

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

**AKCELERACE ALGORITMU SLEDOVÁNÍ ČÁSTIC V EX-  
PERIMENTU CBM**

ACCELERATION OF PARTICLES TRACKING ON CBM EXPERIMENT

**BAKALÁŘSKÁ PRÁCE**

BACHELOR'S THESIS

**AUTOR PRÁCE**

AUTHOR

**MICHAEL ROTH**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Ing. PETR MUSIL, Ph.D.**

BRNO 2023

## Zadání bakalářské práce



150646

Ústav: Ústav počítačové grafiky a multimédií (UPGM)  
Student: **Roth Michael**  
Program: Informační technologie  
Specializace: Informační technologie  
Název: **Akcelerace algoritmu sledování částic v experimentu CBM**  
Kategorie: Zpracování signálů  
Akademický rok: 2022/23

### Zadání:

1. Prostudujte zařízení a algoritmy využívané pro detekci a rekonstrukci trajektorie částic v experimentu CBM.
2. Prostudujte chování a fyzikální modely pohybu částic, zaměřte se na možnosti simulace generování trajektorií částic.
3. Navrhněte a naimplementujte vybrané metody generování trajektorií, které budou sloužit pro ověření vlastností algoritmů pro sledování částic.
4. Vyberte algoritmus pro sledování částic, prozkoumejte možnosti a vlastnosti jeho implementace a zhodnoťte možnosti jeho akcelerace.
5. Implementujte vybraný postup, ověřte jeho vlastnosti na simulovaných nebo reálných datech.
6. Diskutujte dosažené výsledky a možnosti dalšího pokračování práce.

### Literatura:

- Valentina Akishina, 4D Event Reconstruction in the CBM Experiment, Doctoral Thesis, Johann Wolfgang Goethe-Universität, 2017 Frankfurt am Main

Při obhajobě semestrální části projektu je požadováno:  
1 a 2

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Musil Petr, Ing., Ph.D.**  
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.  
Datum zadání: 1.11.2022  
Termín pro odevzdání: 10.5.2023  
Datum schválení: 11.1.2023

## Abstrakt

Cílem této práce je prostudovat metody detekce a rekonstrukce trajektorií částic v experimentu CBM a problematiku akcelerace těchto metod na hardwarových platformách. V práci byly srovnány výhody a nevýhody rozšířených metod a pro další studium byla vybrána metoda rekonstrukce na bázi celulárních automatů a Kalmanových filtrů. Práce se podrobně zabývá zejména vývojem simulačního modelu, vhodného pro generování testovacích dat pro simulaci vybraného sledovacího algoritmu a vlastní rekonstrukční program, který byl následně paralelizován pomocí technologií distribuovaného počítání. Byly vytvořeny dva odlišné simulátory částic, jeden z nichž byl použit pro výpočet predikčního kroku Rozšířeného Kalmanova Filtru (EKF) a jeden pro generování testovacích dat pro vlastní rekonstrukční algoritmus. Pomocí správného rozdělení úkolů se podařilo distribuovaným algoritmem získat téměř osmisetprocentní zvýšení rychlosti a v některých případech i snížení paměťové náročnosti až o čtyři řády.

## Abstract

The focus of this work is to research various methods of particle track reconstruction in the CBM experiment, and the problem of hardware acceleration of these methods. The advantages and disadvantages of the extended methods were discussed and a reconstruction method based on cellular automata and Extended Kalman filters was selected for further study. In particular, the thesis details the development of a simulation model suitable for generating test data to facilitate the implementation of the selected tracking algorithm, which was subsequently sped up using distributed computing methods. Two different particle simulation models and a reconstruction algorithm were implemented, with the reconstruction algorithm offering up to 800 per cent speed up factor in respect to the sequential algorithm and up to four orders of magnitude lower memory complexity.

## Klíčová slova

Experiment CBM, akcelerace na hardwarových platformách, Kalmanův filtr, rozšířený Kalmanův filtr, celulární automaty, modelování a simulace, simulátor částic v magnetickém poli, částicová fyzika, kvark-gluonové plazma, kvantová chromodynamika, fyzika raného vesmíru, HEP experiment, distribuované počítání

## Keywords

CBM experiment, hardware acceleration, Kalman Filter, Extended Kalman Filter, Cellular automata, modelling and simulation, particle simulators in a magnetic field, High energy physics, quark-gluon plasma, quantum chromodynamics, physics of the early universe, HEP experiment, distributed computing

## Citace

ROTH, Michael. *Akcelerace algoritmu sledování částic v experimentu CBM*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr Musil, Ph.D.

# Akcelerace algoritmu sledování částic v experimentu CBM

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Musila, Ph.D. Další informace mi poskytli prof. Dr. Ivan Kisel a Mgr. Jan Jíša. Prohlašuji, že jsem uvedl všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....  
Michael Roth  
10. května 2023

## Poděkování

Děkuji panu doktoru Petru Musilovi za odborné vedení a ochotu pomoci ve všech fázích práce. Dále děkuji panu profesoru Ivanu Kiselovi za poskytnutí inspirace a pomoc při formulaci problému, ze kterého vzniklo zadání práce. Nakonec děkuji panu Mgr. Janu Jíšovi, za poskytnutí nedocenitelné odborné pomoci a konzultace v oblasti fyziky.



# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Experiment CBM a výzkumné středisko FAIR</b>	<b>4</b>
2.1	Pracoviště FAIR . . . . .	4
2.2	Experiment CBM . . . . .	8
<b>3</b>	<b>Teorie rekonstrukce trajektorií částic</b>	<b>11</b>
3.1	Rekurzivní Bayesovský odhad . . . . .	12
3.2	Kalmanův Filtr . . . . .	15
3.3	Metody rekonstrukce trajektorií . . . . .	23
<b>4</b>	<b>Návrh a implementace simulátorů částic v detektoru CBM</b>	<b>33</b>
4.1	Simulátor částic v detektoru . . . . .	33
4.2	Metody řešení diferenciálních rovnic . . . . .	35
4.3	Pohybové rovnice simulačního modelu, první simulační model . . . . .	41
4.4	Speciální relativita v pohybových rovnicích, druhý simulační model . . . . .	42
4.5	Kombinovaný simulátor částic s logováním zásahů v detektoru . . . . .	43
4.6	Pohyb částice v závislosti na vzdálenosti od terče detektoru, třetí simulační model . . . . .	48
<b>5</b>	<b>Návrh a implementace rekonstrukčního algoritmu</b>	<b>51</b>
5.1	Sekvenční verze rekonstrukčního algoritmu . . . . .	51
5.2	Rozdíly v implementaci oproti původnímu algoritmu . . . . .	55
5.3	Paralelní rekonstrukční algoritmus . . . . .	56
5.4	Alternativy k použití distribuovaného počítání . . . . .	60
<b>6</b>	<b>Závěr</b>	<b>62</b>
	<b>Literatura</b>	<b>64</b>

# Kapitola 1

## Úvod

Teorie velkého třesku, nejpopulárnější kosmologická teorie o vzniku a prvních okamžicích existence našeho vesmíru tvrdí, že vesmír, jak ho známe vznikl mohutnou „explozí“ a po miliardy let postupně chladl a rozpínal se do podoby, kterou známe dnes. Přestože současné teorie odhadují čas od velkého třesku na přibližně 13,8 miliardy let, o tom, jak vypadal náš vesmír v prvních okamžicích od svého vzniku víme jen velmi málo a přesný průběh velkého třesku nám není znám. Předpokládáme, že velký třesk probíhal ve fázích, kdy se jeho podoba drasticky lišila od podoby, jakou má dnes. Mezi prvními z těchto fází (řádově mikrosekundy od velkého třesku) byla pravděpodobně fáze, kdy veškerá hmota v mladém vesmíru se vyskytovala ve formě takzvaného **kvark-gluonového plazmatu**<sup>1</sup>, kdy veškerá hmota byla natolik hustá a horká, že se jednotlivé částice hmoty mohly volně a samostatně pohybovat (jev, který například u kvarků za normálních okolností nelze sledovat), aniž by je jakákoliv síla dokázala spoutat. Narozdíl od jiných forem plazmatu se tedy volně pohybovaly všechny elementární částice, aniž by se sdružovaly do větších celků, jako je tomu například u běžně sledovaných forem plazmatu, kdy sledujeme převážně volný pohyb elektronů a atomových jader, odtržených od jejich elektronových obalů. Kvark-gluonové plazma, jak již název napovídá, posouvá tento jev o dva kroky dál, kdy pozorujeme nejen rozpad jader na jednotlivé hadrony (např. neutrony, protony nebo mezony), ale i ty se dále rozpadají na jednotlivé kvarky a gluony, které se v QGP volně pohybují. Předpokládáme, že stav hmoty s podobnými vlastnostmi dnes můžeme ve vesmíru sledovat pouze za extrémních podmínek, například v jádrech neutronových hvězd.

