(\boldsymbol{a}_k^{n,k+1} - \boldsymbol{a}_k)\right\} \\
&= \boldsymbol{a}_k + \boldsymbol{C}_k^n\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}(\boldsymbol{a}_k^{n,k+1} - \boldsymbol{a}_k) \\
&= \boldsymbol{a}_k + \boldsymbol{C}_k^n\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\left(\boldsymbol{f}_k^{-1}(\boldsymbol{a}_{k+1}^{n,k+1}) - \boldsymbol{f}_k^{-1}(\boldsymbol{a}_{k+1}^k)\right) \\
&\simeq \boldsymbol{a}_k + \boldsymbol{C}_k^n\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\boldsymbol{F}_k^{-1}(\boldsymbol{a}_{k+1}^{n,k+1} - \boldsymbol{a}_{k+1}^k)
\end{aligned}$$

となるが、式 2.25 を用いて簡単な変形を行うと

$$\boldsymbol{a}_k^n = \boldsymbol{a}_k + \boldsymbol{A}_k(\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k) \tag{2.30}$$

$$\boldsymbol{A}_k \equiv \boldsymbol{C}_k^n\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1} \tag{2.31}$$

が得られる。$\boldsymbol{A}_k$ の中の $\boldsymbol{C}_k^n$ と $\boldsymbol{C}_k^{n,k+1}$ は、式 2.28、式 2.29 と式 2.28 を用いることで、それぞれ以下のようになる。

$$\begin{aligned}
\boldsymbol{A}_k &\equiv \boldsymbol{C}_k^n\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1} \\
&= \left[(\boldsymbol{C}_k)^{-1} + \left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\right]^{-1}\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1} \\
&= \left[\boldsymbol{C}_k^{n,k+1}\left\{(\boldsymbol{C}_k)^{-1} + \left(\boldsymbol{C}_k^{n,k+1}\right)^{-1}\right\}\right]^{-1}\boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1} \\
&= \left[\left(\boldsymbol{C}_k + \boldsymbol{C}_k^{n,k+1}\right)(\boldsymbol{C}_k)^{-1}\right]^{-1}\boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1} \\
&= \boldsymbol{C}_k\left(\boldsymbol{C}_k + \boldsymbol{C}_k^{n,k+1}\right)^{-1}\boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1}
\end{aligned} \tag{2.32}$$

また、式 2.13 と式 2.29 は

$$\boldsymbol{C}_k + \boldsymbol{C}_k^{n,k+1} = \boldsymbol{F}_k^{-1}\left(\boldsymbol{C}_{k+1}^k + \boldsymbol{C}_{k+1}^{n,k+1}\right)\boldsymbol{F}_k^{-1^T}$$

となることを示しており、式 2.26 より

$$\boldsymbol{C}_k + \boldsymbol{C}_k^{n,k+1} = \boldsymbol{F}_k^{-1}\left(\boldsymbol{C}_{k+1}^k + \boldsymbol{C}_{k+1}^{n,k+1}\right)\boldsymbol{F}_k^{-1^T}$$

$$
\begin{aligned}
&= \boldsymbol{F}_k^{-1}\left(\boldsymbol{C}_{k+1}^k + \boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1}\boldsymbol{C}_{k+1}^n\right)\boldsymbol{F}_k^{-1^T} \\
&= \boldsymbol{F}_k^{-1}\boldsymbol{C}_{k+1}^k\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)^{-1}\boldsymbol{C}_{k+1}^k\boldsymbol{F}_k^{-1^T}
\end{aligned} \tag{2.33}
$$

が導かれる。式 2.32 にこれを代入して式 2.30 の結果と合わせると、最終的に

$$
\left\{
\begin{array}{rcl}
\boldsymbol{a}_k^n &=& \boldsymbol{a}_k + \boldsymbol{A}_k(\boldsymbol{a}_{k+1}^n - \boldsymbol{a}_{k+1}^k) \\
\boldsymbol{A}_k &=& \boldsymbol{C}_k\boldsymbol{F}_k^T\left(\boldsymbol{C}_{k+1}^k\right)^{-1}
\end{array}
\right. \tag{2.34}
$$

が得られる。これで目標としていた $\boldsymbol{a}_k^n$ を求める漸化式が得られたことになる。

最後に $\boldsymbol{a}_k^n$ の共変行列を求めておく。式 2.32 と 2.33 から、

$$
\boldsymbol{C}_k^n\left(\boldsymbol{C}_k^{n,k+1}\right)^{-1} = \boldsymbol{C}_k\boldsymbol{F}_k^T\left(\boldsymbol{C}_{k+1}^k\right)^{-1}\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)\left(\boldsymbol{C}_{k+1}^k\right)^{-1}\boldsymbol{F}_k
$$

が得られる。右から $\boldsymbol{C}_k^{n,k+1}$ を両辺にかけ、式 2.33 を用いて $\boldsymbol{C}_k^{n,k+1}$ に代入することで

$$
\boldsymbol{C}_k^n = \boldsymbol{C}_k - \boldsymbol{C}_k\boldsymbol{F}_k^T\left(\boldsymbol{C}_{k+1}^k\right)^{-1}\left(\boldsymbol{C}_{k+1}^k - \boldsymbol{C}_{k+1}^n\right)\left(\boldsymbol{C}_{k+1}^k\right)^{-1}\boldsymbol{F}_k\boldsymbol{C}_k
$$

が得られる。結果として

$$
\boldsymbol{C}_k^n = \boldsymbol{C}_k + \boldsymbol{A}_k\left(\boldsymbol{C}_{k+1}^n - \boldsymbol{C}_{k+1}^k\right)\boldsymbol{A}_k^T \tag{2.35}
$$

を得ることになる。ここで式 2.34 を用いた。

2.2.5 Inverse Kalman Filter: 測定点の除外

最後の課題として、点 1 から点 n までの filtering と、点 n から点 1 までの smooth back を行った後に、状態ベクトルの値から途中の点 k をとり省きたい場合を考える。当然、測定点を取り省いて点 k がないものとして Kalman filter を実行しなおすことはいつでもできる。しかしながら、点 k の直前の結果を最大限活用するより良い方法がある。この過程は *Inverse Kalman Filter* と呼ばれ、特に、飛跡検出器の場合には、点 k に対応する測定面のアライメントを評価したいとき等に有用である。

点 k をとり省くことは、下記の χ 2 の引算に対応する。

$$
\chi^{*2} = (\boldsymbol{a}_k^{n*} - \boldsymbol{a}_k^n)^T(\boldsymbol{C}_k^n)^{-1}(\boldsymbol{a}_k^{n*} - \boldsymbol{a}_k^n) - (\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{n*})^T\boldsymbol{G}_k(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{n*})
$$

$\boldsymbol{a}_k^{n*}$ に対してこの新たな χ $*2$ を微分して、結果を 0 とおくことで、点 k が抜けた測定ベクトルの点 k における状態ベクトルの最適値を求める方程式を書き下すことができる。

filtering の過程において、この式 2.15 を比較することで、χ_+^2 を χ^{*2} に変換する、下記の結果を得ることができる。

$$
\begin{array}{rcl}
\boldsymbol{a}_k^{k-1} & \rightarrow & \boldsymbol{a}_k^n \\
\boldsymbol{C}_k^{k-1} & \rightarrow & \boldsymbol{C}_k^n \\
\boldsymbol{G}_k & \rightarrow & -\boldsymbol{G}_k
\end{array}
$$

これよりただちに、filtering の過程を記述する式 2.18、式 2.20 と式 2.23 を inverse Kalman filter を記述する式

$$\begin{aligned} \boldsymbol{a}_k^{n*} &= \boldsymbol{a}_k^n + \boldsymbol{K}_k^{n*}\left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^n)\right) && (2.36)\\ \boldsymbol{K}_k^{n*} &= \boldsymbol{C}_k^n \boldsymbol{H}_k^T \left(-\boldsymbol{V}_k + \boldsymbol{H}_k \boldsymbol{C}_k^n \boldsymbol{H}_k^T\right)^{-1} && (2.37)\\ \boldsymbol{C}_k^{n*} &= (1 - \boldsymbol{K}_k^{n*}\boldsymbol{H}_k)\boldsymbol{C}_k \; n \\ &= \left[(\boldsymbol{C}_k^n)^{-1} - \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k\right]^{-1} && (2.38) \end{aligned}$$

へ変換することかできる。本質的な差は、点 k における情報を取り省くことに関連して負の重みを表している、$\boldsymbol{G}_k$ や $\boldsymbol{V}_k$ の直前の符号であることに注意が必要である。

2.3 Track Fitting への Kalman Filter の応用

2.3.1 Kalman Filter の基本的な式

この節では、高エネルギー実験における Kalman-filter に基づいた track fitting の手順を定式化する。衝突実験において、一般的に 4π 検出器はソレノイドマグネット中に置かれ、その軸方向の磁場にそって、中心に位置する衝突点を円筒状に囲むかたちで飛跡検出器がある。飛跡検出器はしばしば、その飛跡に沿った粒子の位置情報を収集するために複数の測定面によって構成されている。

まず、前節で求めた漸化式から Kalman-filter に基づいた track fitting に必要な式をピックアップする。Kalman filter を track fitting に応用する際には、通常最外部の測定面からはじめ、衝突点に向かって戻っていく。これは二つの track の距離の平均は最外部が最も大きく、hit 点と track の関連付けを間違える可能性が最も小さいためである。さらに、最内部の hit 点に到達した際、filtering が行われた状態ベクトルはすでに最適化されており、smooting する必要がない。外側の検出器に track を外挿する必要がない限り、必要とする漸化式は以下の二つ、filtering が行われた状態ベクトル ($\boldsymbol{a}_k$) と、その共変行列 ($\boldsymbol{C}_k$) である。

$$\begin{aligned} \boldsymbol{a}_k &= \boldsymbol{a}_k^{k-1} + \boldsymbol{K}_k\left(\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})\right) \\ \boldsymbol{C}_k &= \left[\left(\boldsymbol{C}_k^{k-1}\right)^{-1} + \boldsymbol{H}_k^T \boldsymbol{G}_k \boldsymbol{H}_k\right]^{-1}, \end{aligned} \qquad (2.39)$$

ここで、($\boldsymbol{a}_k^{k-1}$) は prediction が行われた状態ベクトル、($\boldsymbol{K}_k$) は Kalman gain matrix、($\boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})$) は期待される測定ベクトル、($\boldsymbol{m}_k$) はそれに関する測定ベクトル ($\boldsymbol{C}_k^{k-1}$) は ($\boldsymbol{a}_k^{k-1}$) の共変行列、($\boldsymbol{G}_k \equiv (\boldsymbol{V}_k)^{-1}$) は測定誤差行列の逆数である。また Kalman gain matrix は

$$\boldsymbol{K}_k = \boldsymbol{C}_k \boldsymbol{H}_k^T \boldsymbol{G}_k$$

で与えられ、prediction された状態ベクトルとその共変行列は

$$\begin{aligned} \boldsymbol{a}_k^{k-1} &= \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1}) \\ \boldsymbol{C}_k^{k-1} &= \boldsymbol{F}_{k-1}\, \boldsymbol{C}_{k-1}\, \boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1} \end{aligned}$$

で与えられる。ここで $\boldsymbol{Q}_{k-1}$ は点 $k-1$ と点 k の間のプロセスノイズの共変行列、$\boldsymbol{F}_{k-1}$ と $\boldsymbol{H}_k$ は応用する系に依存するの微分係数であり、propagator matrix($\boldsymbol{F}_{k-1}$) と projector matrix($\boldsymbol{H}_k$) は、系と測定のそれ

ぞれの方程式より

$$\begin{cases} \boldsymbol{F}_{k-1} & \equiv & \left(\frac{\partial \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})}{\partial \boldsymbol{a}_{k-1}}\right) \\ \boldsymbol{H}_k & \equiv & \left(\frac{\partial \boldsymbol{h}_k(\boldsymbol{a}_k^{k-1})}{\partial \boldsymbol{a}_k^{k-1}}\right) \end{cases}$$

と求まる。

これらの式は、プロセスノイズ ($\boldsymbol{Q_{k-1}}$) や測定ノイズ ($\boldsymbol{V}_k$) の具体的な式と共に $\boldsymbol{f}_{k-1}$、$\boldsymbol{h}_k$ やその微分 $\boldsymbol{F}_{k-1}$、$\boldsymbol{H}_k$ のような応用する系及び測定器に特有な関数が測定点ごとの基底に、新たな測定点における新たな測定ベクトルを加えることによって系の情報を蓄積するにしたがい、状態ベクトルを更新していくことができる。

2.3.2 系の方程式: $\boldsymbol{f}_{k-1}$、$\boldsymbol{F}_{k-1}$ と $\boldsymbol{Q}_{k-1}$

測定方程式は検出器に依存するので、まず磁場中の荷電粒子の track に対しての系の方程式について考え、$\boldsymbol{f}_{k-1}$、$\boldsymbol{F}_{k-1}$ と $\boldsymbol{Q}_{k-1}$ の具体的な式を導き出すことを試みる。この目的のためには、track model を明確にする必要がある。

依然述べたように、もし磁場が、問題とする測定点の近辺で、少なくとも局所的には z 方向に一様であるならば、この粒子の track を helix として近似できる。(式 2.6)

$$\begin{cases} x & = & x_0 & + & d_\rho \cos\phi_0 & + & \frac{\alpha}{\kappa}\left(\cos\phi_0 - \cos(\phi_0+\phi)\right) \\ y & = & y_0 & + & d_\rho \sin\phi_0 & + & \frac{\alpha}{\kappa}\left(\sin\phi_0 - \sin(\phi_0+\phi)\right) \\ z & = & z_0 & + & d_z & - & \frac{\alpha}{\kappa}\tan\lambda\cdot\phi \end{cases} \tag{2.40}$$

ここで、$\boldsymbol{x}_0 = (x_0, y_0, z_0)_{k-1}^T$ は測定点 $k-1$ を表す基準点であり、ϕ は基準点から測った方位角である。基準点は任意なので、点 $k-1$ つまり $k-1$ 番目の hit 点を選ぶことができる。測定点、あるいは基準点が与えられると、helix は以下の 5 つのパラメータで規定され、状態ベクトル $\boldsymbol{a}_{k-1} = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$ を形成する。

$$\begin{cases} d_\rho & : & x\text{-}y \text{ 平面における helix と基準点との距離} \\ \phi_0 & : & \text{helix の中心に対する基準点の方位角} \\ \kappa & : & \equiv Q/P_t : \text{電荷 / 横運動量} \\ d_z & : & z \text{ 方向における helix と基準点との距離} \\ \tan\lambda & : & \text{dip anple(helix の } x\text{-}y \text{ 平面からの角度,} \end{cases} \tag{2.41}$$

パラメータ κ は $\rho = \alpha/\kappa$ のように符号付半径 ρ と関連しており[5]、α は $\alpha \equiv 1/cB$ で表されるような光速度 c と磁場 B で与えられる定数である。

図 2.2 は helix パラメータを表している。track の符号次第で、π によって ϕ_0 の定義が異なってくることに注意が必要である。これは、track の曲率の符号が track の fitting 中に変化した際に、helix の中心が突然 track を挟んで反対側に移ってしまうのを補正するために、ϕ_0 を置き換えることで不連続になるのを避けているためである。

[5]高運動量の track の fitting では半径の変化を避けるために、track fitting のパラメータとして ρ の代わりに κ を用いるべきである。

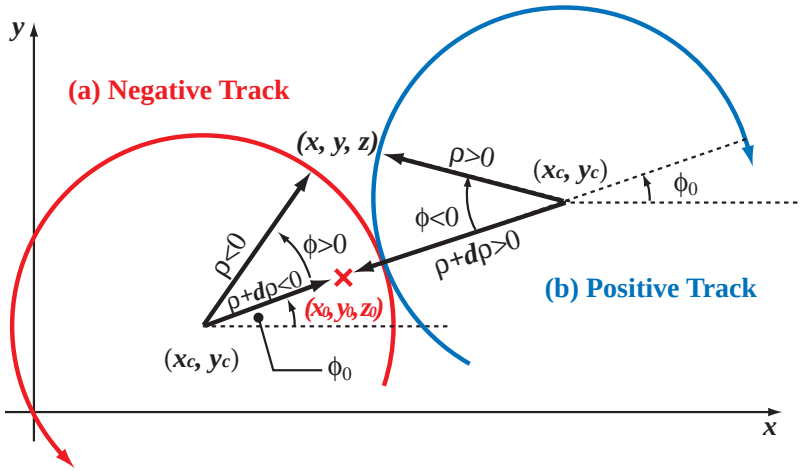

Figure 2.2: 正電荷の track (a) と負電荷の track (b) の幾何学的解釈

State Propagator: $\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})$

点 $k-1$ から点 k への基準点の変更は、helix 自身は変化せずに helix パラメータが変化することに注意する必要がある。点 $k-1$ から点 k へ基準点を移す際に、その移動に伴って state vector は $\boldsymbol{a}_{k-1}$ から $\boldsymbol{a}_k^{k-1}$ へと変化する。つまり基準点の変更は、系の方程式中の伝播関数 $\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})$ を決定する。

図 2.3 は track パラメータ:

$$\boldsymbol{a}_{k-1} \equiv \boldsymbol{a} = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$$

が基準点の変更の際にどのように変化するかを示している。ここで $\boldsymbol{x}$ は helix 上の点、$\boldsymbol{x}_0$ はその基準点、$\boldsymbol{X}_c$ は helix の中心を表している。プライムのついたパラメータ:

$$\boldsymbol{a}' \equiv \boldsymbol{a}_k^{k-1} = (d'_\rho, \phi'_0, \kappa', d'_z, \tan\lambda')^T = \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})$$

はそれぞれ新しい基準点に対応するものである。

図 2.3 を見ると、プライムのついた track パラメータとプライムのついていないものとの間に以下の関係があることが分かる。

$$\begin{cases} d'_\rho &= (X_c - x'_0)\cos\phi'_0 + (Y_c - y'_0)\sin\phi'_0 - \frac{\alpha}{\kappa} \\ \phi'_0 &= \begin{cases} \tan^{-1}\left(\frac{Y_c - y'_0}{X_c - x'_0}\right) & (\kappa > 0) \\ \tan^{-1}\left(\frac{y'_0 - Y_c}{x'_0 - X_c}\right) & (\kappa < 0) \end{cases} \\ \kappa' &= \kappa \\ d'_z &= z_0 - z'_0 + d_z - \left(\frac{\alpha}{\kappa}\right)(\phi'_0 - \phi_0)\tan\lambda \\ \tan\lambda' &= \tan\lambda, \end{cases} \tag{2.42}$$

ここで、

$$\begin{cases} X_c &\equiv x_0 + \left(d\rho + \frac{\alpha}{\kappa}\right)\cos\phi_0 \\ Y_c &\equiv y_0 + \left(d\rho + \frac{\alpha}{\kappa}\right)\sin\phi_0\,. \end{cases} \tag{2.43}$$

方程式 2.42 と共に方程式 2.43 は状態ベクトルに対する伝播関数 $\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})$ を定義する。

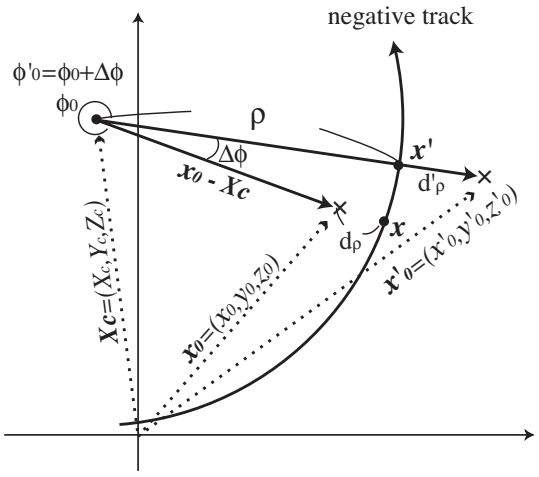

Figure 2.3: track パラメータと基準点の関係

Propagator Matrix: $\boldsymbol{F}_{k-1}$

方程式 2.42 を $\boldsymbol{a} = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$ に対して微分することで、propagator matrix $\boldsymbol{F}_{k-1}$:

$$\boldsymbol{F}_{k-1} \equiv \left(\frac{\partial \boldsymbol{a}'}{\partial \boldsymbol{a}}\right) = \begin{pmatrix} \frac{\partial d'_\rho}{\partial \boldsymbol{a}} \\ \frac{\partial \phi'_0}{\partial \boldsymbol{a}} \\ \frac{\partial \kappa'}{\partial \boldsymbol{a}} \\ \frac{\partial d'_z}{\partial \boldsymbol{a}} \\ \frac{\partial \tan\lambda'}{\partial \boldsymbol{a}} \end{pmatrix}$$