Kvantová chromodynamika<sup>2</sup> je disciplína teoretické fyziky, zabývající se silnou interakcí mezi **barevně nabitými částicemi**, tedy interakcí mezi kvarky, zprostředkovanou částicí zvanou *gluon*, podobně jako je elektromagnetická interakce zprostředkovávána *fotony*. Barevně nabitě částice jsou specifické tím, že je za „normálních“ teplot a tlaků nelze pozorovat samostatně: vždy se vyskytují po trojicích (hadrony, například *proton* a *neutron*) nebo po dvojicích (mezony, například *pion*), zřídka také po čtveřicích, pěticích a šesticích (tetrakvarky, pentakvarky, hexakvarky). To je způsobeno jevem zvaným *barevné uvěznění*<sup>3</sup>, který mimo jiné říká, že dodáme-li kvarkům dostatek energie na to, aby se dokázali rozdělit, je energeticky výhodnější vytvořit nový pár kvark-antikvark. V případě, že stlačíme hmotu do tak extrémních tlaků, že jednotlivé *nukleony* nemají dost prostoru k tomu, aby mohly samostatně existovat, přestávají být kvarky drženy pohromadě gluony a mohou se volně pohybovat. Takové fázi hmoty říkáme *kvark-gluonové plazma*.

<sup>1</sup>z anglického QGP – Quark-Gluon Plasma

<sup>2</sup>z anglického QCD – Quantum Chromodynamics

<sup>3</sup>z anglického *colour confinement*

Jeden ze způsobů, jak můžeme laboratorně pozorovat a studovat strukturu hmoty za extrémních podmínek je pomocí řízených kolizí těžkých iontů s vysokou energií, jako jsou například jádra zlata či olova. Taková jádra urychlujeme na rychlosti velmi blízké rychlosti světla a necháváme s obrovskou energií kolidovat s terčí detektorů (v případě experimentu CBM vyrobený z tenkého proužku zlata), čímž jádra přivedeme k přechodnému stavu s extrémně vysokou hustotou a teplotou, který fyzikové nazývají *fireball* (ohnivá koule). Aby bylo možné vytvořit vhodné podmínky pro zkoumání hmoty za tak extrémních podmínek, musíme použít urychlovače s velmi vysokou energií paprsku. Experiment CBM, připravovaný experiment při moderním výzkumném středisku FAIR v německém Darmstadtu, bude hrát důležitou roli při zkoumání fázového diagramu kvantové chromodynamiky. Schopnost sledovat takto extrémní stav hmoty je ale vykoupena obrovským množstvím experimentálních dat (cca. 1 TB/s), které je potřeba zpracovávat *on-line*.

Rychlé zpracování velkého objemu hrubých experimentálních dat je netriviální problém, jehož vhodné řešení je stěžejní pro úspěšnost fyzikálního experimentu. Ve svých pracích se tímto problémem zabývali například prof. Dr. Ivan Kisel a Dr. Valentina Akishina, kteří se zabývali vývojem algoritmu rekonstrukce trajektorií částic pro potřeby detektoru CBM a jeho možnostmi akcelerace, v tomto případě pro platformu *Intel Xeon Phi*. Tato práce si klade za cíl prozkoumat alternativní způsoby akcelerace rekonstrukce trajektorií pro experiment CBM, zejména pak za pomoci protokolu MPI pro paralelizaci na mnohaprocesorových farmách. Kapitola 2 poskytuje stručný úvod do výzkumného střediska FAIR a experimentu CBM, kapitola 3 se poté dopodrobna zabývá jednotlivými algoritmy, použitými pro rekonstrukci trajektorií a aproximaci vlastností jejich částic a jejich zhodnocením proveditelnosti jejich akcelerace na hardwarových platformách. Kapitola 4 se zabývá vývojem vhodného simulátoru pohybu částic uvnitř detektoru CBM a jeho použitím v rámci vlastního algoritmu sledování částic. Nakonec Kapitola 5 popisuje vlastní implementaci rekonstrukčního algoritmu, jeho akceleraci pomocí metod distribuovaného počítání a diskutuje další možné způsoby akcelerace výpočtu.

## Kapitola 2

# Experiment CBM a výzkumné středisko FAIR

Tato kapitola je věnována stručnému popisu pracoviště FAIR<sup>1</sup> a jeho výpočetní infrastruktury, fyzikálního experimentu CBM<sup>2</sup> a jeho cílů. Kapitola se velmi stručně zabývá detektorem STS<sup>3</sup> a supravodivým dipólovým magnetem, ve kterém se STS nachází. Zvláštní pozornost je pak věnována výpočetní farmě při GSI **Green IT Cube**.

### 2.1 Pracoviště FAIR

Facility for Antiproton and Ion Research (FAIR) je mezinárodní výzkumné centrum, v současné době ve výstavbě v německém Darmstadtu při Helmholtzově centru pro výzkum těžkých iontů GSI<sup>4</sup>. Projekt je realizován partnery z Finska, Francie, Indie, Německa, Polska, Rumunska, Ruska, Slovinska, Švédska a Velké Británie. Česká republika se stala aspirujícím partnerem v roce 2018. [19, 12] FAIR se v budoucnosti stane jedním z největších a nejsložitějších urychlovačů na světě. Výsadou urychlovače bude jeho schopnost produkovat svazky částic všech prvků periodické tabulky (nebo jejich iontů), případně pak svazky antiprotonů, urychlené až na 99% rychlosti světla. [11]

#### Stávající infrastruktura GSI

Stávající zařízení GSI se v budoucnu stanou součástí infrastruktury FAIR a jeho dva urychlovače budou sloužit jako první urychlovací stupeň pro připravované experimenty. Lineární urychlovač UNILAC<sup>5</sup>, dlouhý 120 metrů, urychluje částice až na 20% rychlosti světla. Odsud částice putují k prvním experimentům a prvnímu prstencovému urychlovači SIS18<sup>6</sup> o obvodu 120 metrů, který částice dále urychluje až na 90% rychlosti světla a dále vysílá ke stávajícím experimentům GSI a úložným prstencům. [11]

---

<sup>1</sup>FAIR – Facility for Antiproton and Ion Research (středisko pro výzkum antiprotonů a iontů)

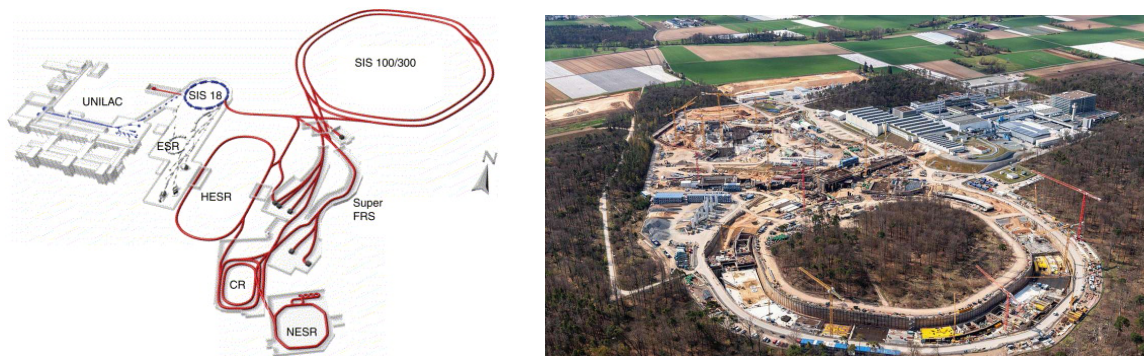
<sup>2</sup>CBM – Compressed Baryonic Matter (stlačená baryonová hmota)

<sup>3</sup>STS – Silicon Tracking System (křemíkový sledovací systém)

<sup>4</sup>GSI – Gesellschaft für Schwerionenforschung (sdružení pro výzkum těžkých iontů)

<sup>5</sup>UNILAC – UNiversal Linear ACcelerator (univerzální lineární urychlovač)

<sup>6</sup>SIS – Schwerionensynchrotron (synchrotron pro těžké ionty)



Obrázek 2.1: Rozvržení objektu FAIR (vlevo). Stávající urychlovače GSI UNILAC a SIS18 (vyobrazené modře) budou kromě obsluhy stávajícího komplexu GSI (vyobrazeno šedě) sloužit jako injektor pro dva velké synchrotrony SIS100 a SIS300 (červeně). Částicové svazky budou rozváděny do různých experimentálních stanic nebo k terčovým stanovištím pro experimenty vyžadující svazky antiprotonů nebo radioaktivních izotopů. Převzato z [3]. Fotografie areálu GSI a staveniště FAIR (vpravo), převzatá z [21], pořízená v dubnu 2021.

## Prstencové urychlovače SIS100 a SIS300

FAIR bude pracovat s částicovými svazky s vysokými intenzitami iontů ( $10^9$  částic za sekundu v případě iontů zlata) nebo pozitronů ( $10^{13}$  částic za sekundu). [1] V současné době jsou ve výstavbě nové prstencové urychlovače SIS100 a SIS300, uloženy v podzemních tunelech v hloubce až 17 metrů s obvodem 1100 metrů. Částice budou do urychlovačů SIS100 a SIS300 vháněny z urychlovače SIS18 a dále urychleny až na 99% rychlosti světla. Takto urychlené částice jsou pak buďto použity přímo k experimentům, k přípravě tzv. sekundárních částic, nebo vešněny do tzv. úložných prstenců k pozdějšímu použití. [11]

## Úložné prstence

K prstencovým urychlovačům je připojen komplexní systém úložných prstenců a dalších experimentálních stanic. Úložné prstence vytvářejí pomocí silného magnetického pole cyklickou trajektorii příchozích svazků, na rozdíl od prstencových urychlovačů ale svazkům již nedodávají žádnou energii, tedy je již dále neurychlují. Tento přístup je výhodný, protože příprava vysokoenergetických svazků je náročná a bez použití úložných prstenců jsou připravené svazky po průletu experimentálním stanovištěm ztraceny. Použitím úložných prstenců zajistíme, že vědecké týmy mohou na stejných částicích pracovat při každém průletu experimentálním stanovištěm a dále manipulovat s vlastnostmi svazku. Navíc použití úložných prstenců de facto dále zvyšuje intenzitu svazku bez nutnosti dalšího použití urychlovače.

Pro skladování a manipulaci se svazky se počítá s několika prstenci, včetně vysokoenergetického prstence HESR<sup>7</sup>, určeného k práci s antiprotonovými svazky, sběrného prstence CR<sup>8</sup> a experimentálního prstence NESR<sup>9</sup>. Komplex úložných prstenců umožňuje chlazení svazků a zahrnuje terčová stanoviště pro rozptylové experimenty. [3]



Obrázek 2.2: Fotografie datového centra Green IT Cube, Administrativní budova GSI je viditelná vpravo. Obrázek byl převzat z webových stránek GSI [17].

## Green IT Cube

V roce 2008 se odhadovalo, že informační a komunikační technologie byly zodpovědné za 2% světové produkce skleníkových plynů, což je množství srovnatelné s emisemi letadlové dopravy. [23]. Firma Intel v roce 2012 odhadovala rozložení emisí své ICT<sup>10</sup> infrastruktury následovně: [6]

- 6% připadá na uživatelskou výpočetní techniku (kancelářské počítače, monitory, tiskárny apod.),
- 24% připadá na ICT infrastrukturu mimo datová centra,
- 70% připadá na datová centra.

Podle výpočtů institutu Borderstep činila již v roce 2008 spotřeba elektrické energie datových center v Německu 10,1 TWh, což odpovídá 1,8% celkové německé spotřeby toho roku a celkovým nákladům přibližně 1,1 miliardy eur. Přitom výpočetní výkon (a s ním i nároky na výkon chlazení) datových center se neustále zvyšuje: Studie stejného institutu a autora o deset let později uvádí, že v roce 2017 činila spotřeba energie datových center v Německu 13,2 Terawatthodin.

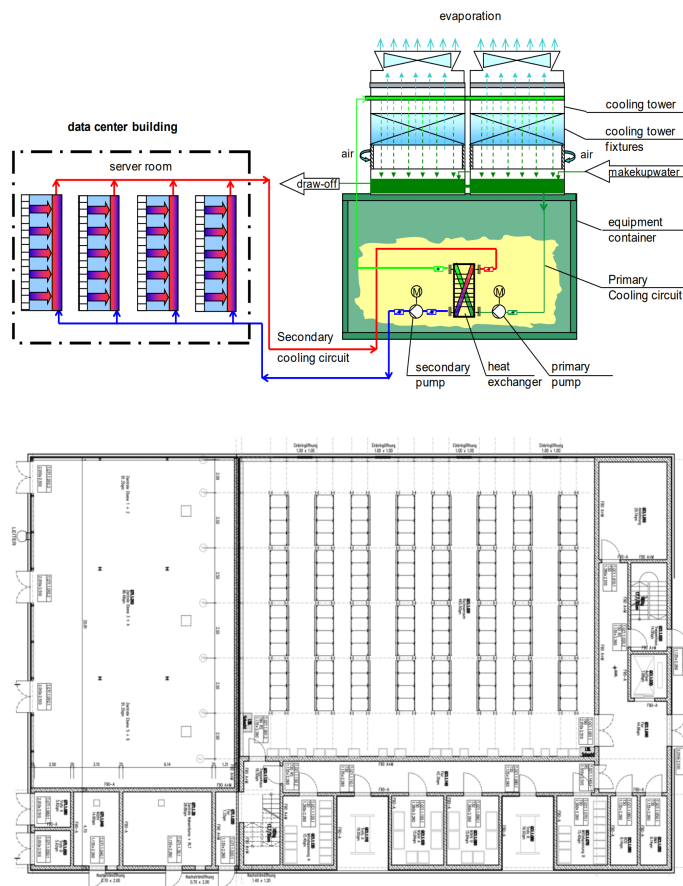
Podle EYP Mission Critical Facilities Inc., New York [23], typicky ne víc než 50% energie potřebné k provozu datových center je využito na vlastní napájení IT vybavení. Uvedme

<sup>7</sup>HESR – High Energy Storage Ring (vysokoenergetický úložný prstenec)

<sup>8</sup>CR – Collector Ring (sběrný prstenec)

<sup>9</sup>NESR – New Experimental Storage Ring (nový experimentální úložný prstenec)

<sup>10</sup>ICT – Information and Communication Technologies (informační a komunikační technologie)



Obrázek 2.3: Diagram chladicího systému datového centra Green IT Cube (nahore), půdorys přízemního patra budovy (dole). Obrázky byly přejaty z [23].

si metriku pro určení energetické efektivity Datového centra: **PUE**<sup>11</sup> je metrika, která zkoumá, jak velkou část celkové energetické spotřeby datového centra vyžaduje napájení výpočetní techniky:

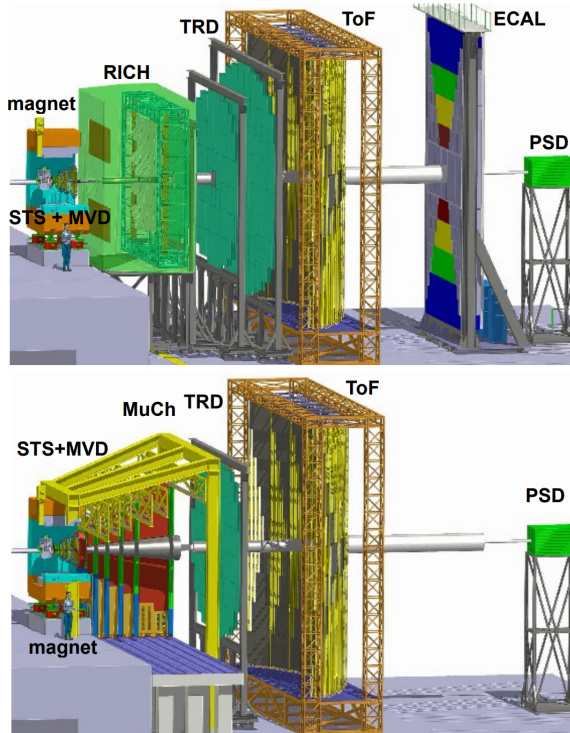
$$\mathbf{PUE} = \frac{\text{Celková spotřeba objektu}}{\text{Spotřeba výpočetní techniky}} \quad (2.1)$$

Přičemž PUE typického datového centra se může pohybovat kdekoli mezi 1,6 a 3,5, a teoretické PUE = 1 by mělo takové datové centrum, které využije veškerou svoji spotřebu energie pouze k napájení výpočetní techniky.

Součástí pracoviště GSI je nové datové centrum **Green IT Cube**, které si mimo jiné klade za cíl prozkoumat způsoby, jak snížit zátěž datových center na životní prostředí. Green IT Cube je postaveno tak, aby mohlo být osazeno 768 standardními devatenáctipalcovými rozvaděči v šesti patrech budovy. Zařízení disponuje kapacitou až 15 MW plně redundantního chladicího výkonu s PUE menším než 1,07. Takové efektivity bylo dosaženo pomocí speciálního dvouokruhového vodního chlazení, kdy sekundární okruh uvnitř datového centra chladí jednotlivé rozvaděče (nikoliv jednotlivá zařízení v nich, díky čemuž je možné rozvaděče osadit výrazně širším výběrem výpočetní techniky), výměník tepla v

<sup>11</sup>PUE – Power Usage Effectiveness (efektivita spotřeby energie)





Obrázek 2.4: Rozvržení experimentu CBM v jeho dvou operačních nastaveních: nastavení pro detekci elektronů (obrázek nahoře) a pro detekci mionů (obrázek dole). Pořadí součástí detektoru v elektronové konfiguraci je následující: Micro Vertex Detector (MVD) a Silicon Tracking System (STS), oba usazené uvnitř silného supravodivého magnetu, Ring Imaging Cherenkov Detector (RICH), Transition Radiation Detector (TRD, dva za sebou) Time of Flight (ToF) detektor, elektromagnetický kalorimetr (ECAL) a Projectile Spectator Detector (PSD) funguje jako hadronový kalorimetr. V mionové konfiguraci je odstraněn elektromagnetický kalorimetr a RICH detektor je nahrazen systémem mionových komnat (Muon Chambers system – MUCH) Převzato z [1].

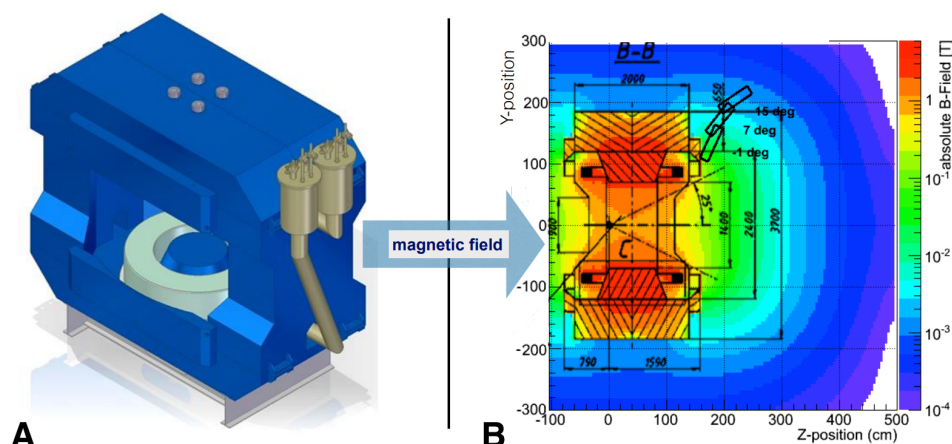
přízemí budovy pak odpadní teplo převádí do primárního okruhu, které je pak částečně využito k vytápění kanceláří a jídelny budovy GSI (viz obrázek 2.3). Po uvedení do plného výkonu bude mít pracoviště FAIR k dispozici výpočetní výkon tří set tisíc procesorových jader, až 135 Petabajtů (35 PB pro stávající infrastrukturu GSI a 100 PB v rámci rozšíření pro FAIR) online úložiště na pevných discích a robota s 4280ti sloty pro magnetické pásky pro archivaci až 62 PB nekomprimovaných experimentálních dat. Pro experimenty FAIR se dále počítá s vysokou přenosovou rychlostí přes 1 Terabajt za sekundu pro přenos velkého objemu nezpracovaných experimentálních dat z pracovišť jednotlivých experimentů[18].