を計算することができる。ここで κ と $\tan\lambda$ は基準点の変更によって変化せず、よって偏導関数の計算は自明なものとなる。d'_ρ と d'_z もまた明白であるので、偏導関数の計算は ϕ'_0 から始める。

$\underline{\partial\phi'_0/\partial\boldsymbol{a}}$

新しい基準点の座標 x'_0、 y'_0 と z'_0 はパラメータではないが、任意に選んだ定数である。式 2.42 から、変数 ω に対する ϕ'_0 の偏導関数は

$$\frac{\partial\phi'_0}{\partial\omega} = \cos^2\phi'_0 \left(\frac{\frac{\partial Y_c}{\partial\omega}}{X_c - x'_0} - \tan\phi'_0 \frac{\frac{\partial X_c}{\partial\omega}}{X_c - x'_0} \right) \tag{2.44}$$

となる。ここで (X_c, Y_c) は式 2.43 中の (x'_0, y'_0) と

$$\begin{cases} X_c \equiv x'_0 + \left(d'_\rho + \frac{\alpha}{\kappa'}\right)\cos\phi'_0 \\ Y_c \equiv y'_0 + \left(d'_\rho + \frac{\alpha}{\kappa'}\right)\sin\phi'_0 \end{cases}$$

のような関係にある。結果として $\kappa' = \kappa$ より、

$$\begin{cases} X_c - x'_0 = \left(d'_\rho + \frac{\alpha}{\kappa}\right)\cos\phi'_0 \\ Y_c - y'_0 = \left(d'_\rho + \frac{\alpha}{\kappa}\right)\sin\phi'_0 \end{cases} \tag{2.45}$$

となる。微分するために 式 2.43 を用いて $\partial X_c/\partial \boldsymbol{a}$ と $\partial Y_c/\partial \boldsymbol{a}$ を計算する必要がある。

$$\begin{cases} \dfrac{\partial X_c}{\partial d_\rho} &= \cos\phi_0 \\ \dfrac{\partial X_c}{\partial \phi_0} &= -\left(d'_\rho + \dfrac{\alpha}{\kappa}\right)\sin\phi_0 \\ \dfrac{\partial X_c}{\partial \kappa} &= -\dfrac{\alpha}{\kappa^2}\cos\phi_0 \\ \dfrac{\partial X_c}{\partial d_z} &= 0 \\ \dfrac{\partial X_c}{\partial \tan\lambda} &= 0 \end{cases} \qquad \begin{cases} \dfrac{\partial Y_c}{\partial d_\rho} &= \sin\phi_0 \\ \dfrac{\partial Y_c}{\partial \phi_0} &= \left(d'_\rho + \dfrac{\alpha}{\kappa}\right)\cos\phi_0 \\ \dfrac{\partial Y_c}{\partial \kappa} &= -\dfrac{\alpha}{\kappa^2}\sin\phi_0 \\ \dfrac{\partial Y_c}{\partial d_z} &= 0 \\ \dfrac{\partial Y_c}{\partial \tan\lambda} &= 0 \end{cases} \tag{2.46}$$

これより求めたい偏導関数は、必要に応じて方程式 2.44 中の式 2.45、式 2.46 を代入することでただちに得られる。結果として $\frac{\partial \phi'_0}{\partial \mathbf{a}}$ は κ の符号を無視すると、

$$\begin{cases} \dfrac{\partial \phi'_0}{\partial d_\rho} &= -\left(d'_\rho + \dfrac{\alpha}{\kappa}\right)^{-1}\sin(\phi'_0 - \phi_0) \\ \dfrac{\partial \phi'_0}{\partial \phi_0} &= \left(d_\rho + \dfrac{\alpha}{\kappa}\right)\left(d'_\rho + \dfrac{\alpha}{\kappa}\right)^{-1}\cos(\phi'_0 - \phi_0) \\ \dfrac{\partial \phi'_0}{\partial \kappa} &= \dfrac{\alpha}{\kappa^2}\left(d'_\rho + \dfrac{\alpha}{\kappa}\right)^{-1}\sin(\phi'_0 - \phi_0) \\ \dfrac{\partial \phi'_0}{\partial d_z} &= 0 \\ \dfrac{\partial \phi'_0}{\partial \tan\lambda} &= 0 \end{cases} \tag{2.47}$$

となる。

$\underline{\partial d'_\rho/\partial \boldsymbol{a}}$

ϕ'_0 についても同様に、式 2.42 から、以下の変数 ω について d'_ρ の偏導関数

$$\begin{aligned} \frac{\partial d'_\rho}{\partial \omega} &= \frac{\partial X_c}{\partial \omega}\cos\phi'_0 - (X_c - x'_0)\sin\phi'_0\frac{\partial \phi'_0}{\partial \omega} \\ &+ \frac{\partial Y_c}{\partial \omega}\sin\phi'_0 + (Y_c - y'_0)\cos\phi'_0\frac{\partial \phi'_0}{\partial \omega} - \frac{\partial}{\partial \omega}\frac{\alpha}{\kappa} \end{aligned}$$

が得られる。この方程式と式 2.46) と式 2.47 を用いることで、$\boldsymbol{a}$ に対する d'_ρ の偏微分を実行することができ、κ の符号を無視すると、

$$\begin{cases} \dfrac{\partial d'_\rho}{\partial d_\rho} &= \cos(\phi'_0 - \phi_0) \\ \dfrac{\partial d'_\rho}{\partial \phi_0} &= \left(d_\rho + \dfrac{\alpha}{\kappa}\right)\sin(\phi'_0 - \phi_0) \\ \dfrac{\partial d'_\rho}{\partial \kappa} &= \dfrac{\alpha}{\kappa^2}\left(1 - \cos(\phi'_0 - \phi_0)\right) \\ \dfrac{\partial d'_\rho}{\partial d_z} &= 0 \\ \dfrac{\partial d'_\rho}{\partial \tan\lambda} &= 0 \end{cases} \tag{2.48}$$

を得る。

他の偏微分

κ' と $\tan\lambda'$ の偏導関数は自明で、

$$\begin{cases} \dfrac{\partial\kappa'}{\partial d_\rho} &= \dfrac{\partial\kappa'}{\partial\phi_0} = \dfrac{\partial\kappa'}{\partial d_z} = \dfrac{\partial\kappa'}{\partial\tan\lambda} = 0 \\ \dfrac{\partial\kappa'}{\partial\kappa} &= 1 \\ \dfrac{\partial\tan\lambda'}{\partial d_\rho} &= \dfrac{\partial\tan\lambda'}{\partial\phi_0} = \dfrac{\partial\tan\lambda'}{\partial\kappa} = \dfrac{\partial\tan\lambda'}{\partial d_z} = 0 \\ \dfrac{\partial\tan\lambda'}{\partial\tan\lambda} &= 1 \end{cases} \tag{2.49}$$

となる。

式 2.47 を用いると、d'_z の偏導関数もまた簡単に求まり、κ の符号を無視すると

$$\begin{cases} \dfrac{\partial d'_z}{\partial d_\rho} &= \dfrac{\alpha}{\kappa}\left(d'_\rho + \dfrac{\alpha}{\kappa}\right)^{-1}\tan\lambda\sin(\phi'_0-\phi_0) \\ \dfrac{\partial d'_z}{\partial\phi_0} &= \dfrac{\alpha}{\kappa}\tan\lambda\left(1-\left(d_\rho+\dfrac{\alpha}{\kappa}\right)\left(d'_\rho+\dfrac{\alpha}{\kappa}\right)^{-1}\cos(\phi'_0-\phi_0)\right) \\ \dfrac{\partial d'_z}{\partial\kappa} &= \dfrac{\alpha}{\kappa^2}\tan\lambda\left(\phi'_0-\phi_0-\dfrac{\alpha}{\kappa}\left(d'_\rho+\dfrac{\alpha}{\kappa}\right)^{-1}\sin(\phi'_0-\phi_0)\right) \\ \dfrac{\partial d'_z}{\partial d_z} &= 1 \\ \dfrac{\partial d'_z}{\partial\tan\lambda} &= -\dfrac{\alpha}{\kappa}(\phi'_0-\phi_0) \end{cases} \tag{2.50}$$

となる。

Process Noise: $\boldsymbol{Q}_{k-1}$

ここでは、点 $k-1$ と点 k の間のプロセスノイズ $\boldsymbol{w}_{k-1}$ として多重クーロン散乱と、エネルギー損失について考察する。

式 2.39 によって与えられる漸化式の中で、プロセスノイズの効果は、その分散 $\boldsymbol{Q}_{k-1}$ が

$$\boldsymbol{C}_k^{k-1} = \boldsymbol{F}_{k-1}\,\boldsymbol{C}_{k-1}\,\boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1}$$

として現れているだけである。当然、プロセスノイズの分散は点 $k-1$ と点 k の間の物質の分布に依存するのだが、単純にするため、二点間の距離が十分に小さく、点 $k-1$ と点 k の間の物質がその中間 $(m; k-1<m<k)$ にある無限に薄い測定面に集中していると近似して、多重散乱やエネルギー損失が扱うことができると仮定する。

この薄い測定面の極限において [16]、中間点 m における helix パラメータのベクトルの共変行列 $\boldsymbol{Q}_m$ は

以下の単純な形になる。

$$\boldsymbol{Q}_m = \sigma_{MS}^2 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1+\tan^2\lambda & 0 & 0 & 0 \\ 0 & 0 & (\kappa' \tan\lambda)^2 & 0 & \kappa' \tan\lambda(1+\tan^2\lambda) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \kappa' \tan\lambda(1+\tan^2\lambda) & 0 & (1+\tan^2\lambda)^2 \end{pmatrix} \tag{2.51}$$

ここで、$\tan\lambda$ は中間点における値であるが、点 $k-1$ と点 k の間の距離が十分に短いので、点 $k-1$ における値によって近似でき、σ_{MS} は通常

$$\sigma_{MS} = \frac{0.0141}{P(\mathrm{GeV})\beta}\sqrt{X_L}\left(1+\frac{1}{9}\log_{10} X_L\right) \tag{2.52}$$

で与えられる。ここで、P、β、X_L はそれぞれ運動量、光速度を単位とした速度、放射長を単位としたときの散乱体の幅である。また κ' は、κ に対して点 $k-1$ から点 k の間のエネルギー損失に対する補正を加えたもので、

$$E = \sqrt{\frac{1+\tan^2\lambda}{\kappa^2}+M^2}, \qquad E' = \sqrt{\frac{1+\tan^2\lambda}{\kappa'^2}+M^2}$$

とすると、一般にエネルギー損失は以下の式で与えられ、

$$E'-E = KQ^2\frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_e c^2\beta^2\gamma^2 T_{max}}{I^2}-\beta^2-\frac{\delta}{2}\right] \tag{2.53}$$

これより κ' を求めることができる。ここで、

$$T_{max} = \frac{2m_e c^2\beta^2\gamma^2}{1+2\gamma m_e/M+(m_e/M)^2}$$

であり、M は粒子の質量、$K = 4\pi N_A r_e^2 m_e c^2$、$N_A$ はアボガドロ数、Z は粒子の原子番号、A は粒子の質量数、m_e は電子の質量、$\gamma = 1/\beta$、δ は密度補正項である。

目的とするプロセスノイズ $\boldsymbol{Q}_{k-1}$ に対する共変行列は、中間点 m から点 k へ基準点を動かすことによって得られる。

$$\boldsymbol{Q}_{k-1} = \boldsymbol{F}_{m,k}\,\boldsymbol{Q}_m\,\boldsymbol{F}_{m,k}^T \tag{2.54}$$

ここで、$\boldsymbol{F}_{m,k}$ は、$\boldsymbol{a} = \boldsymbol{a}_m$ と $\boldsymbol{a}' = \boldsymbol{a}_{k-1}^k$ より、式 2.44、式 2.48、式 2.49、式 2.50 によって与えられる、中間点 m から点 k への、状態ベクトルの共変行列に対する propagator matrix[6]である。

点 $k-1$ と点 k の間の track の断片が、上記で述べた薄い測定面による近似を適用するのに十分短くない場合には、さらに N 分割することで、以下の和をとればよい。

$$\boldsymbol{Q}_{k-1} = \sum_{s=1}^{N-1}\boldsymbol{F}_{m_s,k}\,\boldsymbol{Q}_{m_s}\,\boldsymbol{F}_{m_s,k}^T \tag{2.55}$$

[6] $\boldsymbol{F}_{k-1} \equiv \boldsymbol{F}_{k-1,k}$ とする。

2.3.3 Measurement Equation: $\boldsymbol{h}_k$, $\boldsymbol{H}_k$, and $\boldsymbol{V}_k$

系の方程式に関する式を導くために、測定方程式:

$$\boldsymbol{m}_k \quad = \quad \boldsymbol{h}_k(\bar{\boldsymbol{a}}_k) + \boldsymbol{\epsilon}_k$$

に関する式の導出を行う。ここで、測定ベクトル $\boldsymbol{m}_k$ は、$\boldsymbol{\epsilon}_k$ で表されるプロセスノイズの影響を受けた、点 k において位置検出器によって測定された座標の集合である。

必要なのは、点 k における実際の状態ベクトルの関数として、点 k での正しい hit 座標を与える projector $\boldsymbol{h}_k(\boldsymbol{a})$ と、その track パラメータの微分である projector matrix $\boldsymbol{H}_k$ と、測定誤差に対する共変行列 $\boldsymbol{V}_k$ である。

実際には、荷電粒子の位置は、飛跡に沿った不連続点として検出される。ここで位置を抽出する (仮想的な) 面を考えると、それぞれの測定面は点 k における測定点に対応しており、飛跡に沿って、その交点、hit 点を記録する。hit 点の座標は測定ベクトル $\boldsymbol{m}_k$ を構成する。

Projector: $\boldsymbol{h}_k(\boldsymbol{a})$

状態ベクトルの関数として測定面上の測定ベクトルを予測するために、まず、$S_k(\boldsymbol{x}) = 0$ によって与えられた測定面と、$\boldsymbol{a}$ で規定された track の交点 $\boldsymbol{x}_k(\boldsymbol{a})$ を与える方程式

$$\boldsymbol{x}_k \quad = \quad \boldsymbol{x}_k(\boldsymbol{a}) \quad = \quad \boldsymbol{x}(\phi_k(\boldsymbol{a}), \boldsymbol{a}) \tag{2.56}$$

と、もう一つ交点 $\boldsymbol{x}_k(\boldsymbol{a})$ を測定ベクトルに変換する方程式

$$\boldsymbol{m}_k = \boldsymbol{m}_k(\boldsymbol{x}_k) \tag{2.57}$$

が必要となる。ここで、$\boldsymbol{x} = \boldsymbol{x}(\phi, \boldsymbol{a})$ は helical track の場合に式 2.40 によって与えられる飛跡の方程式であり、$\phi_k(\boldsymbol{a})$ は点 k に対応する測定面と hit 点との方位角である。

これら二つの方程式から、projector $\boldsymbol{h}_k(\boldsymbol{a})$ は

$$\begin{aligned} \boldsymbol{h}_k(\boldsymbol{a}) \quad &= \quad \boldsymbol{m}_k(\boldsymbol{x}_k(\boldsymbol{a})) \\ &= \quad \boldsymbol{m}_k(\boldsymbol{x}(\phi_k(\boldsymbol{a}), \boldsymbol{a})) \end{aligned} \tag{2.58}$$

と定義できる。

Projector Matrix: $\boldsymbol{H}_k$

これらの方程式の具体的な形は測定面の性質に依存しており、検出器に特有なものであるが、$\boldsymbol{a}$ に関する $\boldsymbol{h}_k$ の微分の一般的な式を

$$\boldsymbol{H}_k \equiv \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{a}} = \left(\frac{\partial \boldsymbol{m}_k}{\partial \boldsymbol{x}} \right) \left(\frac{\partial \boldsymbol{x}(\phi_k(\boldsymbol{a}), \boldsymbol{a})}{\partial \boldsymbol{a}} \right) \tag{2.59}$$

と書き下すことが可能である。ここで、右辺の二つ目の要素は、

$$\frac{\partial \boldsymbol{x}(\phi_k(\boldsymbol{a}), \boldsymbol{a})}{\partial \boldsymbol{a}} = \frac{\partial \boldsymbol{x}}{\partial \phi_k} \frac{\partial \phi_k}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}} \tag{2.60}$$

で与えられ、この内 $\partial \boldsymbol{x}/\partial\phi_k$ と $\partial \boldsymbol{x}/\partial \boldsymbol{a}$ は、飛跡の方程式のみによって決定される。

一方、$\partial \boldsymbol{m}_k/\partial \boldsymbol{x}$ および $\frac{\partial\phi_k}{\partial \bar{\boldsymbol{a}}}$ は測定面の形状 $S_k(\boldsymbol{x}) = 0$ と、その系での座標の配置によって決定され、どちらも応用する検出器に依存する。

そこで、以下に $\partial\phi_k/\partial \boldsymbol{a}$ を計算する。ϕ_k の方程式

$$S_k(\boldsymbol{x}(\phi_k, \boldsymbol{a})) = 0. \tag{2.61}$$

から始めると、track パラメータベクトルの両辺を微分することで

$$0 = \frac{\partial S_k}{\partial \boldsymbol{a}} = \frac{\partial S_k}{\partial \boldsymbol{x}} \left(\frac{\partial \boldsymbol{x}}{\partial \phi_k} \frac{\partial \phi_k}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}} \right) \tag{2.62}$$

を得る。ここで

$$\frac{\partial \phi_k}{\partial \boldsymbol{a}} = -\left(\frac{\partial S_k}{\partial \boldsymbol{x}}\right)\left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}}\right) \Big/ \left(\frac{\partial S_k}{\partial \boldsymbol{x}}\right)\left(\frac{\partial \boldsymbol{x}}{\partial \phi_k}\right) \tag{2.63}$$

であった。$\partial S_k/\partial \boldsymbol{x}$ の測定面の形状みから、$\partial \boldsymbol{x}/\partial\phi$、$\partial \boldsymbol{x}/\partial \boldsymbol{a}$ は飛跡の方程式のみから決定するということが重要である。

式 2.61 が解析的に解けるかどうかは、測定面の形状 $S_k(\boldsymbol{x}) = 0$ に依存しており、円筒や平面のような特に単純な面を省いて、単純な形でさえも、解析的に解くのは難しいことがしばしばである。このような場合、方程式を解くためにニュートン法に頼ることになる。近似解 ϕ_n を考えると、

$$0 = S_k(\boldsymbol{x}(\phi, \boldsymbol{a})) \;\simeq\; S_k(\boldsymbol{x}(\phi_n, \boldsymbol{a})) + \frac{\partial S_k}{\partial \boldsymbol{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \phi_n} \cdot (\phi - \phi_n) \tag{2.64}$$

のように、方程式をテイラー展開することができる。これから、以下の漸化式を得る。

$$\phi_{n+1} = \phi_n - \frac{S_k(\boldsymbol{x}(\phi_n, \boldsymbol{a}))}{\left(\frac{\partial S_k}{\partial \boldsymbol{x}}\right)_n \cdot \left(\frac{\partial \boldsymbol{x}}{\partial \phi}\right)_n} \tag{2.65}$$

限り繰り返しこの漸化式を用いることで ϕ_n は収束する。[7] もし方程式中の測定面が現在の基準点に近接しているならば、通常 $\phi_1 = 0$ から始めて僅かな反復で収束する。

Measurement Noise: $\boldsymbol{V}_k$

Kalman filter に必要な式をそろえるには、この場合点 k における位置検出器の分解能を表す誤差行列の共変行列が必要である。測定ベクトルの残差は

$$\Delta \boldsymbol{m}_k \quad \equiv \quad \boldsymbol{m}_k - \boldsymbol{h}_k(\bar{\boldsymbol{a}}_k)$$

で示され、共変行列を

$$\boldsymbol{V}_k \equiv \left\langle \Delta \boldsymbol{m}_k \; \Delta \boldsymbol{m}_k^T \right\rangle \tag{2.66}$$