Při uvedení do provozu v roce 2014 obdrželo centrum Green IT Cube mnohá ocenění, včetně prvního místa v žebříčku superpočítačů Green500, které řadí nejefektivnější superpočítače podle GFLOPS/W.

## 2.2 Experiment CBM

CBM je jeden z vědeckých pilířů pracoviště FAIR. Jeho cílem bude studovat strukturu a stavovou rovnici baryonové hmoty o hustotách, které bychom mohli najít v jádrech ne-





Obrázek 2.5: Nákres supravodivého magnetu (A) a znázornění intenzity magnetického pole v magnetu (B). V bodě (0,0) je znázorněn terč detektoru, graf intenzity je vykreslen podél roviny YZ,  $x = 0$ . Převzato z [1].

utronových hvězd. Experiment bude zkoumat srážky svazků protonů a těžkých iontů se statickým terčem při svazkových energiích od 2 do 45 AGeV (GeV na nukleon). [1] Experiment bude zaměřen mimo jiné na [14]:

- studium stavové rovnice jaderné hmoty o hustotách, které bychom mohli najít v neutronových hvězdách
- hledání fázové hranice mezi hadronovou fází a kvark-gluonovým plazmatem, případně oblasti fázové koexistence a koncového kritického bodu kvantové chromodynamiky.
- pátrání po jednoduchých a dvojitých hyperjádrech: těžkých objektech s nenulovým kvantovým číslem podivnosti

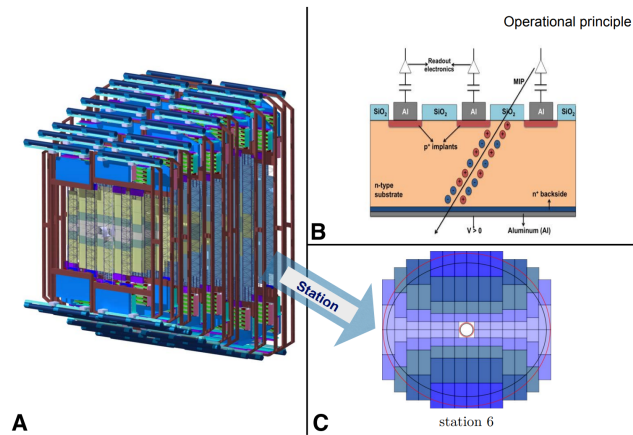
Experiment CBM je velmi rozsáhlý a složitý detektor, stávající se z mnoha podsystémů, my se však v rámci této práce budeme zabývat pouze jeho hlavním detektorem **STS**.

## Supravodivý magnet

Účel silného supravodivého magnetu je *ohýbat* nabitě částice, u kterých pak podle jejich zakřivení dokážeme určit jejich náboj a energii. K tomu, aby bylo možné uvnitř magnetu uložit detektor STS bylo potřeba, aby měl magnet rozměry minimálně  $1,3 \times 1,3 \text{ m}^2$  a generovat dostatečně silné magnetické pole, aby bylo magnetické pole uvnitř prvního detektoru „co nejhomogennější“. Magnet funguje na bázi velké Helmholtzovy cívky, která opravdu uvnitř sebe sama vytváří magnetické pole, které se za určitých podmínek dá považovat za homogenní.

## Silicon Tracking System

Silicon Tracking System (STS) je primární detektor experimentu CBM a druhý detektor v pořadí po detektoru Minimum Vertex Detector (MVD). Cílem detektoru STS je vhodný



Obrázek 2.6: 3D pohled na detektor STS a rozvržení jeho stanovišť (A), diagram křemíkového proužkového detektoru (B) a schéma rozvržení šesti hladiny detektoru STS (světlejší barva proužku značí vytíženější oblasti detektoru) (C). Převzato z [1]

odhad vlastností jednotlivých trajektorií částic, kterých může detektorem proletět během jedné „události“ při vzorkovací frekvenci 10 MHz až 700. Detektor je uspořádán do osmi, na podpůrných lešení z uhlíkových vláken upevněných „hladin“: stanovišť, kdy první je od terče detektoru vzdáleno 30 centimetrů, mezi každou hladinou detektoru je pak vždy deset centimetrů prostor. Detektorové hladiny jsou uspořádány tak, aby pokrývaly polární úhly od 2,5° do 25°. Pro polární úhly menší než 2,5° prochází detektorem silný „paprsek“<sup>12</sup> hadronů, u kterého je vzhledem k vysokým energiím a množství částic detekce nepraktická.

STS má citlivé elektrody na každé straně detektoru, kde každá strana se stará o odečítání jedné ze dvou souřadnic, přitom proužky nesvírají pravý úhel, nýbrž úhel 15°, čímž se sice zmenší rozlišení detektoru podél jedné osy ( $y$ ), zvýší se ale rozlišení druhé osy ( $x$ ). Detektor je takto zařízen z toho důvodu, že nabitě částice jsou ovlivněny magnetickým polem silného supravodivého magnetu, ve kterém se detektor nachází, v důsledku čehož mají částice uvnitř magnetu mnohem větší rozptyl ve směru osy  $x$ , než ve směru osy  $y$ . Algoritmus, který je v této práci použit pro rekonstrukci trajektorií částic, tento fakt zohledňuje upraveným modelem měření  $H$ :

$$H = \begin{bmatrix} \cos \alpha & 0 & 0 & 0 & 0 \\ 0 & \sin \alpha & 0 & 0 & 0 \end{bmatrix} \quad (2.2)$$

kde  $\alpha$  je úhel mezi jednotlivými osami detektoru, v případě STS  $\alpha = 15^\circ$ . Model měření Kalmanova Filtru diskutujeme podrobně v kapitole 3.

<sup>12</sup>z angl. **Jet**

## Kapitola 3

# Teorie rekonstrukce trajektorií částic

Začátek následující podkapitoly byl převzat z disertační práce Dr. Akishiny [1]. Díky neustálému vývoji v oborech **urychlovačové fyziky**<sup>1</sup> a výpočetní techniky je dnes možné provozovat HEP experimenty při energiích a četnostech kolizí, které dříve nebyly možné. Tento fakt má ale také za důsledek, že zpracování stále se zvětšujících objemů experimentálních dat je čím dál komplikovanější úkol, na který již běžně dostupná výpočetní technika není vhodná.

Výrazně vyšší kolizní energie částic v HEP experimentech způsobují komplikovanější vzorce událostí a výrazně větší množství sledovaných částic v detektoru. Zároveň se zvyšuje také frekvence těchto událostí, což má za následek omezení času, který máme k dispozici na vyhodnocení každé takové události, tedy k rekonstrukci trajektorií jednotlivých částic a srážek mezi nimi.

Vzorce srážek rekonstruujeme tak, že nejprve zrekonstruujeme trajektorie jednotlivých částic, které při průletu detektorem zanechávají měřitelné stopy na jednotlivých detektorových hladinách, tzv. **zásahy**. Proces rekonstrukce každé události probíhá v experimentu CBM ve Čtyřech fázích:

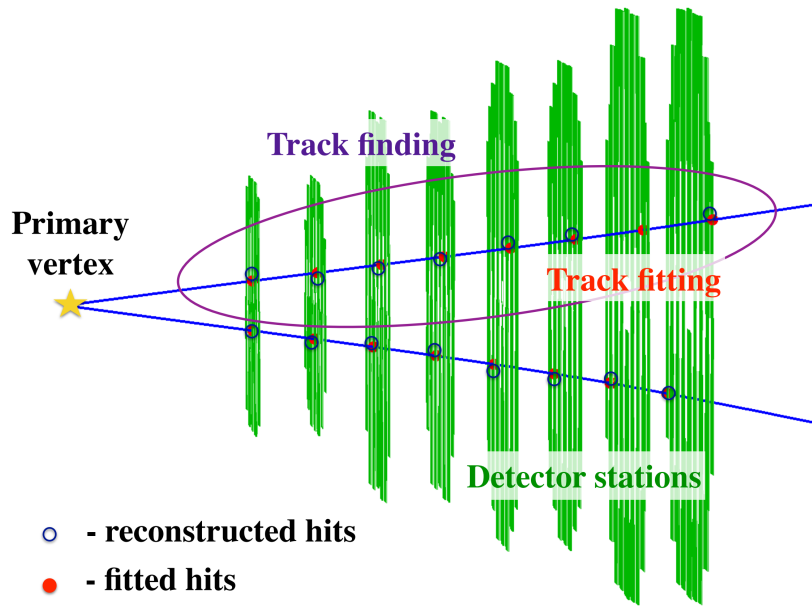
- hledání trajektorií,
- prokládání trajektorií,
- identifikace částic podle proložených trajektorií,
- hledání částic s krátkou dobou života.