で定義することができる。もし点 k における測定面上で二つの座標が測定され、その二つの座標が統計的に独立しているならば、共変行列を以下の形式で表され、

$$\boldsymbol{V}_k \quad = \quad \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix} \tag{2.67}$$

ここで σ_1 と σ_2 はそれぞれ座標の分解能である。これらの分解能は一般に hit の位置や track の角度等に依存してるが、track fitting 中に $\boldsymbol{V}_k$ を共変行列として扱える。

[7] 特に、式 2.65 の第二項の分母には、減衰係数 $(1 + \Lambda)$ を導入する必要がある。ここで $\Lambda \ll 1$ であり、$|S_k|$ が増加するなら Λ に 1 より大きい係数をかけ、減少するなら同じ 係数で割る。

2.4 C++による実装

2.4.1 設計概念

加速器に用いられる飛跡検出器は通常、反応点検出器 (VTX)、中間飛跡検出器 (IT)、中央飛跡検出器 (CT) 等、異なる形状や座標系をもつ様々な構成要素から成り立っているので、Kalman-filter で track fitting を行うソフトウェアパッケージは、いかなる形状、座標系の測定面に対しても対応できなでればならない。ミューオン検出器のような外側にある飛跡検出器へ track を外挿することが可能であるためには、パッケージが、測定点ごとの磁場の変化に対応できることが望ましい。ユーザの最小限の実装でこれらの要求を満たすために、C++を用いて、そのオブジェクト指向の特徴を最大限に生かして開発することにした。また、データの永続化 (オブジェクト I/O) のために ROOT の自動スキーマ展開を使用した。

Kalman filter は汎用的な手順で定義されており、track fitting だけでなくより広範囲に適用することができる。このことは、汎用的な Kalman filter のアルゴ リズムを実装する抽象基底クラスライブラリ (KalLib) の必要性を示唆している。抽象基底クラス KalLib から純粋仮想関数を track fitting 用に継承することで、Kalman filter ベースで track fitter を行うクラスライブラリ (KalTrackLib) を実現することができる。しかしながら、KalTrackLib は上記の指針に従って、特定の形状や座標系をもつ測定面に依存するべきではない。よって、ジオメトリライブラリ (GeomLib) として、track の形状 (helix、直線、...) や面構造 (円筒、双極面、平面、...) を提供するジオメトリクラスを分けることにする。これら三つのライブラリ、KalLib、KalTrackLib、GeomLib は、LEDA (Library Extention for Data Analysis) 又は KalTest (Kalman filter Test bench) として提供される。

この方法で、ユーザが実装すべきクラスの数を以下の三つに最小化することができる。

- MeasLayer: 測定面の抽象クラス `TVMeasLayer` と GeomLib にある形状クラスから多重継承した測定面クラス。
- KalDetector: ユーザが定義した MeasLayer を保持する配列クラス。検出器の物質分布もここで定義する。
- Hit: MeasLayer で定義した座標ベクトルクラス。

また、粒子の飛跡に沿った磁場の種類によって track の形状を変化させることも可能であることも記しておく。

2.4.2 KalLib: Kalman Filter のための基底クラス

まず、一般的な Kalman filter の工程を表現する基底クラスライブラリ (KalLIb) の実装から述べる。

図 2.4 は KalLib の設計概念を示すダイアグラムである。`TVKalSystem` クラスは Kalman filter を行う系を表す抽象クラスであり、`TVKalSite` の `TObjArray`(ROOT のオブジェクト配列クラス) である。`TVKalSite` は測定点に対応しており、prediction、filtering、smooting がなされた各状態ベクトルに対応する `TVKalState` の `TObjArray` である。言い換えれば、`TVKalState` は状態ベクトルを表しており、`TMatrixD`(ROOT のクラス) を継承した、$p \times 1$ の `TKalMatrix` である。これらの抽象クラスはデータメンバとして $\boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1})$、

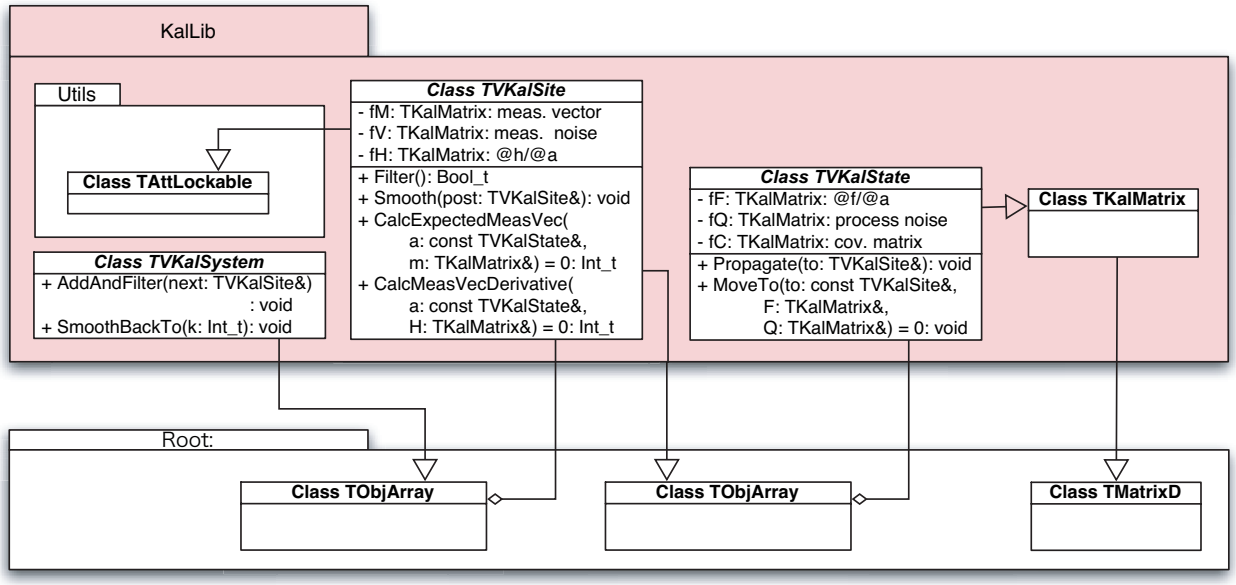

Figure 2.4: KalLib: Kalman filter のための基底クラス

$\boldsymbol{F}_{k-1}$、$\boldsymbol{Q}_{k-1}$、$\boldsymbol{m}_k$、$\boldsymbol{H}_k$、$\boldsymbol{V}_k$ と、それらを評価する純粋仮想関数を保持している。以下に、これらの抽象クラスと、その主なデータメンバとメンバ関数の役割について説明する。

TVKalSystem (TObjArray を継承: 測定点の配列)

`TVKalSystem` クラスは `TObjArray` を継承しており、測定点の配列として振舞う。また、いなかる Kalman filter の系をも表す抽象基底クラスであり、異なる測定点の間の関係を扱える唯一のクラスである。

データメンバ:

```
TVKalSite * fCurSitePtr; // 現在の測定点へのポインタ
Double_t    fChi2;       // χ^2 の値
```

メンバ関数:

```
void AddAndFilter(TVKalSite & next);
```

次の測定点 (`next`) の filtering を行い、基底クラスの `TObjArray` に加える。

```
void SmoothBackTo(Int_t k);
```

最後の点から点 k まで smooting を行う。

TVKalSite (TObjArray を継承: States の配列: 測定点)

`TVKalSite` クラスは `TObjArray` を継承しており、各段階 (prediction, filtering, smoothing) における状態ベクトルを保持する配列として振舞う。また、`TVKalSystem` には必要であるが filtering には関係しないク

ラスのライブラリである、Utils にある TAttLockable からも継承する。測定点を表す抽象基底クラスであり、データメンバとして、この点における観測で得られた情報と、その時点での状態ベクトルを持つ。

データメンバ:

```
TKalMatrix fM; // 測定ベクトル
TKalMatrix fV; // 測定誤差行列
TKalMatrix fH; // projector matrix.
```

メンバ関数:

```
Int_t CalcExpectedMeasVec(const TVKalState & a, TKalMatrix & h) = 0;
```

与えられた状態ベクトル a に対して期待される測定ベクトル h を計算する projector を表す純粋仮想関数。問題にする系に依存するため派生クラスで実装される。

```
Int_t CalcMeasVecDerivative(const TVKalState & a, TKalMatrix & H) = 0;
```

引数の状態ベクトル a に対する projector の微分である、projector matrix H を表す純粋仮想関数。問題にする系に依存するため派生クラスで実装される。

```
Bool_t Filter();
```

式 2.39 で表現される Kalman filter の一般的なアルゴリズムを実装する。自身で保持している前の測定点、測定ベクトルとその誤差行列から予測した状態ベクトルを用いて、TVKalSite のインスタンスを filtering を行い、filtering した状態ベクトル $\boldsymbol{a}_k$ と、その共変行列 $\boldsymbol{C}_k$ を計算して、それらを TVKalState として基底クラス TObjArray に加える。

```
void Smooth(TVKalSite & post);
```

式 2.34 と式 2.35 で与えられる smoothing の一般的なアルゴリズムを実装する。自身で保持している情報と次の測定点 post を用いて、TVKalSite のインスタンスの smoothing を行い、その smoothing された状態ベクトル $\boldsymbol{a}_k^n$ とその共変行列 $\boldsymbol{C}_k^n$ を計算して、それらを TVKalState として基底クラス TObjArray に加える。

TVKalState (TKalMatrix を継承: 状態ベクトル)

TVKalState クラスは、TMatrixD を継承した $p \times 1$ の TKalMatrix であり、基底クラスの TKalMatrix に状態ベクトルを保持する抽象基底クラスである。

データメンバ:

```
TVKalSite  * fSitePtr;   // 対応する測定点へのポインタ
TKalMatrix   fF;         // propagator matrix
TKalMatrix   fQ;         // プロセスノイズの共変行列
TKalMatrix   fC;         // 状態ベクトルの共変行列
```

メンバ関数:

```
TVKalState & MoveTo(const TVKalSite & to, TKalMatrix & F,
                          TKalMatrix & Q) const = 0;
```

TVKalState の自身のインスタンスを目的の測定点 to へ移し、対応する propagator matrix とプロセスノイズ行列と共に参照として自身を返却する、純粋仮想関数。問題にする系に依存するため派生クラスで実装される。

```
void Propagate(TVKalSite & to);
```

TVKalState の自身のインスタンスを目的の測定点 to へ伝播させ、ここでの prediction した状態ベクトルの共変行列を計算を行い、(式 2.9 と 2.13 の一般的なアルゴリズムを実装する。) 関数 MoveTo() によって状態ベクトルとその共変行列を伝播する。

2.4.3 Kaltracklib: Kalman-Filter ベースの Track Fitting ライブラリ

ここでは、KalLib の TVKalSystem、TVKalSite、TVKalState、三つの基底クラスの純粋仮想関数を、派生クラスで track fitting に固有な関数として実装することで、Kalman filter の手法を track fitting に応用する。これらの派生クラスは、ほかの追加のクラスと共に、KalTrackLib と呼ばれるクラスライブラリを構成している。図 2.5 は KalTrackLib の設計概念を示すダイアグラムである。KalTrackLib で定義されるくらすは track fitting に関する共通のアルゴリズムを実装するが、問題にする飛跡検出器の特定の形状や座標系には拠らない。以下に、KalTrackLib のクラスの、基本的な特徴と、エンドユーザに関係する主なデータメンバとメンバ関数について説明する。

TKalTrack: TVKalSystem を継承: トラック

track とは hit 点の集合である。故に、測定点の配列である TVKalSystem から継承したクラスのインスタンスを、track として定義することは至極当然である。TVKalSystem は純粋仮想関数を持っていないので、オーバーライドすべきメンバ関数なない。

メンバ関数:

```
TKalTrack();
```

TKalTrack のインスタンスを作成するコンストラクタ

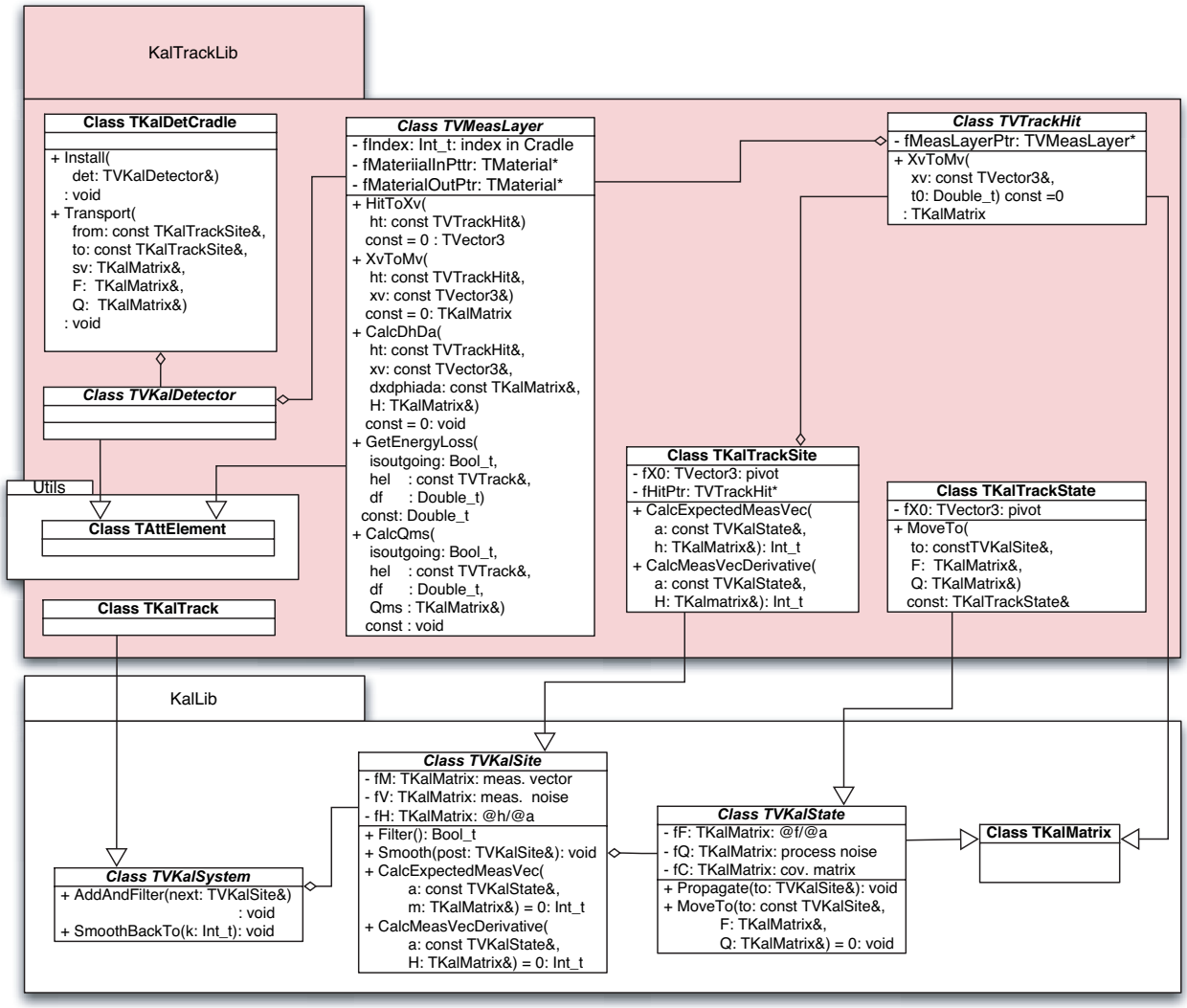

Figure 2.5: KalTrackLib: Kalman-filter ベースの track fitting クラスライブラリ

TKalTrackSite: TVKalSite を継承: 測定点

track fitting において、測定点は hit 点を規定し、その座標は測定ベクトルを構成する。`TKalTrackSite` クラスは測定点の役目を果たすようにデザインされており、基底クラス `TVKalSite` の純粋仮想関数を実装し、データメンバの一つとして想定ベクトル `TVTrackHit` のポインタを保持する。

データメンバ:

```
TVTrackHit * fHitPtr; // hit オブジェクトに対応するポインタ
TVector3     fX0;     // hit 点に対する基準点
```

メンバ関数:

```
TKalTrackSite(const TVTrackHit & ht);
```

hit オブジェクトからインスタンスを作成するコンストラクタ

```
Int_t CalcExpectedMeasVec(const TVKalState & a, TKalMatrix & h);
```

与えられた状態ベクトル a に対して期待される測定ベクトルを計算するための projector を規定する基底クラスの純粋仮想関数この関数は**??**節のアルゴリズムと、2.3.3 節のニュートン法を具体化している。

```
Int_t CalcMeasVecDerivative(const TVKalState & a, TKalMatrix & H);
```

この関数は 2.3.3 節のアルゴリズムを具体化するため、projector の引数の状態ベクトル a に対する微分である、projector matrix H を規定する基底クラスの純粋仮想関数の実装する。

CalcExpectedMeasVec() や CalcMeasVecDerivative() の一般的なアルゴリズムの具体化に見られる $\partial \boldsymbol{x}/\partial \phi_k$ や $\partial \boldsymbol{x}/\partial \boldsymbol{a}$ のような微分関数は、トラックの形状 (飛跡の方程式) に依存する。これらのインターフェースは、後に定義する、GeomLib にある基底クラス TVTrack で規定され、helix や直線のような特定の track の形状を表す派生クラスで実装される。位置ベクトルに対する測定ベクトルの微分 $\partial \boldsymbol{m}_k/\partial \boldsymbol{x}$ は、言い換えれば、問題にする測定面の形状と座標系によって決定される。この計算をする関数のインターフェイスは、後に定義する抽象基底クラス TVMeasLayer で規定され、特定の測定面を規定する派生クラスによって実装される。

TKalTrackState : TVKalState を継承: トラックパラメータベクトル

track fitting において、状態ベクトルはその一部としてトラックパラメータを保持する。TKalTrackState はトラックパラメータベクトルの役目を果たすようにデザインされており、基底クラス TVKalState の純粋仮想関数 MoveTo() を実装する。

データメンバ:

```
TVector3    fX0;     // 基準点
```

メンバ関数:

```
TKalTrackState & MoveTo(const TVKalSite  & to,
                        TKalMatrix & F,
                        TKalMatrix & Q) const;
```

TKalTrackState の自身のインスタンスを目的の点 to へ移し、対応する propagator matrix とプロセスノイズ行列と共に、自身を参照として返却する。

プロセスノイズの計算は現在点と次の点の物質分布に依存する。下記に示すように、飛跡検出器の構造は、抽象基底クラス TVKalDetector から派生した具体的なクラスと、その要素である抽象基底クラス TVMeasLayer から派生した測定面とによって定義される。測定面に関する局所的な物質の情報は TVMeasLayer クラスによって保持される。もし必要ならば、対応する KalDetector をそのフォルダクラス TKalDetCradle にインストールすることで、複数の飛跡検出装置から成る飛跡検出器を構成することができる。その結果、TKalDetCradle のインスタンスは検出器の構成の全てを知ることになる。故に、関数 MoveTo() は点 to へ移る際、物質分布に依存するプロセスノイズを扱うために関数 TKalDetCradle::Transport() を呼ぶ。また、関数 TKalDetCradle::Transport() は、track の形状は測定点から測定点へ移る際に局所的な磁場に従って変化することが可能なので、GomLib にある抽象クラス TVTrack の純粋仮想関数 MoveTo() を呼ぶことになる。

TVTrackHit: TKalMatrix を継承; 測定ベクトル

track fitting において、hit 点は測定ベクトルの役割を果たし、測定面と粒子の飛跡との交点における座標を与える。TVTrackHit クラスは、hit 点や、純粋仮想関数 XvToMv() を通して、その三次元の位置ベクトルを測定面の座標軸へ変換する coordinate projector の、インターフェイスを規定する基底クラスとしてデザインされている。

データメンバ:

```
Double_t      fBfield;       // 磁場
TVMeasLayer * fMeasLayerPtr; // 対応する測定面へのポインタ
```

メンバ関数:

```
TKalMatrix XvToMv(const TVector3 & xv, Double_t t0) const = 0;
```

> 時刻 t0 における、hit 点の座標ベクトル xv から測定面の座標軸へ座標変換する、coordinate projector のインターフェースを規定する純粋仮想関数 coordinate projector は問題にする測定面の座標系に依存するため、派生クラスによって実装される。

新たな飛跡検出器を導入するためには、TVTrackHit を継承した具体的な hit 点のクラスを定義し、その純粋仮想関数 XvToMv() を実装する。

TVMeasLayer: 測定面

前述したとおり、飛跡検出器は測定面の集合とみなすことができる。それぞれの測定面は、最低でも共通のインターフェイスとして、projector $\boldsymbol{h}_k$ と projector matrix $\boldsymbol{H}_k$ と、節 2.3.3 や節 2.3.3 で説明されている一般的なアルゴリズムを用いてそれらを計算するための関数を、用意しなければならない。これらの共通なインターフェイスは抽象基底クラスである、KalTrackLib の TVMeasLayer と GeomLib の TVSurface によって規定される。故に具体的な測定面は、TVMeasLayer と、TVSurface から派生した円筒 (TCylinder) や双極面 (THype) や平面 (TPlane) のような具体的な表面構造との多重継承によって作成され、その結果特定の測定面特有の形状や座標系を具体化する。TVMeasLayer クラスは TAttElement を継承しており、下記に述べる TVKalDetector の要素として振舞う。

データメンバ:

```
TMaterial    * fMaterialInPtr;        // 内側の物質へのポインタ
TMaterial    * fMaterialOutPtr;       // 外側の物質へのポインタ
Int_t          fIndex;                // TKalDetCradle に対する index
Bool_t         fIsActive;             // 測定面が active か否かのフラグ
```

また、測定面は座標測定面である (active である) か、単に異なる物質との間の境界である (active でない) かのどちらかで定義される。

メンバ関数:

```
TKalMatrix XvToMv(const TVTrackHit & ht, const TVector3 & xv) const = 0;
```

式 2.57 に対応する純粋仮想関数で、hit 点の三次元ベクトル xv から座標軸への変換を行い、結果として測定ベクトルを返却する。

```
TVector3 HitToXv(const TVTrackHit & ht) const = 0;
```

測定ベクトル ht を hit 点の三次元ベクトルへ変換する純粋仮想関数。次の基準点を計算するのに用いられる。

```
void CalcDhDa(const TVTrackHit & ht,
              const TVector3   & xv,
              const TKalMatrix & dxphiada,
                    TKalMatrix & H) const = 0;
```

2.3.3 節で与えられる一般的なアルゴリズムを仮定して得られる引数 dxphiada= $\partial \boldsymbol{x}(\phi(\boldsymbol{a}), \boldsymbol{a})/\partial \boldsymbol{a}$ を用いて、hit 点 xv における projector matrix H を計算する純粋仮想関数。

```
Double_t CalcEnergyLoss(      Bool_t     isoutgoingt,
                        const TVTrack  & hel,
                              Double_t   df) const;
```

track hel と、その方位角 df と、粒子の飛ぶ向き isoutgoing から、式 2.51 で与えられるエネルギー損失を計算して返却する。

```
void CalcQms(      Bool_t       isoutgoingt,
             const TVTrack    & hel,
                   Double_t     df,
                   TKalMatrix & Qms) const;
```

track hel と、その方位角 df と、粒子の飛ぶ向き isoutgoing から、式 2.51 で与えら

れるプロセスノイズ行列 $\boldsymbol{Q}_m$ を計算する。

上記で述べたように、具体的な測定面はこの TVMeasLayer クラスと、TVSurface から派生した具体的な表面構造を持つクラスとの多重継承をする。TCylinder や THype、TPlane といった表面オブジェクトは既に track と表面との交点、その他を計算するメソッドが準備されているので、ユーザが測定面の具体的なクラスを実装するのに必要なことは、式 2.57 で与えられる座標軸への projector と projector matrix だけである。

TVKalDetector: TObjArray を継承; 検出器のコンポーネント

抽象基底クラス TVMeasLayer と、TCylinder や THype、TPlane といった具体的な表面との多重継承のインスタンスとして実装された、複数の測定面から成り立つ、VTX、IT や CT といった飛跡検出器を規定するために、TObjArray を継承した、TVKalDetector と呼ばれる抽象基底クラスを定義する。TVKalDetector クラスコンテナとしてのみ機能し、特別なメンバ関数をもっていない。TVKalDetector クラスはまた、TAttElement も継承しており、下記で述べる TKalDetCradle の要素として振舞う。

ここで、測定面が TVMeasLayer と TVSurface を継承している限り、TVKalDetector にはいかなる形状、座標系を持つ測定面に対しても Add() を行うことができることを付け加えておく。この仕組みによって、様々な形状、座標系を持つ測定面から成る測定器を、統一的に構成することが可能となる。

TKalDetCradle: TObjArray を継承; 検出器システム

VTX、IT や CT といった飛跡検出器コンポーネントの全体を表す検出器システムを規定するために、TObjArray を継承し、配列の要素として、それぞれの検出器コンポーネントの測定面を保持する、TKalDetCradle と呼ばれるクラスを定義する。TKalDetCradle の目的はビームパイプやサポートシリンダー等に対応するような測定面を含めた全ての測定面を保持することである。TKalDetCradle は検出器の物質分布の全てを知る唯一のオブジェクトであり、メンバ関数 Transport() は TKalTrackState::MoveTo() より呼ばれる。

メンバ関数:

```
TKalDetCradle();
```

TKalDetCradle のインスタンスを作成するコンストラクタ

```
void Install(TVKalDetector & det);
```

この TKalDetCradle のインスタンスのコンポーネントとして det をインストールする。

```
void Transport(const TKalTrackSite & from,
               const TKalTrackSite & to,
                     TKalMatrix    & sv,
                     TKalMatrix    & F,
                     TKalMatrix    & Q) const;
```

点 `from` から点 `to` へ状態ベクトル `sv` を移し、式 2.51 と式 2.55 で与えられる薄い測定面の散乱近似より propagator matrix `F` とプロセスノイズ `Q` を計算する。

2.4.4 GeomLib: ジオメトリクラスライブラリ

GeomLib は track 形状と測定面形状を規定するクラスで構成されるライブラリである。helix や直線といった具体的な track 形状を表現するクラスは `TVTrack` と呼ばれる抽象基底クラスから継承されなければならず、円筒や平面といった具体的な面の形状を表現するクラスは全て `TVSurface` と呼ばれる抽象基底クラスから継承されなければならない。これらの抽象基底クラスはインターフェイスの集合と見なされる。図 2.6 は GeomLib の設計概念を示すダイアグラムである。GeomLib で定義されているクラスは track と測定面、

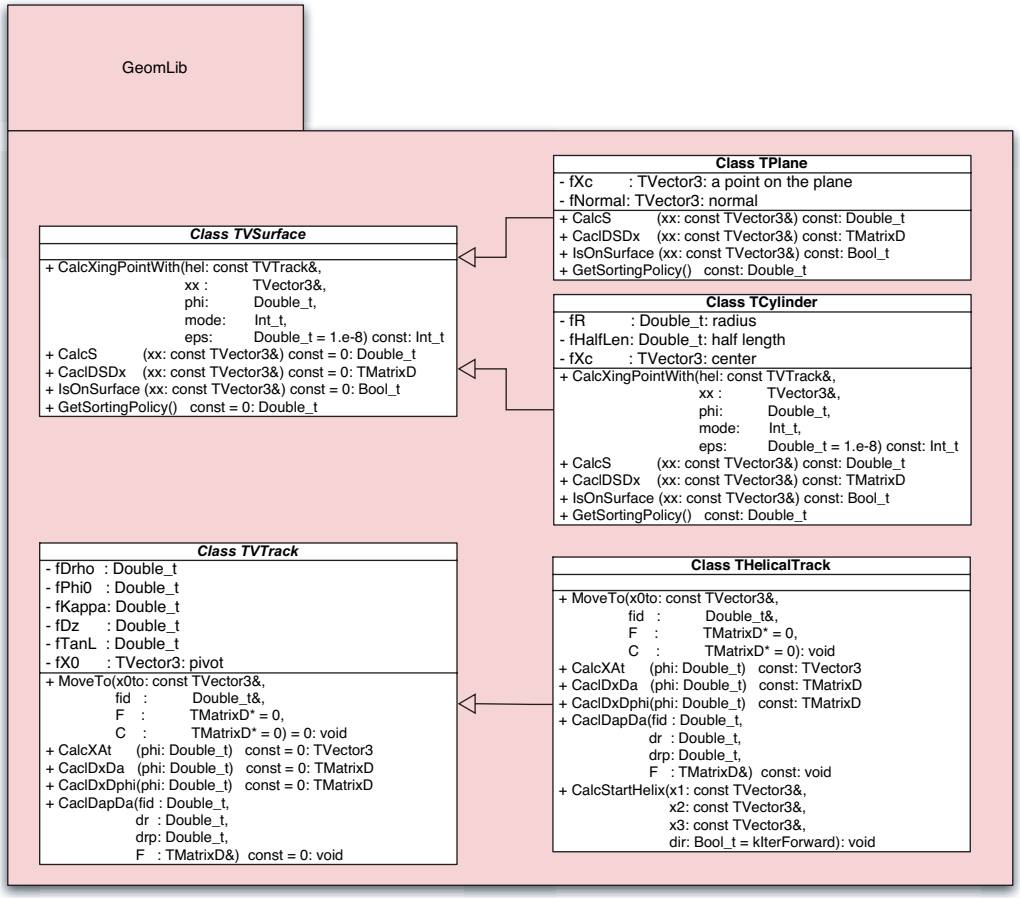

Figure 2.6: GeomLib: track 形状と測定面形状を規定するジオメトリクラスライブラリ

track の接線ベクトルや測定面の勾配等を計算する、ユーティリティメソッドを実装する。以下に、GeomLib のクラスの、基本的な特徴と、エンドユーザに関係する主なデータメンバとメンバ関数について説明する。

TVTrack Class: TVCurve を継承; トラック

TVTrack クラスはトラックパラメータ $\boldsymbol{a}$ と飛跡上の位置を規定する ϕ パラメータ[8] の関数として、飛跡の方程式のインターフェイスを定める抽象基底クラスである。

データメンバ:

```
Double_t    fDrho;        // d_ρ.
Double_t    fPhi0;        // φ_0.
Double_t    fKappa;       // κ.
Double_t    fDz;          // d_z.
Double_t    fTanL;        // tan λ.
Double_t    fAlpha;       // α ≡ 1/cB (式 2.41 参照)
```

メンバ関数:

```
void MoveTo(const TVector3 & xv0,
                  Double_t & fid,
                  TMatrixD * F = 0,
                  TMatrixD * C = 0) = 0;
```

基準点を xv0 へ動かす純粋仮想関数であり、新たな基準点に対する phi の変化分、F$\neq 0$、C$\neq 0$ であればさらに*F$= \partial \boldsymbol{a}'/\partial \boldsymbol{a}$ と新たな基準点*C における共変行列を共に計算する。

```
TVector3 CalcXAt(Double_t phi) const = 0;
```

phi における track 上の粒子の位置ベクトルを計算する純粋仮想関数

```
TMatrixD CalcDxDa(Double_t phi) const = 0;
```

$\partial \boldsymbol{x}(\phi, \boldsymbol{a})/\partial \boldsymbol{a}$ (2.3.3 節を参照) を計算し、返却する純粋仮想関数

```
TMatrixD CalcDxDphi(Double|_t phi) const = 0;
```

飛跡の接線ベクトル: $\partial \boldsymbol{x}(\phi, \boldsymbol{a})/\partial \phi$ (図 2.3.3 を参照) を計算し、返却する純粋仮想関数

TVSurface Class: TObject を継承; 表面

TVSurface クラスは全ての表面クラスの基底クラスとなるもので、2.3.3 節の一般的なアルゴリズムを必要に応じて用いることで、track との交点、与えられた hit 点における勾配等を計算するメソッドのインターフェイスを規定する。

[8] ϕ は helix track の場合基準点からの方位角。直線 track の場合は基準点からの符号付距離。

メンバ関数:

```
Int_t CalcXingPointWith(const TVTrack  & hel,
                        TVector3 & xx,
                        Double_t & phi,
                        Int_t      mode,
                        Double_t   eps = 1.e-8) const;
```

2.3.3 節のニュートン法を実装することで、track `hel` との交点 `xx` と、それに対応する交差 `eps` における位置パラメータ `phi` を計算し、交点の数を返却する。

```
Double|_t CalcS(const TVector3 & xx) const = 0;
```

$\boldsymbol{x}$ =xx における表面の方程式 2.61 の左辺 $S_k(\boldsymbol{x})$ を計算する純粋仮想関数。

```
TMatrixD CalcDSDx(const TVector3 & xx) const = 0;
```

$\boldsymbol{x}$ =xx における $S_k(\boldsymbol{x})$ の勾配 $\partial S_k/\partial \boldsymbol{x}$ を計算する純粋仮想関数。

```
Bool_t IsOnSurface(const TVector3 & xx) const = 0;
```

xx が表面上にあるかどうかを確かめる純粋仮想関数。

```
Double_t GetSortingPolicy() const = 0;
```

表面オブジェクトを内側から外側へソートするための指針を与える純粋仮想関数。

THelicalTrack: TVTrack を継承; Helical Track

`THelicalTrack` クラスは helical な track を具体化するために、`TVTrack` を継承し、その純粋仮想関数を実装する。

メンバ関数:

```
void MoveTo(const TVector3 & xv0,
                  Double_t & fid,
                  TMatrixD * F = 0,
                  TMatrixD * C = 0);
```

基準点を `xv0` へ移動させ、新たな基準点への方位角 `fid` と、F$\neq$ 0、C$\neq$ 0 ならばさらに、2.3.2 節で説明されている*F= $\partial \boldsymbol{a}'/\partial \boldsymbol{a}$ と、新たな基準点における共変行列*C を計算

する。

```
TVector3 CalcXAt(Double_t phi) const;
```

式 2.40 より、方位角 phi における helix 上の粒子の位置ベクトルを計算する。

$$\boldsymbol{x}\ (\phi, \boldsymbol{a}) \quad = \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 & + & d_\rho \cos\phi_0 & + & \frac{\alpha}{\kappa}\left(\cos\phi_0 - \cos(\phi_0 + \phi)\right) \\ y_0 & + & d_\rho \sin\phi_0 & + & \frac{\alpha}{\kappa}\left(\sin\phi_0 - \sin(\phi_0 + \phi)\right) \\ z_0 & + & d_z & - & \frac{\alpha}{\kappa}\tan\lambda \cdot \phi \end{pmatrix}$$

```
TMatrixD CalcDxDphi(Double|_t phi) const = 0;
```

飛跡の接線ベクトルを計算し、返却する。

$$\frac{\partial \boldsymbol{x}(\phi, \boldsymbol{a})}{\partial \boldsymbol{\phi}} = \begin{pmatrix} \frac{\partial x}{\partial \phi} \\ \frac{\partial y}{\partial \phi} \\ \frac{\partial z}{\partial \phi} \end{pmatrix} = -\left(\frac{\alpha}{\kappa}\right) \cdot \begin{pmatrix} -\sin(\phi_0 + \phi) \\ \cos(\phi_0 + \phi) \\ \tan\lambda \end{pmatrix} \tag{2.68}$$

2.3.3 節を参照

```
TMatrixD CalcDxDa(Double_t phi) const;
```

トラックパラメータの phi による微分を計算し、返却する。

$$\frac{\partial \boldsymbol{x}(\phi, \boldsymbol{a})}{\partial \boldsymbol{a}} = \begin{pmatrix} \cos\phi_0 & \sin\phi_0 & 0 \\ -\left(d_\rho + \frac{\alpha}{\kappa}\right)\sin\phi_0 + \frac{\alpha}{\kappa}\sin(\phi_0 + \phi) & \left(d_\rho + \frac{\alpha}{\kappa}\right)\cos\phi_0 - \frac{\alpha}{\kappa}\cos(\phi_0 + \phi) & 0 \\ -\frac{\alpha}{\kappa^2}(\cos\phi_0 - \cos(\phi_0 + \phi)) & -\frac{\alpha}{\kappa^2}(\sin\phi_0 - \sin(\phi_0 + \phi)) & \frac{\alpha}{\kappa^2}\phi\tan\lambda \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{\alpha}{\kappa}\phi \end{pmatrix}^T$$

2.3.3 節を参照

```
void CalcDapDa(Double_t   fid,
               Double_t   dr,
               Double_t   drp,
               TMatrixD & F) const;
```

$\Delta\phi_0$ = fid と、dr と drp との差分 d_ρ を基にして基準点を移したときの、F = $\partial \boldsymbol{a}'/\partial \boldsymbol{a}$ を計算する。2.3.2 節を参照

```
void CalcStartHelix(const TVector3 & x1,
                    const TVector3 & x2,
                    const TVector3 & x3,
                          Bool_t     dir = kIterForward) const;
```

三点 x1、x2、x3 を通る helix を作成し、その結果得られるトラックパラメータを、粒子の飛ぶ向き dir を考慮して、自分自身にセットする。

TStraightTrack: TVTrack を継承; 直線トラック

TStraightTrack は THelicalTrack クラスの、$\alpha \to \infty$ と $\phi \to 0$ の極限として定義され、

$$\frac{\alpha}{\kappa}\phi \quad = \quad \text{constant.}$$

である。$\frac{\alpha}{\kappa}\phi$ を定義すると、

$$\lim_{\phi\to 0}\begin{pmatrix} \frac{\alpha}{\kappa}(\cos\phi_0 - \cos(\phi_0+\phi)) \\ \frac{\alpha}{\kappa}(\sin\phi_0 - \sin(\phi_0+\phi)) \\ -\frac{\alpha}{\kappa}\tan\lambda\phi \end{pmatrix} \quad = \quad -\lim_{\phi\to 0}\frac{\alpha}{\kappa}\phi\begin{pmatrix} \frac{\cos(\phi_0+\phi)-\cos\phi_0}{\phi} \\ \frac{\sin(\phi_0+\phi)-\sin\phi_0}{\phi} \\ \tan\lambda \end{pmatrix} \quad = \quad \psi\begin{pmatrix} -\sin\phi_0 \\ \cos\phi_0 \\ \tan\lambda \end{pmatrix}$$

が得られ、ϕ を ψ とすることで、下記の飛跡の方程式が導かれる。

$$\boldsymbol{x}\,(\phi,\boldsymbol{a}) \quad = \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 & + & d_\rho\cos\phi_0 & - & \phi\sin\phi_0 \\ y_0 & + & d_\rho\sin\phi_0 & + & \phi\cos\phi_0 \\ z_0 & + & d_z & + & \phi\tan\lambda \end{pmatrix} \tag{2.69}$$

また、ここで ϕ は基準点から測った x-y 平面に投影した路程である。

メンバ関数:

```
void MoveTo(const TVector3 & xv0,
                  Double_t & fid,
                  TMatrixD * F = 0,
                  TMatrixD * C = 0);
```

基準点を xv0 へ移動させ、新たな基準点における路程 fid と、F$\neq$ 0、C$\neq$ 0 であればさらに、*F= $\partial\boldsymbol{a}'/\partial\boldsymbol{a}$ と共変行列*C を計算する。

```
TVector3 CalcXAt(Double_t phi) const;
```

式 2.69 より投影した路程 phi における粒子の位置ベクトルを計算する。

```
TMatrixD CalcDxDphi(Double|_t phi) const = 0;
```

飛跡の接線ベクトルを計算し、返却する。

$$\frac{\partial \boldsymbol{x}(\phi, \boldsymbol{a})}{\partial \boldsymbol{\phi}} = \begin{pmatrix} -\sin\phi_0 \\ \cos\phi_0 \\ \tan\lambda \end{pmatrix} \tag{2.70}$$

```
TMatrixD CalcDxDa(Double_t phi) const;
```

飛跡のトラックパラメータによる微分を計算する[9]:

$$\frac{\partial \boldsymbol{x}(\phi, \boldsymbol{a})}{\partial \boldsymbol{a}} = \begin{pmatrix} \cos\phi_0 & \sin\phi_0 & 0 \\ -d_\rho \sin\phi_0 - \phi\cos\phi_0 & d_\rho\cos\phi_0 - \phi\sin\phi_0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & \phi \end{pmatrix}^T$$

```
void CalcDapDa(Double_t   fid,
               Double_t   dr,
               Double_t   drp,
               TMatrixD & F) const;
```