Přestože hledání a prokládání trajektorií částic spolu do značné míry souvisí, tradičně na ně pohlížíme jako na dvě odlišné fáze rekonstrukce. Zatímco hledání trajektorií je problém rozpoznávání vzorů, kde jednotlivé zásahy rozdělujeme do skupin, o kterých předpokládáme, že by mohly být vyvolány stejnou částicí. Po tom, co jednotlivé zásahy takto seskupíme, lze pro každou skupinu odhadnout nejpravděpodobnější parametry dráhy částice, a to i za přítomnosti velkého množství šumu. Tomuto postupu říkáme prokládání trajektorií.

Proces hledání trajektorií pracuje zpravidla nad nezpracovanými, silně zašuměnými daty, která nemohla být předem nijak předzpracována ani zredukována. Z tohoto důvodu je pro-

---

<sup>1</sup>z angl. accelerator physics – odnož aplikované fyziky, zabývající se návrhem, stavbou a provozem urychlovačů částic



Obrázek 3.1: První dvě fáze rekonstrukce trajektorií. Převzato z [1].

ces hledání trajektorií považován za nejsložitější a časově nejnáročnější fázi rekonstrukce trajektorií.

Jednotlivé fáze rekonstrukce trajektorií se ale postupně stávají čím dál provázanějšími a čím silnější jsou souvislosti mezi nimi, tím jsou hranice mezi nimi mlhavější. V poslední době je běžné během první rekonstrukční fáze odhadovat jednotlivé parametry dráhy částice v každém kroku, díky čemuž získáváme s každým dalším měřením lepší povědomí o parametrech proložené dráhy, což nám dále umožňuje již v této první fázi vylučovat zásahy, u kterých je nepravděpodobné, že budou patřit k právě rekonstruované dráze. Tento trend je primárně určován existencí rekurzivních algoritmů užívajících metody Kalmanovy Filtrace. Tyto algoritmy jsou hlavním tématem této práce a metodami Bayesovských Filtrů se budeme dopodrobna zabývat v příští podkapitole.

### 3.1 Rekurzivní Bayesovský odhad

Rekurzivní Bayesovský odhad, též Bayesovský Filtr, je obecný algoritmus pro výpočet odhadu stavu systému na základě jednotlivých měření systému:

$$bel(x_t) = p(x_t | z_{1:t}, u_{1:t}) \quad (3.1)$$

kde  $x_t$  značí stav systému v čase  $t$ ,  $bel(x_t)$  značí předpokládaný stav v čase  $t$ ,  $z_t$  značí měření v čase  $t$  a  $u_t$  značí **ovládání systému**<sup>2</sup> v čase  $t$ . Stěžejní zákon pro odvození Bayesovského Filtru je *Bayesův teorém*, který zní následovně:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \quad (3.2)$$

<sup>2</sup>z angl. system control commands

kde  $A$  a  $B$  jsou náhodné jevy,  $p(A)$  je pravděpodobnost jevu  $A$  a  $p(A|B)$  je pravděpodobnost jevu  $A$  za předpokladu, že je jev  $B$  pravdivý. Pro účely odvození Bayesovských filtrů je užitečnější aplikace Bayesova teorému pro 3 náhodné jevy:

$$p(A|B, C) = \frac{p(B|A, C)p(A|C)}{p(B|C)} \quad (3.3)$$

Když dosadíme definici Bayesovského filtru (rovnice 3.1) do rovnice výše, dostaneme:

$$p(x_t|z_t, z_{1:t-1}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \quad (3.4)$$

Pro lepší čitelnost zavedeme za jmenovatel rovnice 3.4 normalizační konstantu  $\eta$ :

$$\eta = \frac{1}{p(z_t|z_{1:t-1}, u_{1:t})} \quad (3.5)$$

$$bel(x_t) = \eta p(z_t|x_t, z_{1:t-1}, u_{1:t}) p(x_t|z_{1:t-1}, u_{1:t}) \quad (3.6)$$

Pro odvození rekurzivního vzorce Bayesovského filtru použijeme fakt, že skutečný stav systému  $x$  je **Markovův proces se skrytými stavy**. Na rozdíl od klasických *Markovových řetězců* neznáme vnitřní stav systému, o tomto stavu získáváme informace pouze za pomoci měření  $z_k$ . Protože je vnitřní stav systému dán Markovovým procesem, víme, že bude systém vykazovat *Markovovu vlastnost*, která říká, že vývoj Markovova procesu je závislý pouze na jeho předchozím stavu. Díky této vlastnosti víme, že pokud známe stav  $x_t$ , k určení pravděpodobnosti měření  $z_t$  nepotřebujeme žádné znalosti, protože pravděpodobnost  $p(z_t|x_t)$  nijak neovlivní:

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \quad (3.7)$$

Rovnici 3.6 tedy můžeme zjednodušit následujícím způsobem:

$$bel(x_t) = \eta p(z_t|x_t) p(x_t|z_{1:t-1}, u_{1:t}) \quad (3.8)$$

V dalším kroku aplikujeme **pravidlo úplné pravděpodobnosti**, které říká, že máme-li úplný systém jevů  $\{B_n : n = 1, 2, 3, \dots\}$ , pak pravděpodobnost libovolného jevu  $A$  lze určit jako:

$$p(A|C) = \sum_{i=1}^n p(A|B_i, C) p(B_i|C) \quad (3.9)$$

V našem případě vzorec úplné pravděpodobnosti (v jeho spojitě podobě) umožní přidat k druhému členu rovnice 3.8 další podmínku, kterou využijeme následně při odvození rekurzivního kroku filtru:

$$p(a|c) = \int_{-\infty}^{\infty} p(a|b, c) p(b|c) db \quad (3.10)$$

Dosadíme-li do původní rovnice, dostaneme:

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) p(x_{t-1}|z_{1:t-1}, u_{1:t}) dx_{t-1} \quad (3.11)$$

Nyní můžeme zjednodušit člen  $p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})$  pomocí Markovovy vlastnosti, díky které víme, že ke stanovení pravděpodobnosti, že se nacházíme ve stavu  $x_t$ , nám stačí znát

předchozí stav systému  $x_{t-1}$  a informaci o tom, jak se systém vyvíjí ze stavu  $x_{t-1}$  do stavu  $x_t$  (tuto znalost reprezentujeme jako  $u_t$ ). Všechny ostatní znalosti jsou pro tento výpočet irelevantní, takže tento člen můžeme pomocí Markovovy vlastnosti přepsat jako:

$$p(x_t|x_{t-1}, z_{1:t}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \quad (3.12)$$

Výsledná rovnice má pak následující tvar:

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) p(x_{t-1}|z_{1:t-1}, u_{1:t}) dx_{t-1} \quad (3.13)$$

V následujícím kroku zkoumáme člen  $p(x_{t-1}|z_{1:t-1}, u_{1:t})$ : zkoumáme pravděpodobnost, že minulý stav systému byl  $x_{t-1}$  za předpokladu, že známe veškerá předešlá měření až do času  $t-1$ , stejně tak víme, jak se systém vyvíjel v každém okamžiku až do času  $t-1$ . Zároveň je ale přítomna podmínka  $u_t$ , která říká, že víme, jakým způsobem se systém vyvine ze stavu  $x_{t-1}$  do stavu  $x_t$ . Můžeme předpokládat, že znalost, jakým způsobem se bude systém v budoucnu vyvíjet, nemá na pravděpodobnost, že v čase  $t-1$  bude systém ve stavu  $x_{t-1}$  žádný vliv, a tuto podmínku ze člena  $p(x_{t-1}|z_{1:t-1}, u_{1:t})$  vyloučit:

$$p(x_{t-1}|z_{1:t-1}, u_{1:t}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) \quad (3.14)$$

Dosazením získáváme tvar rovnice:

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) dx_{t-1} \quad (3.15)$$

Konečně, z původní definice odhadu  $bel(x_t)$  v rovnici 3.1 vyplývá, že člen

$$p(x_{t-1}|z_{1:t-1}, u_{1:t-1})$$

můžeme přepsat jako:

$$p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) = bel(x_{t-1}) \quad (3.16)$$

Čímž získáme potřebný rekurzivní krok a můžeme dosazením do rovnice 3.13 dokončit odvození Bayesovského filtru:

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \quad (3.17)$$

Výsledný filtr můžeme dále rozepsat (a typicky rozepisujeme) na následující dva kroky:

- predikční krok

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \quad (3.18)$$

- korekční krok

$$bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t) \quad (3.19)$$

kde  $\overline{bel}(x_t)$  je odhad  $x_t$  na základě předchozích měření,  $bel(x_t)$  je upravený odhad  $x_t$  na základě nového měření,  $p(x_t|x_{t-1}, u_t)$  v predikčním kroku nazýváme *pohybový model* a  $p(z_t|x_t)$  v korekčním kroku nazýváme *model měření*.