$\Delta\phi_0 =$ `fid` と、`dr` と `drp` との差分 d_ρ を基にして基準点を移したときの、`F` $= \partial \boldsymbol{a}'/\partial \boldsymbol{a}$ を計算する。

TCylinder: TVSurface を継承; 円筒

`TCylinder` クラスは円筒状の面を具体化するために、`TVSurface` を継承し、その純粋仮想関数を実装する。

データメンバ:

```
Double_t    fR;             // 半径
Double_t    fHalfLen;       // 全長の半分
TVector3    fXc;            // 中心点: x_c = (x_c, y_c, z_c)^T
```

メンバ関数:

```
Int_t CalcXingPointWith(const TVTrack  & hel,
                              TVector3 & xx,
                              Double_t & phi,
                              Int_t      mode,
                              Double_t   eps = 1.e-8) const;
```

[9] helix との連続的な扱いを容易にするため、helix と同じトラックパラメータ $\boldsymbol{a} = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$ を用いる。この場合、κ は飛跡に寄与しないことになる。

解析的に解くことが可能であるため、CPU 時間を節約するため、基底クラスのニュートン法をオーバーライドして、track `hel` との交点 `xx` と、それに対応する交差 `eps` における位置パラメータ `phi` を計算し、交点の数を返却する。交点の選び方として (前方、最近接、後方) を、`mode = (-1,0,+1)` として選ぶことが可能である。

```
Double|_t CalcS(const TVector3 & xx) const;
```

$\boldsymbol{x}$ =`xx` における表面の方程式 2.61 の左辺 $S_k(\boldsymbol{x})$ を計算する:

$$S_k(\boldsymbol{x}) = (x - x_c)^2 + (y - y_c)^2 - \texttt{fR}^2 \tag{2.71}$$

```
TMatrixD CalcDSDx(const TVector3 & xx) const;
```

$\boldsymbol{x}$ =`xx` における $S_k(\boldsymbol{x})$ の勾配 $\partial S_k/\partial \boldsymbol{x}$ を計算する:

$$\frac{\partial S_k}{\partial \boldsymbol{x}} = \left(\frac{\partial S_k}{\partial x} \quad \frac{\partial S_k}{\partial y} \quad \frac{\partial S_k}{\partial z} \right) = \left(2(x - x_c) \quad 2(y - y_c) \quad 0 \right) \tag{2.72}$$

```
Bool_t IsOnSurface(const TVector3 & xx) const;
```

`xx` が表面上にあるかどうかを確かめる。

```
Double_t GetSortingPolicy() const;
```

表面オブジェクトを内側から外側へソートするための指針として半径 `fR` を返却する。

THype: TVSurface を継承; 双極面

`THype` クラスは双極面を具体化するために、`TVSurface` を継承し、その純粋仮想関数を実装する。

データメンバ:

```
Double_t    fR0;            // z = 0 でのウエスト長
Double_t    fHalfLen;       // 全長の半分
TVector3    fXc;            // 中心点: x_c = (x_c, y_c, z_c)^T
Double_t    fTanA;          // tan A、A はワイヤのステレオ角
```

メンバ関数:

```
Double|_t CalcS(const TVector3 & xx) const;
```

$\boldsymbol{x}$ =xx における表面の方程式 2.61 の左辺 $S_k(\boldsymbol{x})$ を計算する:

$$S_k(\boldsymbol{x}) = (x - x_c)^2 + (y - y_c)^2 - \texttt{fR}^2 - (z - z_c)^2\,\texttt{fTanA}^2 \tag{2.73}$$

```
TMatrixD CalcDSDx(const TVector3 & xx) const;
```

$\boldsymbol{x}$ =xx における $S_k(\boldsymbol{x})$ の勾配 $\partial S_k/\partial \boldsymbol{x}$ を計算する:

$$\frac{\partial S_k}{\partial \boldsymbol{x}} = \left(\frac{\partial S_k}{\partial x} \quad \frac{\partial S_k}{\partial y} \quad \frac{\partial S_k}{\partial z} \right) = \left(2(x - x_c) \quad 2(y - y_c) \quad -2(z - z_c)\,\texttt{fTanA}^2 \right) \tag{2.74}$$

```
Bool_t IsOnSurface(const TVector3 & xx) const;
```

xx が表面上にあるかどうかを確かめる。

```
Double_t GetSortingPolicy() const;
```

表面オブジェクトを内側から外側へソートするための指針として半径 fR0 を返却する。

TPlane: TVSurface を継承; 平面

TPlane クラスは平面を具体化するために、TVSurface を継承し、その純粋仮想関数を実装する。

データメンバ:

```
TVector3    fXc;            // 平面状の参照点: x_c = (x_c, y_c, z_c)^T.
TVector3    fNormal;        // 規格化された垂線: n = (n_x, n_y, n_z)^T.
```

メンバ関数:

```
Double|_t CalcS(const TVector3 & xx) const;
```

$\boldsymbol{x}$ =xx における表面の方程式 2.61 の左辺 $S_k(\boldsymbol{x})$ を計算する:

$$S_k(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_c) \cdot \boldsymbol{n} = (x - x_c)\,n_x + (y - y_c)\,n_y + (z - z_c)\,n_z \tag{2.75}$$

```
TMatrixD CalcDSDx(const TVector3 & xx) const;
```

$\boldsymbol{x}$ =xx における $S_k(\boldsymbol{x})$ の勾配 $\partial S_k/\partial \boldsymbol{x}$ を計算する:

$$\frac{\partial S_k}{\partial \boldsymbol{x}} = \boldsymbol{n} = \left(n_x \quad n_y \quad n_z \right) \tag{2.76}$$

```
Bool_t IsOnSurface(const TVector3 & xx) const;
```

xx が表面上にあるかどうかを確かめる。

```
Double_t GetSortingPolicy() const;
```

表面オブジェクトを内側から外側へソートするための指針として原点からの距離を返却する。

2.5 hybrid パッケージの実装

この節では、前節で述べた KalLib、KalTrackLib、GeomLib を用いて、リニアコライダー実験のための飛跡検出器を具体的に実装することを考える。このリニアコライダー用に実装されたクラスの集合を hybrid パッケージと呼ぶことにする。前節にも述べたが、リニアコライダー実験で用いられる飛跡検出器の種類としては、TPC、(barrel/Forward)IT、VTX がある。TPC、barrel IT[10]、VTX は、測定面がそれぞれ、KalTrackLib の `TVMeasLayer` と GeomLib の `TCylinder` の多重継承で表現できるような円筒形で[11]、原点を取り囲むように配置される。一方 Forward IT は、KalTrackLib の `TVMeasLayer` と GeomLib の `TPlane` の多重継承で表現できるような円盤状をしており、円筒の側面に相当する向きに配置される。以下に、実装に必要な、coordinate projector、reverse coordinate projector、projector matrix について、TPC、IT、VTX(円筒) と、Forward IT(平面) それぞれの場合を説明する。

2.5.1 TPC, IT, VTX (円筒)

Coordinate Projector

hit 点座標 $\boldsymbol{x}_k$ を座標軸へ投影する、coordinate projector は次のように表される。

$$\boldsymbol{m}_k(\boldsymbol{x}_k) \quad = \quad \begin{pmatrix} R_k \cdot \phi_k \\ z_k \end{pmatrix} = \begin{pmatrix} R_k \cdot \tan^{-1}\left(\frac{y_k}{x_k}\right) \\ z_k \end{pmatrix}$$

ここで R_k は k 番目の測定面における半径である。

Reverse Coordinate Projector

測定ベクトル $\boldsymbol{m}_k$ を、逆に hit 点座標 $\boldsymbol{x}_k$ へ投影する reverse coodinate projector は次のように表される。

$$\boldsymbol{x}_k \quad = \quad \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} = \begin{pmatrix} R_k \cdot \cos\phi_k \\ R_k \cdot \sin\phi_k \\ z_k \end{pmatrix}$$

[10] これ以降、簡略化のため、barrel 型の IT を単に IT、Forward 型の IT を Forward IT と呼ぶことにする。

[11] VTX、IT に関しては、実際の検出器では平面をつなぎ合わせて円筒状に配置する構造なのだが、ここでは単純に円筒と見なすことにする。

Projector Matrix

prjector matrix は次のように表される。

$$\boldsymbol{H}_k = \begin{pmatrix} \frac{\partial(R_k \cdot \phi_k)}{\partial \boldsymbol{a}} \\ \frac{\partial z_k}{\partial \boldsymbol{a}} \end{pmatrix} = \begin{pmatrix} R_k \left(\frac{\partial \phi_k}{\partial \boldsymbol{x}_k}\right)\left(\frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{a}}\right) \\ \frac{\partial z_k}{\partial \boldsymbol{a}} \end{pmatrix} = \begin{pmatrix} -\frac{y_k}{R_k}\left(\frac{\partial x_k}{\partial \boldsymbol{a}}\right) + \frac{x_k}{R_k}\left(\frac{\partial y_k}{\partial \boldsymbol{a}}\right) \\ \frac{\partial z_k}{\partial \boldsymbol{a}} \end{pmatrix}$$

ここで、

$$\frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{a}} = \frac{\partial \boldsymbol{x}\left(\phi\left(\boldsymbol{a}\right), \boldsymbol{a}\right)}{\partial \boldsymbol{a}} = \frac{\partial \boldsymbol{x}}{\partial \phi_k} \cdot \frac{\partial \phi_k}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}}$$

$$\frac{\partial \phi_k}{\partial \boldsymbol{a}} = -\frac{1}{\left(\frac{\partial S_k}{\partial \boldsymbol{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \phi_k}\right)} \frac{\partial S_k}{\partial \boldsymbol{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}}$$

2.5.2 Forward IT (平面)

Coordinate Projector

hit 点座標 $\boldsymbol{x}_k$ を座標軸へ投影する、coordinate projector は次のように表される。

$$\boldsymbol{m}_k(\boldsymbol{x}_k) = \begin{pmatrix} x_k \\ y_k \end{pmatrix}$$

Reverse Coordinate Projector

測定ベクトル $\boldsymbol{m}_k$ を、逆に hit 点座標 $\boldsymbol{x}_k$ へ投影する reverse coodinate projector は次のように表される。

$$\boldsymbol{x}_k = \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \\ Z_k \end{pmatrix}$$

ここで、Z_k は k 番目の測定面における z 座標である。

Projector Matrix

prjector matrix は次のように表される。

$$\boldsymbol{H}_k = \begin{pmatrix} \frac{\partial x_k}{\partial \boldsymbol{a}} \\ \frac{\partial y_k}{\partial \boldsymbol{a}} \end{pmatrix}$$

ここで、

$$\frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{a}} = \frac{\partial \boldsymbol{x}\left(\phi\left(\boldsymbol{a}\right), \boldsymbol{a}\right)}{\partial \boldsymbol{a}} = \frac{\partial \boldsymbol{x}}{\partial \phi_k} \cdot \frac{\partial \phi_k}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}}$$

$$\frac{\partial \phi_k}{\partial \boldsymbol{a}} = -\frac{1}{\left(\frac{\partial S_k}{\partial \boldsymbol{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \phi_k}\right)} \frac{\partial S_k}{\partial \boldsymbol{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{a}}$$

2.6 hybrid パッケージを用いた解析

この節では、前節で述べた hybrid パッケージを用いて行った運動量分解能と、衝突点分解能の解析について述べる。単独の μ 粒子を、検出器内に横運動量が一定となるように打ち込んで解析を行った。各検出器で設定した分解能はそれぞれ以下の表のとおりである。

検出器	分解能	測定点数
VTX	$\sigma_{r,\phi} = 4\mu m$	4
IT	$\sigma_{r,\phi} = 10\mu m$	4
TPC	$\sigma_{r,\phi} = \sqrt{\sigma_0^2 + \frac{C_d^2}{\alpha N_e} \times z}$ $(\sigma_0 = 55\mu m,\ C_d = 55.33\mu m/\sqrt{cm},\ N_e = 63)$ $\sigma_z = 600\mu m$	200

Table 2.1: 各検出器の性能

2.6.1 エネルギー損失補正の効果

まず、Kalman filter のエネルギー損失補正の効果を見るために、入射粒子の横運動量が 0.5 GeV のときの κ 分布を調べた。エネルギー損失の補正を行っていないものの結果を図 2.7、補正を行った結果を図 2.8 に示す。結果を見ると、エネルギー損失の補正を行ったほうが、若干 κ の分布の幅が狭くなっており、分解能が向上してることが分かる。また、補正を行ったほうが分布の中心値が $2GeV^{-1}(\kappa = 1/P_T = 1/0.5GeV)$ に近づいており、エネルギー損失の補正が正しく行われていることが分かる。

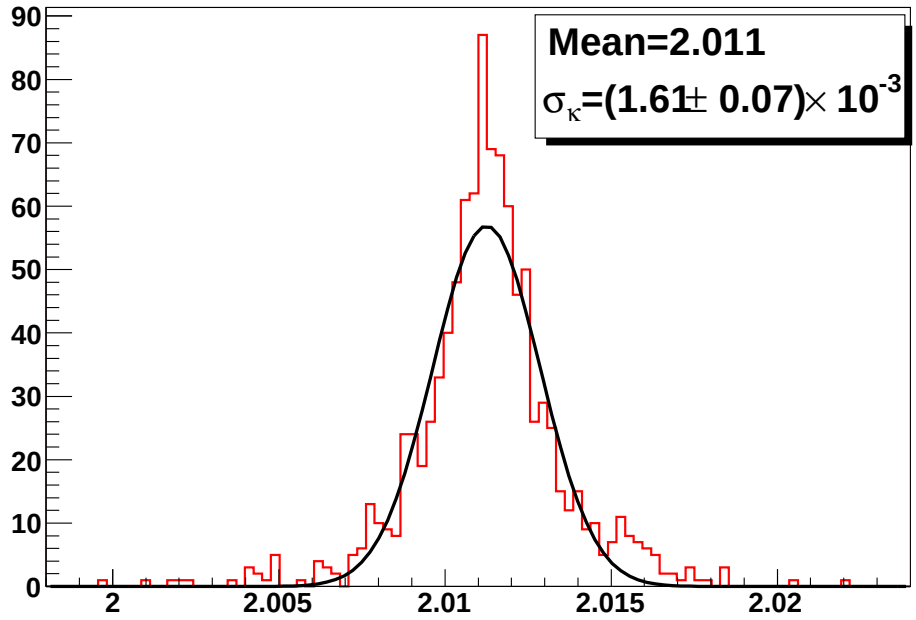

Figure 2.7: κ 分布 (エネルギー損失の補正なし)

2.6.2 運動量分解能

次に、各飛跡検出器の運動量分解能について調べた。

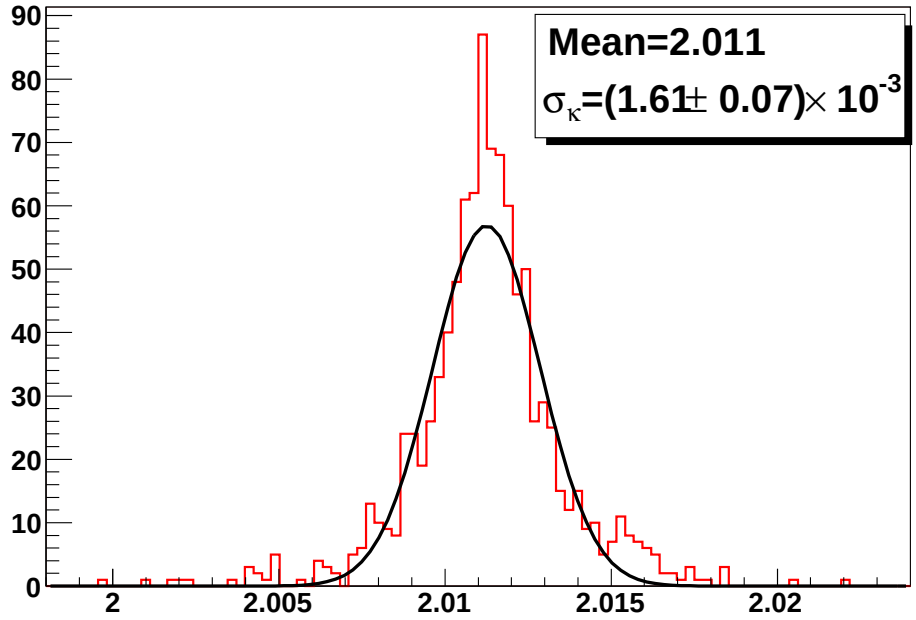

Figure 2.8: κ 分布 (エネルギー損失の補正有り)

コサイン依存性

始めに運動量分解能の $\cos\theta$[12] 依存性について述べる。入射粒子の横運動量が 1 GeV、5 GeV、100 GeV のときの結果を、それぞれ図 2.9、図 2.10、図 2.11 に示す。

結果を見ると、$\cos\theta > 0.8$ の領域で急激に分解能が悪化してることが分かる。これは、$\cos\theta$ が 0.8 よりも大きくなると track が TPC の端板に当たるようになるので、$\cos\theta$ が大きくなればなるほど測定点の数が減少していくためである。

100 GeV の結果では、$\cos\theta < 0.8$ の範囲では、$\cos\theta$ が大きくなるにつれて分解能が向上している。これは TPC の位置分解能がドリフト距離 (hit 点から TPC の端板までの距離) に依存しており 2.1、$\cos\theta$ が大きくなるほどドリフト距離が短くなり、位置分解能が上がるためであると考えられる。

また 1 GeV の結果を見ると、$\cos\theta$ が大きくなるにつれて IT を加えたもの、さらに VTX を加えた track の分解能が向上していることが分かる。低運動領域では r-z 平面における track のサインカーブが見えるようになり、z 方向の座標が分解能へ寄与するようになる。そして $\cos\theta$ が大きくなるにつれてこの寄与も大きくなるために、このような結果になると考えられる。一般に、IT や VTX を加えたときの分解能の向上は高運動量領域でしか見込めないと考えらており、この結果は非常に興味深い。

次に $\cos\theta > 0.9$ の範囲では、track は IT や VTX ではなく Forward IT と交差するようになるのだが、他の範囲と同様に TPC 単体より Forward IT を加えた分解能が TPC 単体より向上していることので、超前方の track において Forward IT が有効に機能するといえる。

[12] $\cos\theta$ は r-z 平面に対する track の角度で $\cos\theta = \tan\lambda/\sqrt{1+\tan\lambda^2}$ である。

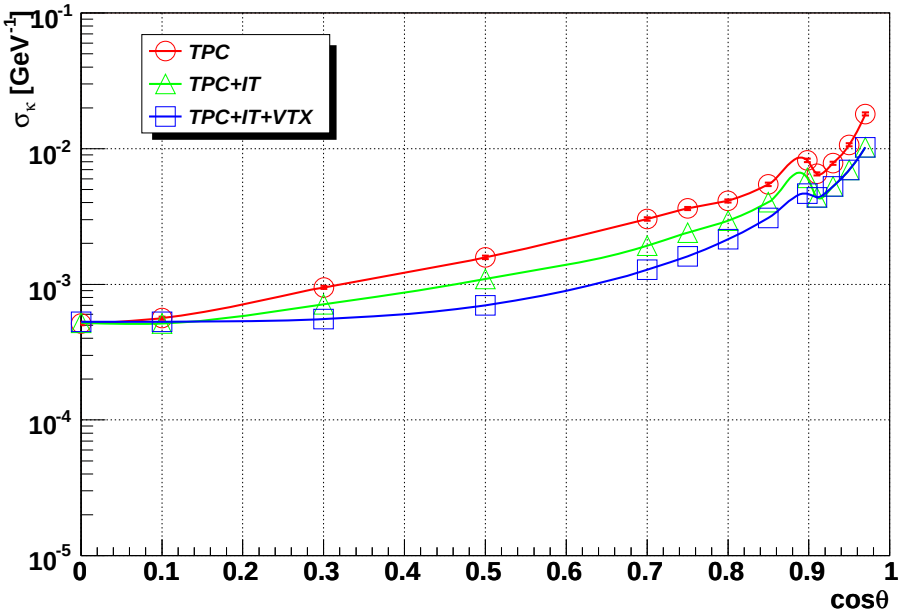

Figure 2.9: Momentum resolution vs. $\cos\theta$ at 1 GeV.

横運動量依存性

次に、入射粒子の横運動量に対する運動量分解能の依存性について調べた。ここでは、track が TPC の端板に当たらないようにするため、$\cos\theta < |0.7|$ とした。図 2.12 に結果を示す。

結果を見ると、高運動領域で TPC、TPC+IT、TPC+IT+VTX の分解能がそれぞれ約 $1.5\times10^{-4}[GeV^{-1}]$、$5\times10^{-5}[GeV^{-1}]$、$4\times10^{-5}[GeV^{-1}]$ であり、リニアコライダー実験に求められる運動量分解能に達していることが分かる。低運動領域で TPC+IT、TPC+IT+VTX の分解能が TPC 単体に比べて向上しているのは前述の z 方向の寄与のためと考えられる。

また、track に衝突点拘束を課したときの運動量分解能についても調べた。衝突点拘束とは、track が途中で kink したものでなく原点から出発していることが判明してる場合に、原点を track の hit 点として加えることである。これによって track に精度の良い hit 点を加えられ、さらに track の lever arm が大きくなるので、分解能を向上させることができる。図 2.13 に結果を示す。

結果を見ると、まず TPC 単体の運動量分解能が高運動領域で大きく向上して、$5\times10^{-5}[GeV^{-1}]$ に到達していることが分かる。これは track の lever arm が、TPC の内筒の分だけ大きくなるためであると考えられ、前述の衝突点拘束をかけていない TPC+IT+VTX の分解能とほぼ等しいこととも矛盾していない。また TPC+IT、TPC+IT+VTX の結果に関しても、高運動量領域で若干分解能が改善して、それぞれ $4\times10^{-5}[GeV^{-1}]$、$3\times10^{-5}[GeV^{-1}]$ となることが分かった。

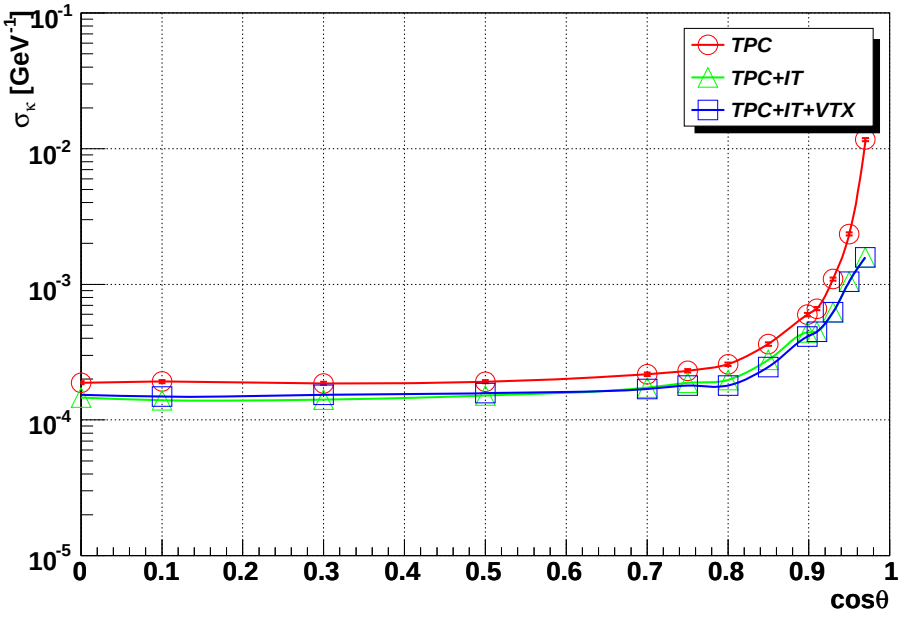

Figure 2.10: Momentum resolution vs. $\cos\theta$ at 5 GeV.

2.6.3 衝突点分解能

最後に衝突点分解能の、入射粒子の横運動量に対する依存性を調べた結果を図 2.14 に示す。衝突点分解能とは track の衝突点の位置分解能であり、track がビームの衝突から生成されたものなのか、途中の kink や他の物質との反応によって生成されたものかどうかを見分けるために重要となる。

結果を見ると、高運動量の極限において、TPC+IT+VTX の衝突点分解能が 2μ に到達しており、リニアコライダー実験に要求される分解能を満たすことができることが分かった。

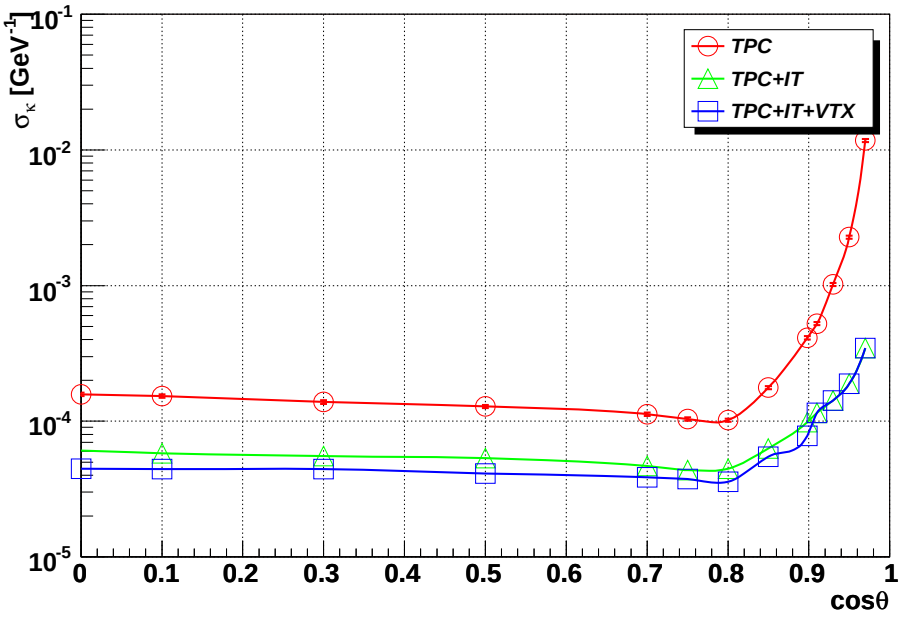

Figure 2.11: Momentum resolution vs. $\cos\theta$ at 100 GeV.

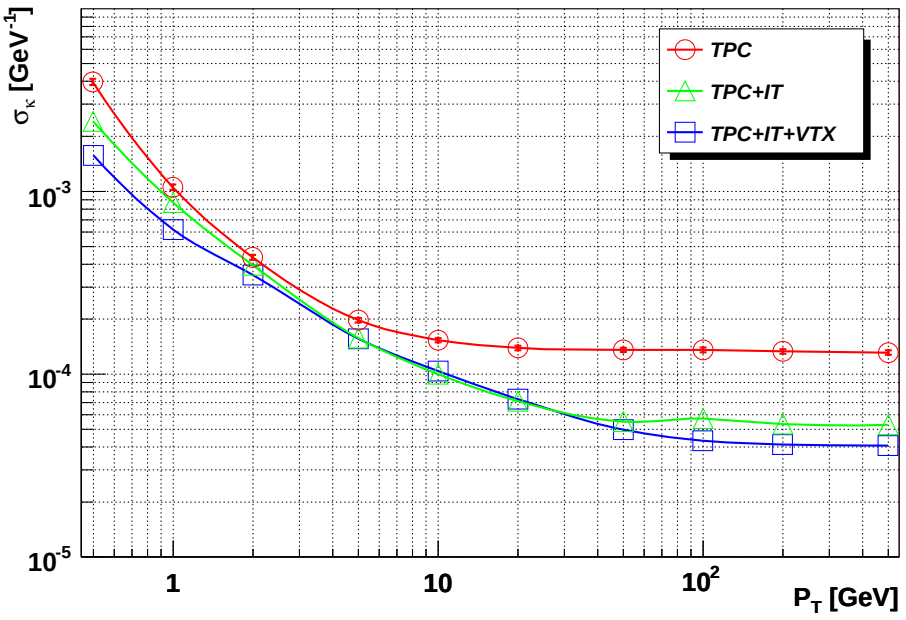

Figure 2.12: Momentum resolution vs. P_T.

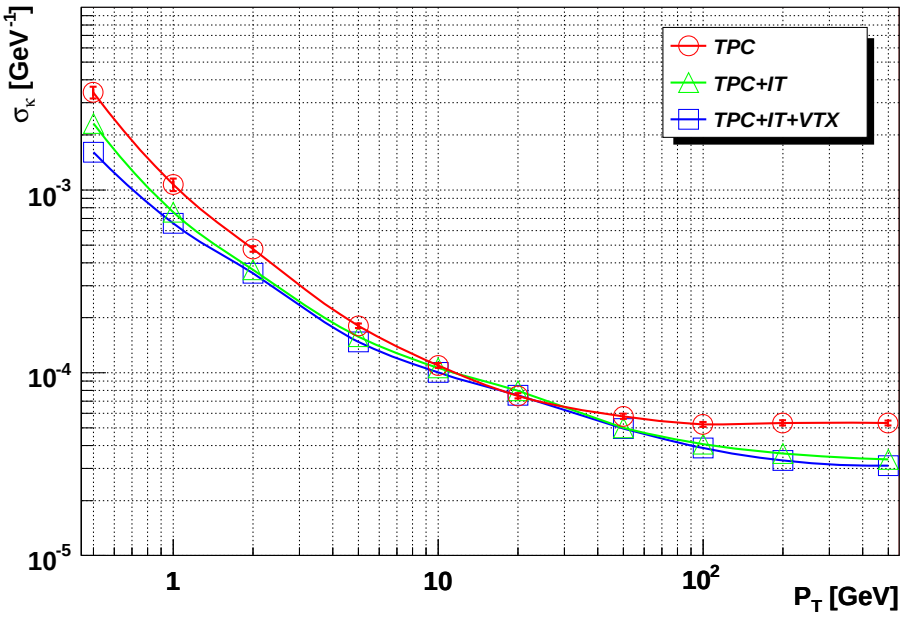

Figure 2.13: Momentum resolution vs. P_T with IP constraint.

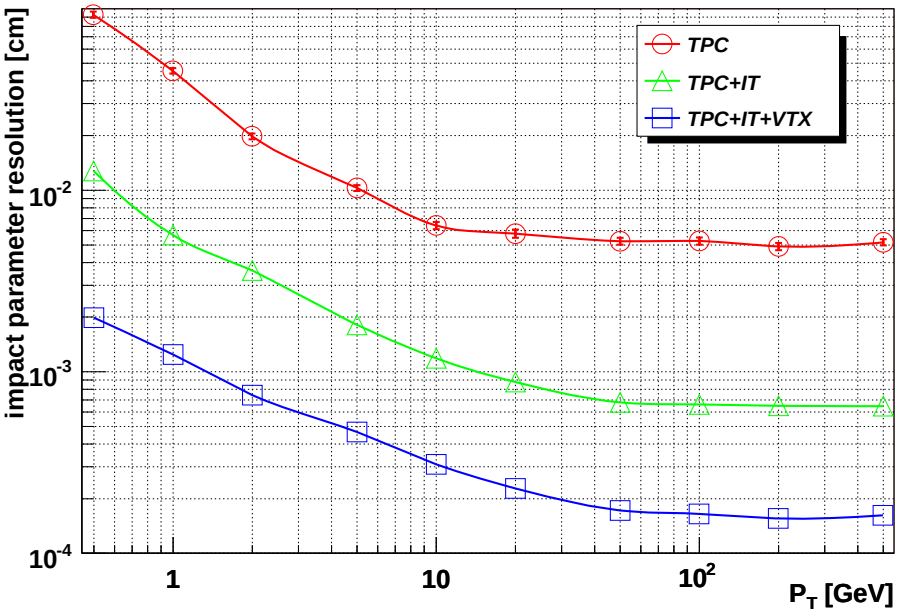

Figure 2.14: Impact paramter resolution vs. P_T.

Chapter 3

JUPITERとSatellitesの枠組み

3.1 シミュレータの構造と役割分担

このシミュレータはまず以下を目標として作られた。リニアコライダー用の飛跡検出器のフルシミュレーションを行うと同時に、他の検出器グループが、自由にそれぞれの検出器部分をインストールできる仕様にする。また、オブジェクト指向言語の特徴を生かし、部品を入れ替えたり、検出器のデザインパラメータを変更したりといった、検出器のデザインの変更が簡単に行えるようにする。(デザインパラメータチューニングのため)。更に、イベント再構成段階で混入する誤差を正確に見積もるため、最終的にはほぼ実験データ解析用のプログラムと同じ行程をシミュレートできる仕様にする。また、ビームラインからのバックグラウンドが検出器に与える影響を見積もるため、加速器の衝突点周辺の構造をもシミュレータに組み込み、バックグラウンドスタディを加速器と統合して行えるようにする。

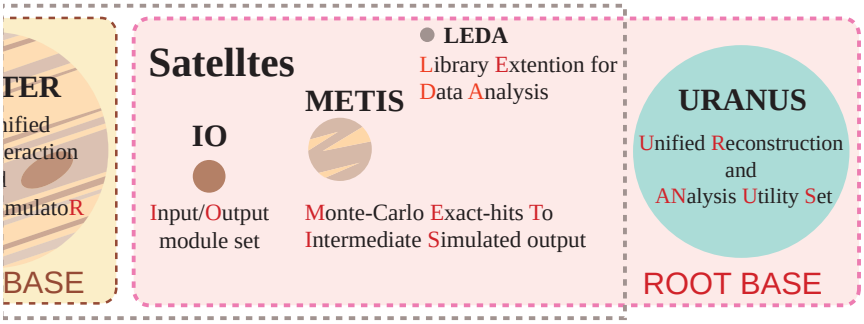

Figure 3.1: JUPITER、Satellites、URANUS の機能

以上を踏まえて設計したシミュレータのおおまかな構造と役割分担を図 3.1 に示す。主に検出器物質と粒子の反応部分のプログラムを軽くするため、Monte-Carlo Exact Hit (より一般的には Monte-Carlo Truth と呼ぶ) を出力するところで一度区切る仕様になっている。Monte-Carlo Truth を生成する部分 (JUPITER) は Geant4 をベースに開発されている。

この章の次節以降では、まずオブジェクト指向の概念について簡単に説明した後、フレームワーク「JUPITER」「Satellites」「URANUS」について述べる。

3.2 Monte-Carlo Truth 生成プログラム JUPITER

3.2.1 特長

JUPITER とは、**J**LC **U**nified **P**article **I**nteraction and **T**racking **E**mulato**R** の略である。前述のとおり、物質と粒子の反応部分のシミュレーションを Geant4 に依っているが、Geant4 は近年宇宙線分野、医学から地雷発見などの平和利用まで広く使用されるプログラムに発展しており、Geant3 に比べより高エネルギー実験色の薄い汎用プログラムになった。従って、個々の事例に応用する際、特に JLC のような大型プロジェクトで使用する際には、よりユーザーが使用しやすいように、アプリケーションを開発する必要がある。

JUPITER では、JLC の測定器パラメータがまだ完全に決定していない点、今後のバックグラウンドの研究によっては、大幅なデザイン変更があり得ることなどから、次の 3 点の特性を要求された。

1. 複数の研究グループが同時に開発可能であること。
2. 可能な限り、実器のデザインに近付けるため、検出器のコーディングそのものが簡単であること。
3. 検出器デザインの変更が簡単であること。特定の測定器や、測定器の一部のインストール／アンインストール、置き換えを含む。

これらの要求を満たすために JUPITER は以下のようになっている。

複数のサブグループによる同時開発

JLC には、バーテックス、Intermediate Tracker、TPC、カロリーメータ、Muon chamber の測定器開発グループがあり、さらにビーム衝突点グループ、Solenoid Magnet などの測定器以外の研究グループがある。これらのサブグループが同時にシミュレータの開発を行えるようにするためには、これらのグループが担当する部分を完全に独立になるように分けてしまうのが良い (各検出器のモジュール化)。

図 3.2 は、これをもっとも単純明快な形で実現したものである。各サブグループには、それぞれのディレクトリ (ワークスペース) を割り当て、自分のディレクトリの中のファイル以外の変更は、原則として行えないようにした。これにより、各サブグループは、いつでも好きなときに自分の担当する検出器のアップグレードを行うことが出来る。アップグレードは、ただディレクトリごと新しいものに置き換えるだけでよい。

シミュレータの管理者は、main ディレクトリと kern(kernel の略。意味するところは木星の核である) ディレクトリの管理を行い、各サブグループに物質を置いてよいスペースを配分する。kern ディレクトリには、Geant4 を走らせる為のクラス (Event ,Run, Physics に関係するクラス)、視覚化のためのクラスなどが置かれており、この kern クラスだけで、実験室がひとつ置かれているだけのミニマムセットを作成し、イベントを走らせることができる。

3.2.2 ベースクラス構造

JUPITER のベースクラスは、大きく分けて、次の 3 つの区分に分けられる。

1. 検出器のジオメトリカルな部品 (Detector Component) が共通して持つべき特性を集約したベースクラス (J4VComponent、J4VXXXDetectorComponent)

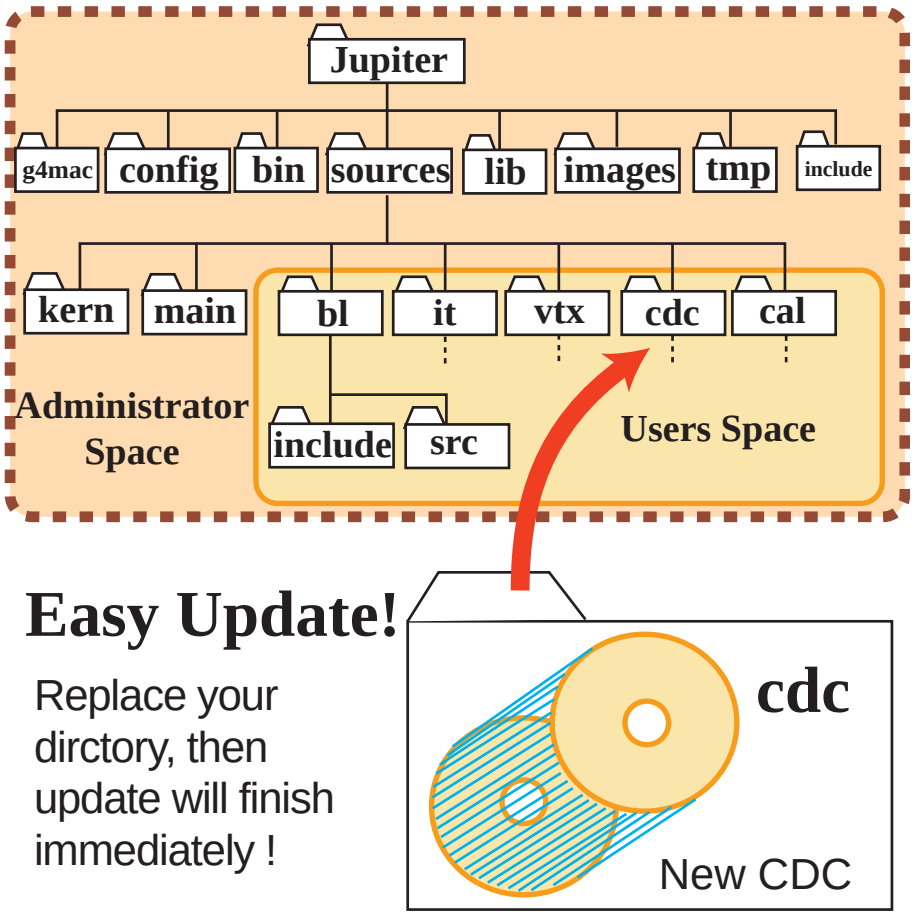

Figure 3.2: JUPITER のディレクトリ構造

2. 検出器の反応部分 (Sensitive Detector) に関する共通特性を集めたクラスと、Monte-Carlo Exact Hit を扱うベースクラス (J4VSensitiveDetector、J4VSD、J4VHit)

3. 物質の定義と管理を行うクラス (J4VMaterialStore、J4XXXMaterialStore)

これらのクラス同士の UML 図を、図 3.3 に示す。なお、頭に J4 がついているのが JUPITER のクラス、G4 がついているものは Geant4 のデフォルトのクラスである。

J4VComponent クラス

J4VComponent は、全ての検出器部品 (広くは加速器部分の部品も含まれる) の親となるベースクラスである。J4VComponent クラスは、検出器を構成するそれぞれの部品 (コンポーネント) のひな型の原形として働くが、そのひな型は固定された形、材質のコンポーネントを量産できるだけではなく、少しずつ大きさの違うコンポーネントを生成したり、材質の違うコンポーネントを生成したりすることもできる。