## Implementace Bayesovského Filtru

Rekurzivní Bayesův odhad je obecný pravděpodobnostní přístup k rekurzivnímu odhadu neznámého rozložení pravděpodobnosti; v závislosti na typu řešeného problému vybíráme jednu z implementací Bayesovského filtru. Existuje více možných implementací Bayesovských filtrů, zejména pak:

- **Kalmanův filtr (KF)**, optimální Bayesovský filtr pro filtrování nad normálními rozděleními a lineárními modely,
- **Rozšířený Kalmanův filtr (EKF)**, rozšíření Kalmanova filtru pro nelineární modely. Rozšířený Kalmanův filtr je v praxi nejrozšířenější metoda pro aproximaci pohybu částic.
- **Částicové filtry**<sup>3</sup>, jinak také metody SMC<sup>4</sup>, pracují nad nelineárními modely s arbitrárními rozloženími pravděpodobností. Díky tomu jsou ze zde zmíněných metod použitelné pro nejširší spektrum problémů, pro naše účely jsou ale z důvodu nízké rychlosti nevhodné.

Právě Kalmanovými filtry a EKF se budeme v další podkapitole podrobně zabývat.

## 3.2 Kalmanův Filtr

**Kalmanův Filtr**, známý také jako metoda **LQE**<sup>5</sup>, je implementace Bayesova Filtru, která poskytuje optimální odhad stavu dynamického systému za předpokladu, že:

- pohybový model i model měření jsou lineární
- všechny náhodné proměnné v systému mají *normální rozdělení*

Metoda slouží k získání optimálního odhadu  $\mathbf{r}$  neznámého stavového vektoru  $\mathbf{r}^t$  za pomoci *vektorů měření* (jinak také *pozorování*)  $\mathbf{m}_k, k = 1 \dots n$  vektoru  $\mathbf{r}^t$ . Metoda je inicializována volbou počátečního odhadu  $\mathbf{r} = \mathbf{r}_0$ <sup>6</sup> a postupně s každým dalším měřením zpřesňuje odhad  $\mathbf{r}$  stavového vektoru  $\mathbf{r}^t$ . [16].

Hodnota vektoru  $\mathbf{r}^t$  se může v čase měnit: jeho změnu u lineárních modelů reprezentujeme maticí  $F_k$ , která obsahuje informaci o tom, jak se v čase mění každá složka stavového vektoru  $\mathbf{r}^t$ . Stav systému se tak mezi jednotlivými měřeními mění, nemusíme ale přesně vědět, jak a/nebo kdy k těmto změnám dochází. Veškeré náhodné jevy mezi měřeními  $\mathbf{m}_{k-1}$  a  $\mathbf{m}_k$ , které nedokážeme modelovat v rámci propagační matice  $F_k$ , modelujeme pomocí takzvaného *procesního šumu*, který je reprezentován vektorem  $\boldsymbol{\nu}_k$ . Výsledná rovnice změny stavového vektoru  $\mathbf{r}^t$  vypadá následovně:

$$\mathbf{r}_k^t = F_k \mathbf{r}_{k-1}^t + \boldsymbol{\nu}_k \quad (3.20)$$

kde  $F_k$  je lineární propagační operátor a  $\boldsymbol{\nu}_k$  je vektor procesního šumu mezi měřeními  $\mathbf{m}_{k-1}$  a  $\mathbf{m}_k$ . V případě Kalmanova Filtru předpokládáme, že měření  $\mathbf{m}_k$  je lineárně závislé na stavu systému  $\mathbf{r}_k^t$ , tedy říkáme, že **model měření**  $H_k$  je lineární:

---

<sup>3</sup>z angl. Particle Filters

<sup>4</sup>SMC – Sequential Monte Carlo (sekvenční Monte Carlo metody)

<sup>5</sup>LQE – Linear Quadratic Estimation (lineárně-kvadratický odhad)

<sup>6</sup>U EKF musíme být narozdíl od klasického KF při volbě  $\mathbf{r}_0$  opatrní, metoda může při špatně zvolené počáteční hodnotě divergovat

$$\mathbf{m}_k = H_k \mathbf{r}_k^t + \boldsymbol{\eta}_k \quad (3.21)$$

kde  $\boldsymbol{\eta}_k$  je vektor chyby měření  $\mathbf{m}_k$ ,  $H_k$  je *model měření*. Model měření je matice s dimenzí  $m \times n$ , kde  $m$  je rovno velikosti stavového vektoru  $\mathbf{r}^t$  a  $n$  je rovno velikosti vektoru měření  $\mathbf{m}_k$ . Model měření určuje, jaký je vztah mezi jednotlivými měřeními systému a jeho reálným stavem. Kalmanův Filtr předpokládá, že chyba měření a procesní šum nijak nekorelují a že známe jejich **kovarianční matice**:

$$\begin{aligned} \langle \boldsymbol{\eta}_i \rangle &= \langle \boldsymbol{\nu}_j \rangle = \mathbf{0}, \\ \langle \boldsymbol{\eta}_k \cdot \boldsymbol{\eta}_k^T \rangle &= V_k, \\ \langle \boldsymbol{\nu}_k \cdot \boldsymbol{\nu}_k^T \rangle &= Q_k. \end{aligned} \quad (3.22)$$

kde notace  $\langle \mathbf{X} \rangle = E(\mathbf{X})$  značí střední hodnotu náhodné veličiny  $\mathbf{X}$ . **Kovarianční matice** (také variačně-kovarianční matice) reálné vícerozměrné náhodné veličiny  $X$  o  $n$  složkách je čtvercová matice ( $n \times n$ ), která obsahuje **kovarianci**  $i$ -té a  $j$ -té složky náhodné veličiny  $\mathbf{X}$ :

$$X = \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2 & \cdots & \sigma_1\sigma_n \\ \sigma_2\sigma_1 & \sigma_2^2 & \cdots & \sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_n\sigma_1 & \sigma_n\sigma_2 & \cdots & \sigma_n^2 \end{bmatrix}$$

kde  $\sigma_i\sigma_j = cov(X_i, X_j) = E[(X_i - E[X_i])(X_j - E[X_j])]$  je kovariance veličin  $X_i$  a  $X_j$  a  $E[X_i] = \langle X_i \rangle$  je střední hodnota veličiny  $X_i$ .

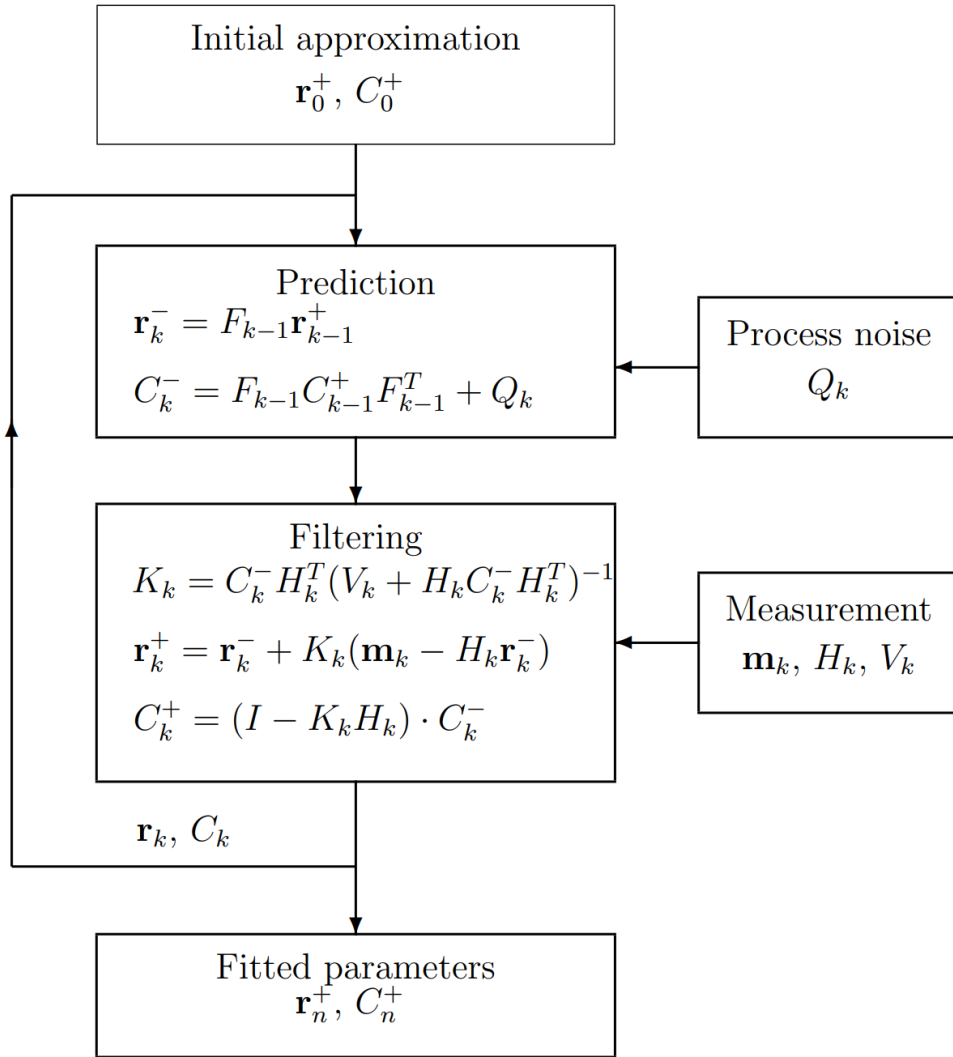
Klasický Kalmanův Filtr běžně členíme do čtyř základních kroků:

1. **inicializace**, kdy zvolíme počáteční hodnoty odhadu stavového vektoru  $\mathbf{r}_0^+$  a kovarianční matice  $C_0^+$ ,
2. **predikce**, kdy pomocí propagačního operátoru  $F_k$  stanovíme odhad dalšího stavu stavového vektoru a kovarianční matice,
3. **zpracování procesního šumu**, kdy upravíme odhad příští kovarianční matice  $C_k^-$  pomocí kovarianční matice procesního šumu  $Q_k$ ,
4. **filtrace**, kdy pomocí nového měření upravíme odhad stavového vektoru a kovarianční matice tak, aby příští odhady byly přesnější.