J4VComponent クラスは、Geant4 の用語でいう PhysicalVolume 1 つに対し、1 つのオブジェクト[1]を作るよう設計されている。これは、Geant4 で測定器のジオメトリを定義する際に必ず必要な 3 つのオブジェ

[1]オブジェクト指向のオブジェクトだが、ここでは測定器の個々の部品を表すコンポーネントだと思ってほぼさし支えない。

クト (Solid、LogicalVolume、PhysicalVolume) のうち、もっとも実際の測定器部品に近い特徴を備えているからである[2]。以下に、J4VComponent クラスのデータメンバと、代表的な関数 (機能) について述べる。

データメンバと自動ネーミング機能

表**??**は、検出器の部品がもっているべき属性と、J4VComponent クラス中での表現の対応を示す。なお、* がついているものは対応するコンポーネント (オブジェクト) へのポインタを所持していることを示す。このうち、名前は、個々のコンポーネントを特定するために大変重要であるが、それゆえに重複した名前をつけると大変なことになる。したがって、JUPITER では全てのコンポーネントの名前は J4VComponent クラスが重複しないようにつける仕様になっている。

Assemble 関数

Assemble 関数は、名前の通り、検出器コンポーネントの組み立てを行う関数である。Geant4 の用語でいえば、LogicalVolume をセットするところまでを行う。検出器コンポーネントは、その内部にまだ内部構造があれば、その内部構造を作り込んだところで形が完成するのであるから、内部にインストールされるコンポーネントを作成[3]するのは、基本的にはこの Assemble 関数の中である。

検出器コンポーネントの組立は、実験ではインストールよりも随分前に行うことが多いが、かならずしもこの点においてまで実験を模倣する必要はなく、インストールの直前で組み立てが仕上がっていればよい。したがって、基本的に Assemble 関数は後に述べる InstallIn 関数の中で呼ばれるべきものであり、外部に公開する必要のないものである。そこで、Assemble 関数はプライベート関数に設定した。これにより、検出器コンポーネントのジオメトリ・材質等についての完全な情報隠蔽が行われ、あるコンポーネントの内部の構造変更がその他のプログラムに影響を及ぼさない仕様になっている。

InstallIn 関数

InstallIn 関数は、自分自身を親コンポーネントにインストールするための関数である。引数に親コンポーネントへのポインタをもらい、主に、その中にどのような形態でインストールされるのかを記述する[4]。引数に位置や回転をとることも出来るので、親コンポーネントの Assemble 関数の中で、位置を指定して娘コンポーネントの InstallIn 関数を読んでやれば、親コンポーネントの望みの位置にインストールすることが可能である。この関数は親コンポーネントから呼ばれるため、当然パブリック関数である。実際のコーディングの一例を、図**??**に示す。

表題の XXX には、サブグループ名が入る。ユーザーは、各サブグループに一つ、J4VComponent クラスを公開継承してサブグループ独自のベースクラスを作り、全ての検出器コンポーネントは、このサブグループ専用のベースクラスを公開継承しなくてはならない。

J4VXXXDetectorComponent クラスの役目は 3 つある。一つは、名前の中にサブグループ名を入れることである。これにより、もしコンポーネントに他の検出器グループと重複する名前 (例えば Layer など) を指定しても、全体では違った名前がつけられることになる。

[2]Solid は検出器部品の形のみを定義する。LogicalVolume は、Solid に材質、感応領域であるか否か、絵を描かせたときに表示するか否か、などの情報を合わせたものである。PhysicalVolume は、LogicalVolume にそれが置かれる位置、回転角度などの配置情報を合わせたものである。従って、ビームラインにインストールされた検出器は配置情報を持つので、PhysicalVolume が一番近い。

[3]プログラム用語で言えば、new 演算子で娘オブジェクトを生成すること。

[4]現実的には、PVPlacement を呼ぶか、Replica を呼ぶかの二者択一である。Geant4 にはもう一つ PVParametrized というオブジェクトの置き方があるが、これは内部構造をもつオブジェクトではうまく作動しない上、実行メモリを気にしなければ PVPlacement で代用出来るので、JUPITER ではサポートしていない。

あとの二つは、各サブグループに一つあればよいクラスを開くという役目である。クラスの中には、複数存在すると混乱を来すものがある。代表的なものが定義を行うクラスであり、検出器パラメータや検出器の材質を定義するクラスがこれに相当する。前者を ParameterList クラスとし、後者を別に MaterialStore クラスとする。これは、後者がコンポーネントの物質を合成するためのマテリアル工場を持っており、単なるパラメータリスト以上の仕事を行うためである。これらのクラスについては、後に言及する。

勿論、各サブグループ独自に付け加えたい機能があれば、このクラスの中で定義することによって、同じサブグループに属する全ての検出器コンポーネントでその機能が使えるようになる。

J4VSensitiveDetector、J4VSD、J4VHit

これらの 3 つのクラスは互いに連係して働くので、3 つで一つの機能を提供すると考えてさし支えない。

Geant4 では、測定器の感応部分 (Sensitive Region) に対し、G4VSensitiveDetector の指定を行う。SensitiveDetector は、LogicalVolume に対して定義され、LogicalVolume の範囲内に粒子が飛び込んで来たときに、何らかのヒット情報を生成し、この Hit 情報を G4VHit クラスのオブジェクトに詰めた上で、バッファに記録する役目を負う。

バッファに記録されたヒット情報は、イベントの終わりに出力される。このとき、書出し命令を行うのは G4EventAction クラスであり、このクラスはシミュレータ全体の動作に関わるクラスなので、当然サブグループユーザの触れない kern ディレクトリの中におさめられている。ここで、G4EventAction にユーザがアクセスしなくてもヒットの書出しを行えるようにしたのが、この 3 つのクラスからなる仕組みである。

まず、G4EventAction の中からは、J4VComponent クラスの OutputAll 関数が呼ばれる。この関数は、全ての検出器コンポーネントを再帰的にスキャンし、SensitiveDetector に指定されたコンポーネントがあれば、そのコンポーネントの SensitiveDetector の OutputAll 関数を呼び出す。SensitiveDetector の OutputAll 関数は、自分が持っているバッファからヒットオブジェクトを一つずつ取り出し、そのヒットオブジェクトの Output 関数を呼ぶ。こうして、ヒットオブジェクトに詰め込まれたヒット情報がアウトプットされる。

この仕組みがうまく働くためには、全てのサブグループの SensitiveDetector は J4VSD クラスを公開継承して作られねばならない。また、Hit クラスも同様に、J4VHit を公開継承して作られる必要がある。

J4VMaterialStore、J4XXXMaterialStore

これらは、検出器の物質を定義し、提供するためのクラスである。J4XXXMaterialStore の XXX の部分には、サブグループ名が入る。

Geant4 では、検出器の物質を自分で定義できる仕様になっている。ガスの混合比を変えたり、特殊合金を作ったりといったことも可能である。検出器のデザインを試行錯誤する段階で、物質を構成する材料の混合比を少しずつ変えてみたり、あるいは温度や圧力を変えてみたりすることは十分あり得る話であるが、その度に管理者に頼んで望みの物質を作ってもらうのは面倒である。

そこで、管理者は基本的な材料カタログを提供するのみとし、各サブグループではカタログにない材料を自分で合成出来る仕様にした。J4VMaterialStore クラスは、仮想関数で Order 関数と Create 関数を持つ。これを継承して J4XXXMaterialStore クラスを作り、Create 関数のみ物質を定義して実装すると、材料の Order を受けたときにまず J4VMaterialStore に行ってカタログの中を探し、そこになければ自分の Create 関数の中を見て物質を合成する、といった作業を行わせることができる。

このしくみは、全体で統一して使用されるべき物質は全体管理者の管理とし、サブグループに対し、他の検出器部分に影響を与えるような勝手な物質定義を許さない、という意味でも重要である。

3.2.3 TPC の実装

これらのベースクラスを用いて、実際に TPC の実装を行った。

CDC の検出器コンポーネントの作成、インストールは、全て J4VTPCDetectorComponent を継承し、Assemble、Install 関数を使って行った。コンポーネントの関係は以下のとおりである。

1. Pad→PadRow→PadPlane
2. Layer→DriftRegion
3. PadPlane, DriftRegion, Endcap, SupportTube→Half
4. TPCHalf→TPC

図 3.4 は $E_{CM} = 350GeV$、$e^+e^- \to ZH$ における TPC のイベント図である。[h]

3.3 イベント再構成プログラム URANUS とそのシミュレータ Satellites

イベント再構成部分のシミュレータの開発はイベント再構成プログラムの開発にほぼ等しい。JLC のサブグループはこれまでにも多くの R&D 実験を行っており、これらの実験の解析で培われたテクニックは、そのままイベント再構成シミュレータに用いることができる。

イベント再構成シミュレータ Satellites は、イベント再構成プログラム URANUS を継承して作成され、URANUS と自由に連係しながら動かせる形になっている。この節では、これらのプログラムの詳細と連係の方法について述べる。

3.3.1 TPC におけるイベント再構成と URANUS

URANUS は、**U**nited **R**econstruction and **AN**alysis **U**tility **S**et の略である。名前の示す通り、将来的には他のサブグループ検出器と統合して、イベント再構成と解析を受け持つ。JLC では、イベント再構成プログラムのためのフレームワークとして既に JSF が存在するので、JUPITER の場合のように改めて開発しなければならないサブグループ共通のベースクラスはほとんどない[5]。したがって、ここでは TPC 部分の実装に焦点を絞って述べることとする。

TPC における飛跡再構成は、図 3.5 の手順で行われる。

1. FADC データをヒットごとのクラスターに分ける (Clustering)。

[5]シミュレータ部分には、データの形式を統一するためのフレームワークを用意している。実際の実験データのためのフレームワークは、各サブグループのイベント解析プログラムが出揃った段階で整備される予定。現状では、イベント解析に JSF を用いているのは JLC-CDC グループのみである。

2. クラスターの立ち上がりの時間を読む。FADC の横軸 (時間軸) のチャンネル 0 が、トラックがセルを通過した時刻 t_0 に一致していれば、クラスターの立ち上がりの時刻とガスのドリフト速度を用いて、トラックがパッドからどれだけの距離の位置を通ったか (Hit point) を知ることができる。これを Hit Making という。

3. 得られたヒットを集めて、同じトラック起源と思われるヒットを選びだす。Clustering で間違って生成してしまった Cluster 起源のヒットは、ここでふるいおとされる。これを Track Finding という。

4. Track Finding で選ばれたヒットを Fitting する。

これらの作業を行うために、URANUS の TPC パートでは次のモジュールを備えている (そのモジュール内で解析された結果を保存しておくバッファを含む)。

1. TPCUnpacker
2. TPCClusterMaker
3. TPCHitMaker
4. TPCTrackFinder
5. TPCTrackFitter

ここで、TPCUnpacker は、実験によって得られたデータを FADC 1 チャンネル分ずつ切り出し、FADC datum オブジェクトに詰める役目を果たすものである。DAQ(オンラインプログラム) の出力を受け取るモジュールでもあるので、DAQ 側のプログラムの変更は全てここで吸収される。その他のモジュールは、それぞれ上で挙げた仕事を受け持っており、直前のモジュールのバッファのデータを解析して、自分のバッファにつめる作業を行っている。この形式の意味するところは、バッファにつまっているデータの形が同じであれば、モジュールを差し換えても問題なく動くということである。

次の節では、この特性を生かしたシミュレータの構造について述べる。

3.3.2 イベント再構成シミュレータ Satellites の全体構造

Satellites とは、木星の衛星の名前を冠したイベント再構成シミュレータのモジュールセットを指す。図 (3.6) は、このシミュレータモジュールセットと JUPITER、URANUS がどのような関係で結ばれているかを示す。

[h]

Satellites のうち METIS は、URANUS と同じ構造を持っていることが明らかである。J4TPCHitMaker の上に 2 か所点線で描かれた空欄は、将来その隣に位置する URANUS のモジュールに対応するシミュレータモジュールがおかれる可能性を示している。この METIS のモジュールに関しては、全てが対応する URANUS のモジュールを継承して作られており、したがって、バッファにつまっているデータの形も同じ顔つきをしている。それゆえ、ある部分までをシミュレーションプログラムで行い、途中から URANUS 解析プログラムに移行するといったシミュレーションレベルの変更が可能である。

このことの利点は、モジュール化されたイベント再構成のそれぞれの過程で、どのように誤差が混入するのかを、モジュールごとに調べられる点である。イベント再構成の過程で混入する誤差を可能な限り減

らし、その誤差の大きさと全体のイベント再構成に与える影響を正確に見積もるという点で、URANUS と METIS のセットは分かりやすくかつ使い易い関係にあるといえる。

3.3.3 IO(Input/Output module set)

木星のすぐ下には、ガリレオ衛星の中でももっとも有名な衛星の名前をもつ IO が配置されている。IO は、その名のとおり I/O を司る。第一の役目は、JUPITER のアウトプットをそれぞれの検出器に振り分け、Monte-Carlo Exact Hit を格納する J4VExactHit オブジェクトにつめることである。これにより、JUPITER 側のアウトプット形式の変化に柔軟に対応することができる。現在、IO の仕事はこの 1 番目の役目に終始している。

一方、将来的には、IO の仕事はもっとも大きく膨らむであろうと考えられる。まずは、シミュレーションの途中経過の書出しと読み込みをサポートすべきであるし、物理解析プログラムとの連結も考えなければならない。JUPITER 本体との接続も、もう少しスマートに行えるべきである[6]。更に言えば、JUPITER 本体の検出器パラメータを IO が管理する可能性もあり得る。この場合には、XML や CAD データとの連係も考えられる。

現在は、仕事量でやや METIS に押され気味であるが、近い将来、ガリレオ衛星の一の名に恥じない働きをする可能性が十分にあるモジュールである。

3.3.4 METIS(Monte-Carlo Exacthit To Intermediate Simulated output)

HitMaker

METIS の特徴は、第 3.3.2 節で触れた通り、シミュレーションの途中結果をアウトプットして (Intermediate Simulated output)、その先に通常の解析プログラムをつないで走らせることができる点である。個々のモジュールは、ほとんどが URANUS の対応するモジュールと同じ顔をしているが、シミュレータ独自のメソッドとして、Monte-Carlo Truth に誤差を加えてぼかすメソッドがついている。

HitMaker では、この作業を J4TPCHit クラスの CalculatePosition メソッドの中で行っている。この関数の中では、Monte-Carlo Exact Hit から得られたドリフト距離を、テストチェンバーのビームテスト実験で得られた位置分解能のデータに基づいて、ドリフト距離に依存する幅のガウス関数でぼかしている**??**。このぼかしたヒットの情報は、同時に Monte-Carlo Exact Hit の Smeared Hit ポインタにも登録され、Exact Hit が辿れる環境では常に Smeared Hit も参照できるようになっている。

TrackFinder

TrackFinder では、シミュレータ独自の方法として、Exact Hit がもっている TrackNumber の情報をみて、トラックごとにヒットを分けていくことができるようになっている (カンニング法)。これは、本当の TrackFinder を Uranus に実装したとき、その性能評価に使用できる。

[6]現在、JUPITER 側から直接 ROOT オブジェクトの形でデータをアウトプットする方針も考えられている。

TrackFitter

Track Fitting の部分では、Fitting アルゴリズムの違いだけが全体の Tracking の精度に関係するので、シミュレーション用のプログラムをわざわざ作成する必要はない。そこで、METIS に組み込む Fitting アルゴリズムとして、簡単な Helix パラメータによる Fitting と、Kalman Filter を応用した Fitting の両方を組み込んである。Kalman Filter を用いた Track Filter については次章で詳述する。

3.3.5 LEDA(Library Extension for Data Analysis)

LEDA はデータ解析を行うさいに使用する汎用ライブラリ群である。今回作成する Kalman Filter もこの LEDA 内に実装され、イベント再構成に使用される。

3.3.6 解析

上記で述べた JUPITER、URANUS/Satellites を用いて、前章 2 と同じく運動量分解能と、衝突点分解能を調べた。以下に hybrid パッケージの場合の図 2.9 から図 2.14 に対応して、図 3.7 から図 3.9 に結果を示す。