**Inicializace** Během inicializace Kalmanova Filtru musíme určit první aproximaci stavového vektoru  $\mathbf{r}_0^+$  a kovarianční matice  $C_0^+$ : Mějme reálný stav systému, vyjádřený vektorem  $\mathbf{r}^t$  a jeho odhad  $\mathbf{r}$ . Odhad stavu systému  $\mathbf{r}$  má vždy konečnou přesnost: zavedeme vektor chyby  $\boldsymbol{\xi}$ , který vyjadřuje rozdíl mezi reálným stavem systému a jeho odhadem a kovarianční matici  $C$ , které definujeme následovně:

$$\begin{aligned} \mathbf{r} &= \mathbf{r}^t + \boldsymbol{\xi} \\ C &= \langle \boldsymbol{\xi} \cdot \boldsymbol{\xi}^T \rangle \end{aligned} \quad (3.23)$$





Obrázek 3.2: Blokové schéma klasického Kalmanova Filtru. Převzato z [16]

Podle dostupných informací zvolíme počáteční odhad vektoru  $\mathbf{r}_0^+$ , a jeho kovarianční matici potřebujeme zvolit tak, aby reprezentovala fakt, že před prvním měřením nemáme o stavu systému žádné znalosti. Kovarianční matici  $C_0$  proto zvolíme následovně:

$$C_0^+ = \mathbf{I} \cdot inf^2 \quad (3.24)$$

kde  $\mathbf{I}$  je jednotková matice a *inf* značí velké kladné číslo.

**Predikce a zpracování procesního šumu** Pokud očekáváme, že se stav systému mezi jednotlivými měřeními změní, musí se odpovídajícím způsobem změnit i jeho odhad a kovarianční matice: Pomocí propagačního operátoru  $F_{k-1}$  získáme predikci stavu systému  $\mathbf{r}_k^-$  a jeho kovarianční matice  $C_k^-$ :

$$\begin{aligned}\mathbf{r}_k^- &= F_{k-1}\mathbf{r}_{k-1}^+ \\ C_k^- &= F_{k-1}C_{k-1}^+F_{k-1}^T + Q_k\end{aligned}\tag{3.25}$$

kde operátor  $F_{k-1}$  modeluje vývoj systému mezi stavy  $\mathbf{r}_{k-1}$  a  $\mathbf{r}_k$ ,  $F_{k-1}^T$  je transponovaná matice  $F_{k-1}$ ,  $Q_k$  je kovarianční matice procesního šumu, kterou metoda Kalmanova Filtru předpokládá, že předem známe. Zatímco operátor  $F_{k-1}$  modeluje deterministické změny uvnitř systému, přidáním kovarianční matice procesního šumu  $Q_k$  zohledňujeme stochastické procesy, které popsat v pohybovém modelu je buď nemožné, nebo zbytečně složité.

Zatímco  $\mathbf{r}_k^-$  a  $C_k^-$ , které vypočítáváme v rámci predikčního kroku na základě minulých měření  $\mathbf{m}_i$ ,  $i = 1 \dots k-1$ , jsou predikce hodnot  $\mathbf{r}_k$  a  $C_k$ ,  $\mathbf{r}_{k-1}^+$  a  $C_{k-1}^+$  jsou již opravené (resp. filtrované) predikce z předchozí iterace Kalmanova Filtru. Vypočítáváme je v rámci filtračního kroku pomocí hodnoty nejnovějšího měření  $m_k$  (viz. níže).

V případě, že jsme dosud neprovedli žádné měření, propagujeme původní odhad stavu systému  $\mathbf{r}_0^+$  a jeho kovarianční matici  $C_0^+$ , které jsme určili během inicializačního kroku:

$$\begin{aligned}\mathbf{r}_1^- &= F_0\mathbf{r}_0^+ \\ C_1^- &= F_0C_0^+F_0^T + Q_1\end{aligned}\tag{3.26}$$

**Filtrace** V závěrečném kroku Kalmanova filtru upravíme odhad hodnoty  $\mathbf{r}_{k-}^-$  (a její kovarianční matice  $C_{k-1}^-$ ) pomocí nejnovějšího měření  $\mathbf{m}_k$ , díky čemuž získáme nové, optimální odhady těchto veličin  $\mathbf{r}_k^+$  a  $C_k^+$ . Zároveň v tomto kroku vypočítáme  $\chi^2$  odchylku odhadu  $\mathbf{r}_k^+$  od dosud proběhlých měření  $\mathbf{m}_i$ ,  $i = 1 \dots k$ , díky čemuž dokážeme pro každou iteraci Kalmanova Filtru určit, jak dobře odpovídá odhad měřením, což je pro naše účely velmi užitečné, protože díky  $\chi_k^2$  odchylce můžeme rychle vypočítat, které zásahy v detektoru s nejvyšší pravděpodobností patří ke stejné trajektorii.

Pro zjednodušení dalších výpočtů (rovnice 3.31 a 3.32) zavedeme nejprve pomocný člen – takzvané **reziduum** (resp. reziduál, z anglického *residual*, v praxi používáno nejčastěji), korekční člen, který vyjadřuje rozdíl mezi odhadem Kalmanova Filtru  $\mathbf{r}_k^-$  a vlastním měřením  $\mathbf{m}_k$ .

$$\boldsymbol{\zeta}_k = \mathbf{m}_k - H_k\mathbf{r}_k^-\tag{3.27}$$

Abychom mohli aktualizovat odhad stavu  $\mathbf{r}_k^+$ , potřebujeme spočítat takzvanou *matici Kalmanova zesílení*<sup>7</sup>  $K_k$ :

$$K_k = C_k^- H_k^T (V_k + H_k C_k^- H_k^T)^{-1}\tag{3.28}$$

kde  $C_k^-$  je v predikčním kroku předpokládaná hodnota kovarianční matice,  $V_k$  je kovarianční matice chyby měření a  $H_k$  je model měření. Rovnici Kalmanova zesílení můžeme ještě drobně zjednodušit tak, že zavedeme takzvanou *váhovou matici*  $W_k$ :

$$W_k = (V_k + H_k C_k^- H_k^T)^{-1}\tag{3.29}$$

Rovnice Kalmanova zrychlení pak bude mít následující tvar:

$$K_k = C_k^- H_k^T W_k\tag{3.30}$$

---

<sup>7</sup>z angl. Gain Matrix

Jakmile máme Kalmanovo zesílení spočítané, můžeme aktualizovat odhad stavového vektoru  $\mathbf{r}_k^+$  (a jeho kovarianční matice):

$$\begin{aligned}\mathbf{r}_k^+ &= \mathbf{r}_k^- + K_k \boldsymbol{\zeta}_k \\ C_k^+ &= (\mathbf{I} - K_k H_k) \cdot C_k^-\end{aligned}\tag{3.31}$$

kde  $\mathbf{I}$  je jednotková matice a  $\mathbf{m}_k$  je poslední měření systému. V závěrečné fázi spočítáme novou odchylku  $\chi_k^2$  na základě posledního měření  $\mathbf{m}_k$ :

$$\chi_k^2 = \chi_{k-1}^2 + \boldsymbol{\zeta}_k^T W_k \boldsymbol{\zeta}_k\tag{3.32}$$

V tuto chvíli získaná hodnota  $\mathbf{r}_k^+$  s kovarianční maticí  $C_k^+$  je optimální odhad současného stavu systému. Tato dvojice je zároveň výstup dokončené iterace a vstup následující iterace Kalmanova Filtru. Hodnota  $\chi_k^2$  nám později pomocí celulárního automatu pomůže ke hledání zásahů, které s nejvyšší pravděpodobností patří ke trajektorii stejné částice.

## Kalmanův filtr v experimentu CBM

Následující podsekcce byla převzata z disertační práce Dr. Akishiny [1]. Výsledkem hledání trajektorií (fáze 1: Track finding, viz. kapitola 3) by měly být sady měření detektoru uskupené do souborů zásahů, které by v ideálním případě byly všechny vytvořené stejnou částicí. Úkolem fáze 2 (Track fitting) rekonstrukce trajektorií je odhadnout parametry trajektorií a jejich chyby, aby bylo možné získat kinematické vlastnosti částic, které nám později umožní rekonstrukci částic s krátkou *střední dobou života* a nakonec i analýzu fyzikálních procesů, které v detektoru proběhly.