結果は、それぞれ前章の hybrid パッケージのものとほぼ一致しており、運動量分解能や衝突点分解能についてはフルシミュレータを用いる必要がないといえる。

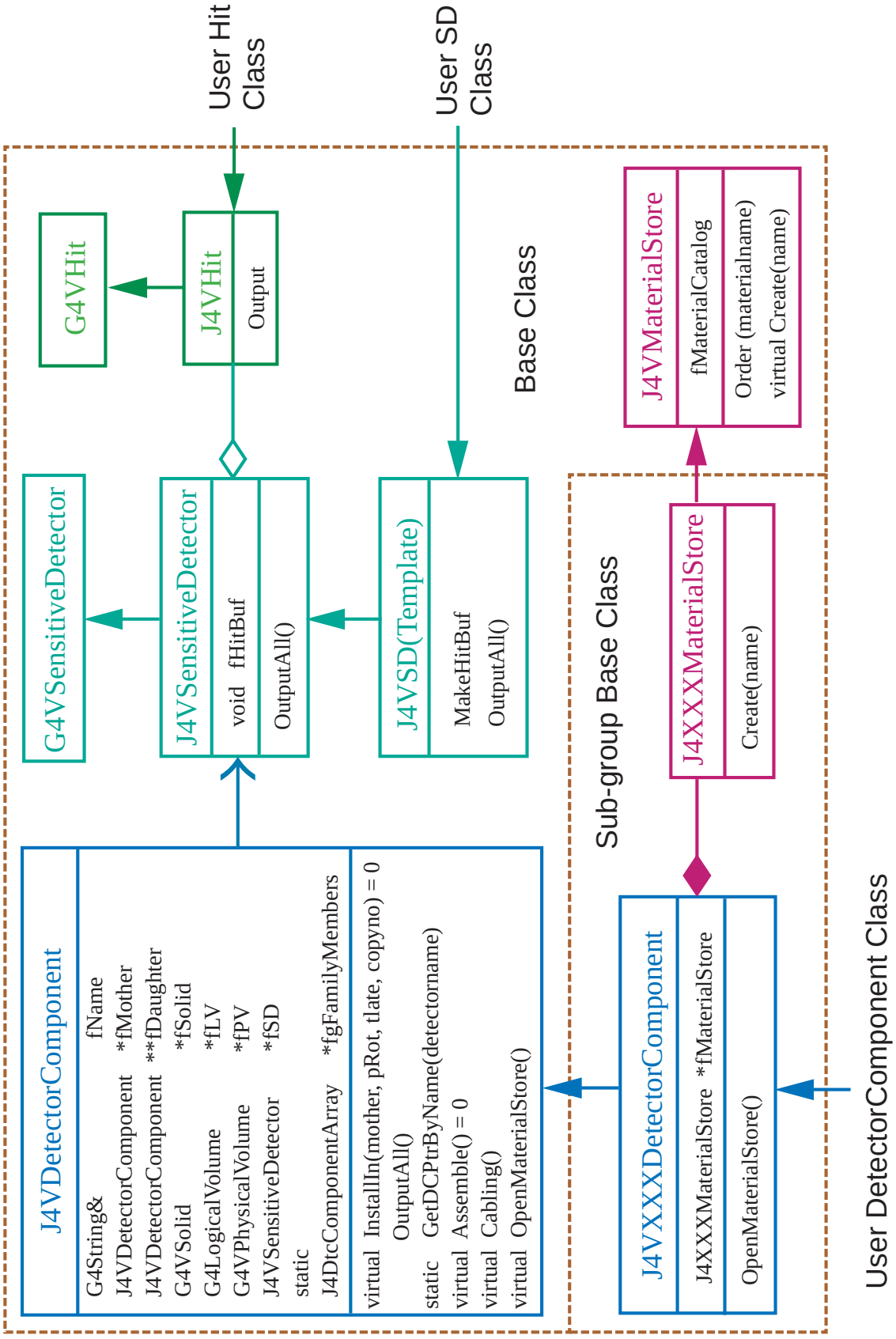

Figure 3.3: JUPITER の Base class

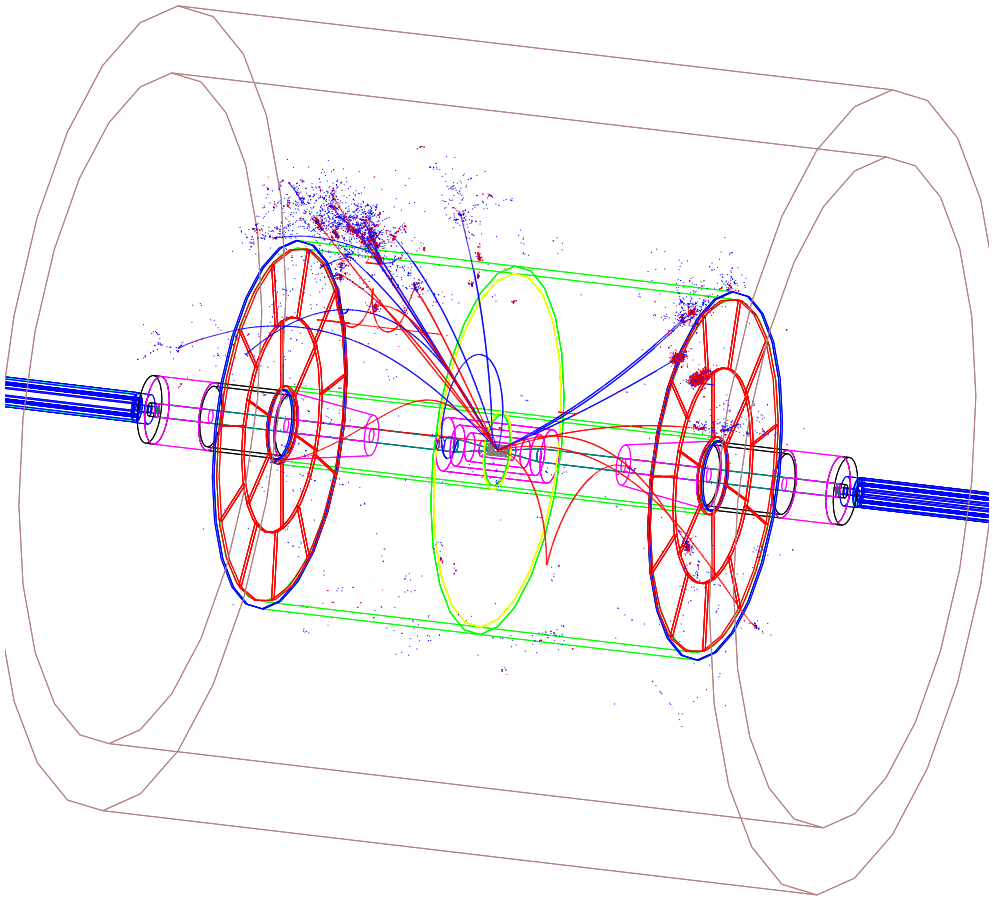

Figure 3.4: $E_{CM} = 350GeV$、$e^+e^- \to ZH$ におけるイベント図

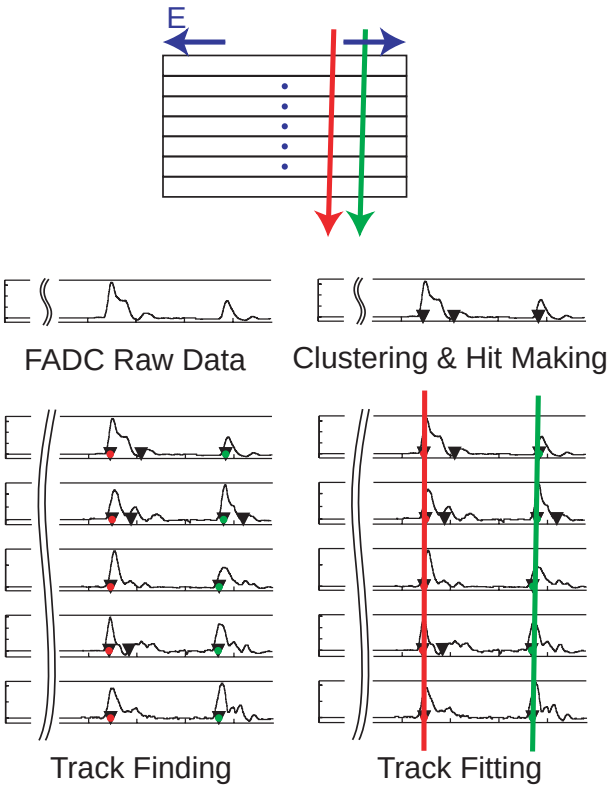

Figure 3.5: 飛跡再構成の手順

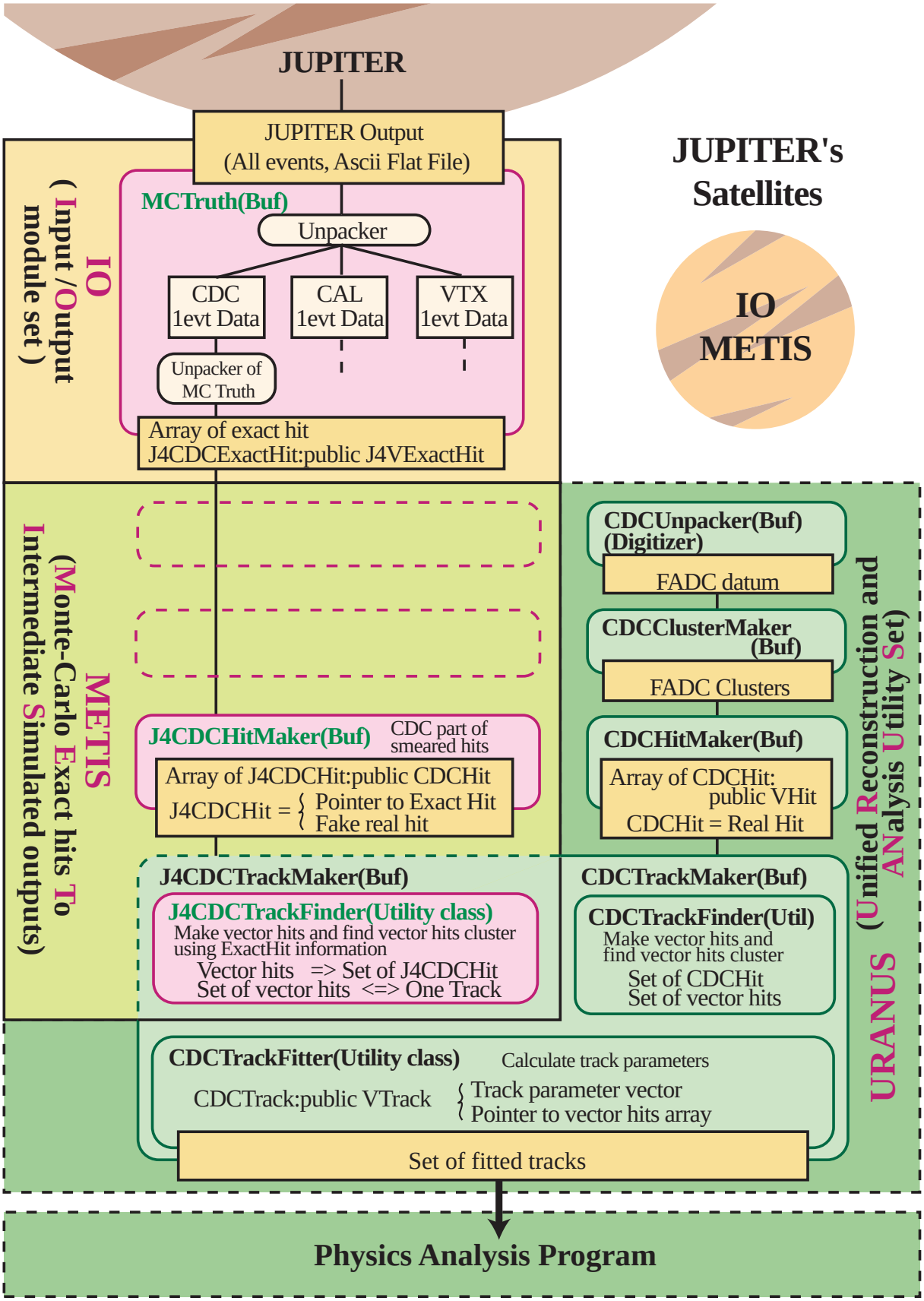

Figure 3.6: Satellites と JUPITER、URANUS との関係

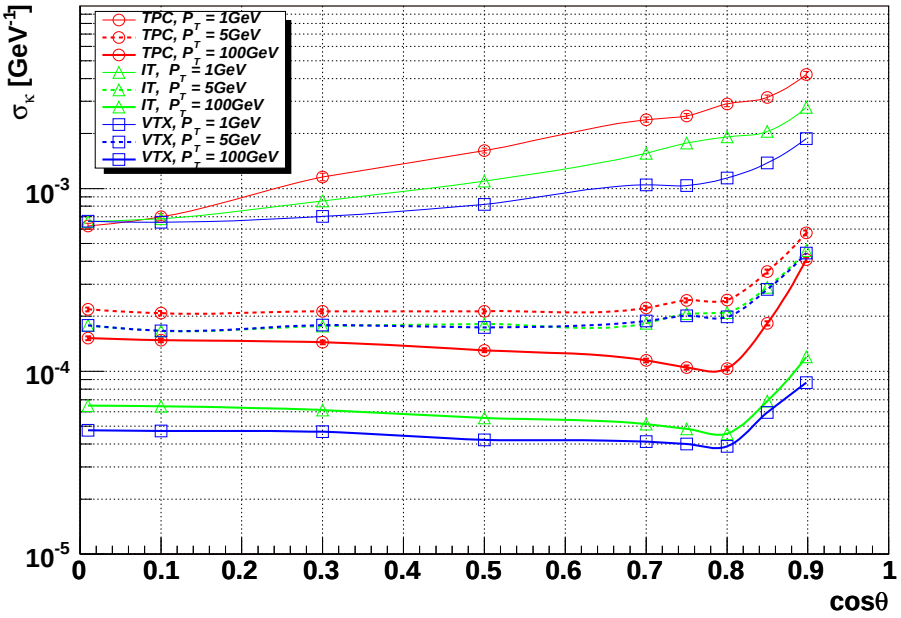

Figure 3.7: $\cos\theta$ に対する運動量分解能 (上から 1 GeV、5 GeV、100GeV)

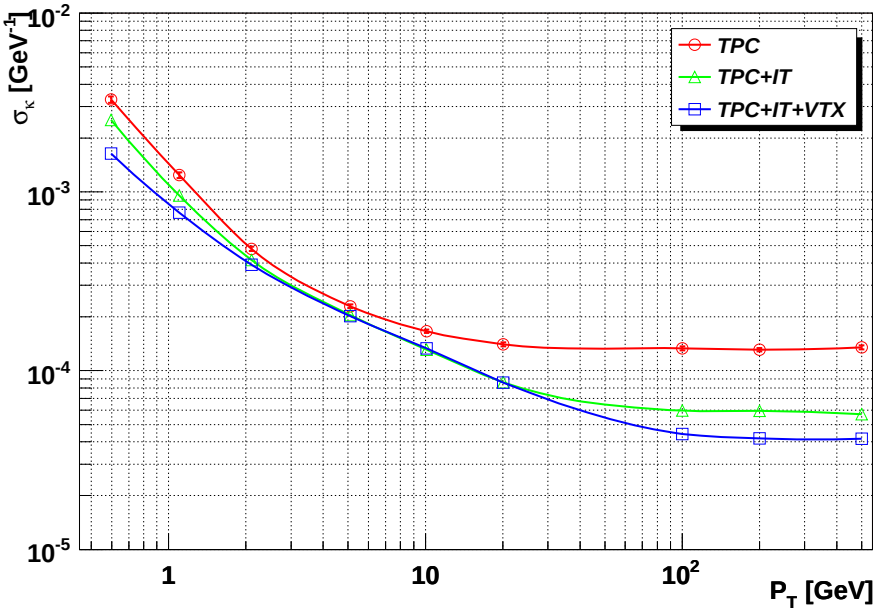

Figure 3.8: 横運動量に対する運動量分解能

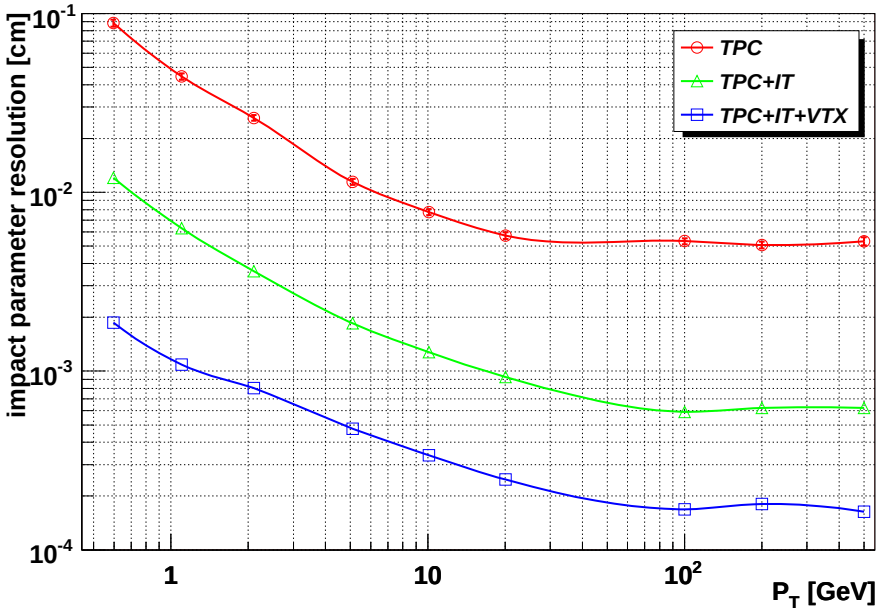

Figure 3.9: 横運動量に対する衝突点分解能

Chapter 4

結論

4.1 本研究の成果

LC-TPC(中央飛跡検出器) は、リニアコライダー計画で期待される物理のあらゆる場面で中心的な役割を担うことになる測定器であり、その基本設計は、物理からの性能要求を適切に反映したものである必要がある。本研究は、リニアコライダーの基本設計を目標として結成された、JLC-CDC グループの開発研究の一環としてスタートした。基本設計は、物理からくる性能要求を測定器パラメータに翻訳する作業と、その測定器パラメータが実現可能である事を実証する作業に大別できる。前者は、主としてシミュレーションを主要な手段とするが、後者においても、実機のプロトタイプ建設前の段階では要素技術のハードウェア開発が主体となるため、全体システムの性能評価は、要素技術のハードウェア開発の結果を踏まえたシミュレーションが本質的である。一方、要素技術のハードウェア開発は、その結果がフィードバックされる結果として、測定器パラメータに影響を与えるため、両者は車の両輪の関係にある。

JLC-CDC グループでは、システム全体としての性能評価をし、基本設計を完成するためフル・シミュレータ (JUPITER+Satellites) の開発を行っている。JUPITER は Geant4 ベース、Satellites は ROOT ベースで、いずれもオブジェクト指向技術を最大限に活用し、C++言語を用いて開発中である。本研究ではテストチェンバーデータ解析プログラムを元にして、URANUS 及び Satellites、TPC 解析プログラムの実装を行った。

続いてオブジェクト指向技術を活用し、Kalman Filter の基本アルゴリズムを提供する汎用的なトラックフィッティングクラスライブラリを開発した。これらのクラスライブラリは、異なる形状、異なる材質でできた測定器を統一的に扱えるように設計されている。必要に応じて検出器を薄い層に分割することで、連続散乱の効果を十分な精度で再現できるようにした事が本研究の大きな成果の一つである。これにより異なる物質の間の相互作用を考慮できるようになったために、様々な検出器を組み合わせて解析を行うことが可能になった。そこでこのトラックフィッティングクラスライブラリから、リニアコライダーの飛跡検出器の簡易シミュレータ (hybrid パッケージ) を構築し、TPC、IT、Forward IT、VTX に対応させる派生クラスを作成した。これを用いてミューオンシングルトラックイベントにおける運動量分解能と衝突点分解能を調べた。運動量分解能については、LC-TPC に要求される性能を満たしていた。また、低運動量域において TPC 単体よりも、TPC+IT、TPC+IT+VTX の分解能が向上するとが分かった。これは、これまでの

常識 (低運動量域での combined fit は分解能を改善しない) を覆す結果であり、低運動量域では z 座標が分解能に寄与するためと考えられる。また衝突点分解能についても、要求される性能を満たしていた。

続いて JUPITER、URANUS/Satellites を用いて同様の解析を行った。結果は hybrid パッケージの結果とほぼ一致するものが得られた。これより、運動量分解能や衝突点分解能等の解析は、バックグラウンドを考慮しないのであればフルシミュレータである JUPITER、URANUS/Satellites を用いなくとも、簡易シミュレータである hybrid パッケージで十分であると結論できる。

4.2 今後の方向

現在、track finding については Monte-Carlo 情報を用いて行っており、これを実際の実験で用いられる方法で実装する必要がある。そしてこの track-finding を用いてバックグラウンド中の運動量分解能や track の findng 効率、近接 hit 分離能等を解析する必要がある。また、Kalman filter ライブラリを用いて vertex-finding を実装することも興味のある課題といえる。

JUPITER および Satellites の枠組みは、すでに TPC、IT、VTX、IR、カロリメータ、ミューオン検出器とリニアコライダーの様々な検出器に実装されており、シミュレーションによる検出器 R&D の根幹をなす存在になっている。本研究は、飛跡検出器の解析フレームワークを通して、リニアコライダー測定器全体の完成に向けて大きな役割を果たすであろう。

謝辞

本研究は大学院修士課程の 2 年間、ILC のための中央飛跡検出器 (JLC-CDC) の開発研究の一環として行ったものです。これまでの間、JLC 物理グループでお世話になりました皆様に、心より感謝致します。

特に、筑波大学浅野研究室の浅野侑三教官、Samo Stanic 教官には、数々のご指導を頂きました。高エネルギー加速器研究機構素粒子原子核研究所 (KEK) においては、藤井恵介氏、松田武氏、小林誠氏、宮本彰也氏から、研究計画から論文の執筆に至るまで、研究生活全般にわたって様々な局面で貴重なご意見を賜りました。

そして、亀島敬氏をはじめとする浅野研究室の皆様や、JLC-CDC グループ、Acfa-sim グループの皆様には、研究生活、学生生活共に大変お世話になりました。皆様方の励ましとお力添えがなければ、この研究は完成を見ませんでした。

ここに、再度皆様に深くお礼申し上げます。

Bibliography

[1] R.L. Gluckstern, Nucl. Instr. and Meth. **A24** (1963) 381.
JLC physics group, http://www-jlc.kek.jp/subg/offl/lib/docs/helix_manip.ps.gz.

[2] S. Sudou *et al.*, Nucl. Instrum. Meth. A **383**(1996) 391 .

[3] ACFA Linear Collider Working Group, KEK Report 2001-11, August (2001), 374,
http://www-jlc.kek.jp/subg/offl/jim/index-e.html

[4] http://wwwinfo.cern.ch/asd/geant4/geant4.html

[5] ACFA Linear Collider Working Group, KEK Report 2001-11, August (2001), 360,
http://www-jlc.kek.jp/subg/offl/jsf/index.html

[6] http://root.cern.ch/

[7] http://www-jlc.kek.jp/subg/offl/www-jlcsim/index.html

[8] http://wwwinfo.cern.ch/asd/geant4/G4UsersDocuments/Overview/html/index.html

[9] http://www.thep.lu.se/ torbjorn/Pythia.html

[10] R.Kuboshima, JLC-CDC シミュレータへのチェンバー性能テスト結果の組込み

[11] http://www-jlc.kek.jp/subg/offl/lib/docs/helix_manip/main.html

[12] R. E. Kalman, J. Basic Eng. 82 (1961) 34.

[13] R. Frühwirth, Nucl. Instr. and Meth. **A262** (1987) 444.

[14] E. J. Wolin and L. L. Ho, Nucl. Instr. and Meth. **A219** (1993) 493.

[15] P. Astier *et al.*, Nucl. Instr. and Meth. **A450** (2000) 138.

[16] Helix Manipulation, the JLC Group, internal circulation (1998).