Algoritmus ve fázi 2 využívá tzv. *modelu trajektorií*, což je teoretický předpoklad o pohybových rovnicích pro nabitě částice v rámci objemu detektoru. Na pohyb částic v detektoru má vliv vícero jevů, které nelze v rámci modelu trajektorií jednoduše zohlednit, zejména se jedná o:

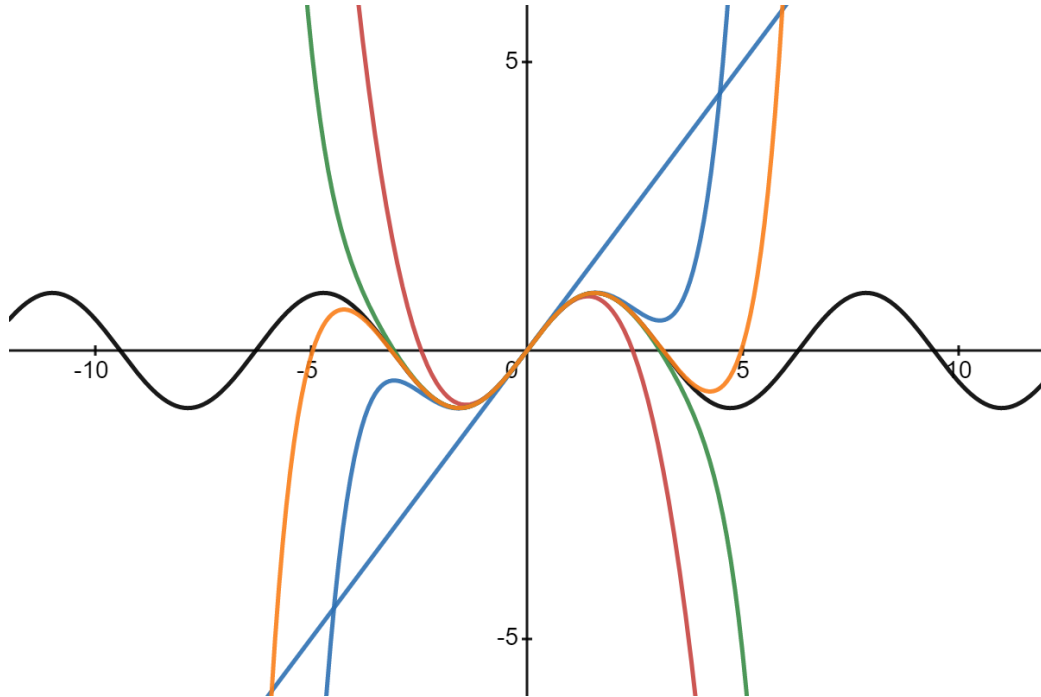
- mnohonásobný rozptyl,
- ionizace částic a
- ztráty energie způsobené radiací.

Tyto vlivy zavádějí do rekonstrukce kinematických vlastností částic perturbace a celý proces tak narušují. Použití správné rekonstrukční metody nám umožňuje dopad těchto rušivých jevů na měření minimalizovat. V dnešní době se ve většině moderních HEP experimentů používá pro odhad parametrů trajektorií částic Kalmanův Filtr.

I když se parametry trajektorií dají z jednotlivých měření odvodit pomocí *metody nejmenších čtverců*, v praxi upřednostňujeme Kalmanův filtr z důvodu, že díky své rekurzivní povaze je výpočetně mnohem výhodnější. I když obě metody počítají nad maticemi, zatímco mocnost matice u metody nejmenších čtverců závisí na počtu měření, u Kalmanova filtru na počtu stupňů volnosti systému.

## Nedostatky konvenčního Kalmanova Filtru

Konvenční Kalmanův Filtr je určený pro lineární dynamické systémy s normálními rozděleními pravděpodobnosti, což znamená, že vývoj systému mezi jednotlivými měřeními



Obrázek 3.3: Funkce  $y = \sin(x)$  a její Taylorovy rozvoje prvního, třetího, pátého, sedmého a devátého stupně. Každý vyšší stupeň Taylorova rozvoje aproximuje původní funkci přesněji. Vytvořeno pomocí nástroje Desmos [9].

$\mathbf{m}_{k-1}$  a  $\mathbf{m}_k$  dokážeme popsat pomocí **lineární** rovnice (viz rovnice 3.20). V případě, že systém pracuje s lineárními modely a pouze s normálními rozloženými pravděpodobnostmi, Kalmanův filtr je **optimální metoda aproximace** stavu takového systému.

Bohužel, v praxi se často setkáváme se situacemi, kdy model závislostí mezi jednotlivými měřeními nelze popsat pomocí lineární funkce, případně stochastické jevy systému nelze modelovat jako normální rozdělení pravděpodobnosti. V případě nelineárních modelů (případ užití, který platí i pro aproximaci trajektorií částic v detektoru CBM) je potřeba nejprve najít takový lineární model, který bude co nejlépe aproximovat chování původního, nelineárního modelu. K nalezení takového modelu nám pomůže speciální případ aproximace pomocí **Taylorova rozvoje**, kterému říkáme **linearizace**.

### Taylorův rozvoj

Mějme nepolynomiální funkci  $f : \mathbb{R} \rightarrow \mathbb{R}$ , (například  $f(x) = \sin(x)$ ): Pokud potřebujeme použít takovou funkci a z jakéhokoli důvodu je pro nás nepraktické vypočítávat její výstup přesně (například pokud potřebujeme vypočítat mnoho hodnot pro vzájemně blízká  $x$  a pokaždé vypočítávat přesnou hodnotu je příliš výpočetně náročné), můžeme funkci vhodně *aproximovat* a dále používat aproximovanou funkci, která je pro náš případ užití výhodná.

**Taylorova řada** je mocninná řada, pomocí které můžeme za určitých předpokladů vyjádřit nepolynomiální funkci v okolí bodu  $x_{lin}$  jako polynom. Taylorovu řadu reálné, **hladké** funkce  $f(x)$  zapisujeme následujícím způsobem:

$$f(x) = f(x_{lin}) + \frac{f'(x_{lin})}{1!}(x - x_{lin}) + \frac{f''(x_{lin})}{2!}(x - x_{lin})^2 + \dots = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_{lin})}{k!}(x - x_{lin})^k \quad (3.33)$$

kde  $f^{(n)}$  značí  $n$ -tou derivaci funkce  $f$ . Funkce je **hladká** tehdy, kdy nejenom funkce samotná, ale i všechny její derivace jsou spojité.

Uvedme jako příklad funkci  $f(x) = \sin x$ , kterou chceme aproximovat Taylorovým polynomem v bodě 0 (v případě Taylorovy řady v bodě 0 mluvíme o tzv. *Maclaurinově řadě*):

$$\begin{aligned} T_1^{f,0} &= f(0) + \frac{f'(0)}{1!}(x - 0)^1 = 0 + \cos(0)x = x \\ T_2^{f,0} &= T_1^{f,0} + \frac{f''(0)}{2!}(x - 0)^2 = x - \frac{\sin(0)}{2}x^2 = x \\ T_3^{f,0} &= T_2^{f,0} + \frac{f^{(3)}(0)}{3!}(x - 0)^3 = x - \frac{\cos(0)}{6}x^3 = x - \frac{x^3}{6} \\ T_4^{f,0} &= T_3^{f,0} + \frac{f^{(4)}(0)}{4!}(x - 0)^4 = x - \frac{x^3}{6} + \frac{\sin(0)x^4}{4!} = x - \frac{x^3}{6} \end{aligned} \quad (3.34)$$

Můžeme si všimnout, že sudé stupně Maclaurinovy řady se u funkce  $f(x) = \sin x$  vždy rovnají nule, díky čemuž víme, že nám pro její aproximaci stačí liché stupně rozvoje:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \quad (3.35)$$

pro  $x \in (-\infty, \infty)$ . Analogicky můžeme zapsat například i Maclaurinův rozvoj funkce  $f(x) = \cos x$ , kde jsou nenulové naopak sudé členy rozvoje. Jednotlivé polynomy jsou ukázány na obrázku 3.3.

**Linearizace** je speciální případ aproximace Taylorovým rozvojem, kdy aproximujeme pouze prvním stupněm Taylorova polynomu (tj. přímkou):

$$L(x)_f = f(x^{lin}) + f'(x^{lin})(x - x^{lin}) \quad (3.36)$$

Taková lineární funkce nám poskytuje vyhovující aproximaci pro  $x$  blízka  $x^{lin}$ . Mějme funkci  $f(x) = \sqrt{x}$ , a hledejme hodnotu pro  $\sqrt{4,41}$ : Víme, že  $\sqrt{4} = 2$ , a tak vhodně zvolíme  $x^{lin} = 4$ :

$$L(x) = \sqrt{4} + \frac{1}{2\sqrt{4}}(x - 4) = 1 + \frac{x}{4} \quad (3.37)$$

Linearizovaná funkce  $L(x)$  aproximuje  $\sqrt{4,41} \approx 2,1025$ , přesné řešení je přitom  $\sqrt{4,41} = 2,1$ . Linearizace nám tedy poskytuje solidní aproximaci pomocí relativně jednoduchého výpočtu. Pokud ale chceme linearizaci použít pro funkce, které pracují nad vektory, narazíme na problém, jak získat první derivaci matice. K tomuto účelu nám poslouží tzv. **Jacobiho matice**.

### Jacobiho matice

Mějme funkci  $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ . Jacobiho matice  $J$  je **matice parciálních derivací** takové funkce ve tvaru  $m \times n$ :

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad (3.38)$$

Jacobiho matice nám pomůže provádět linearizaci nelineárních modelů pro použití v rámci Kalmanova Filtru. Této verzi algoritmu pak říkáme **Rozšířený Kalmanův Filtr (EKF)**<sup>8</sup>.

## Rozšířený Kalmanův filtr

**Rozšířený Kalmanův Filtr**, jak název napovídá, je rozšíření původní metody Kalmanova Filtru, které pracuje nad nelineárními modely měření a/nebo modely vývoje systému.

Rozšířený Kalmanův Filtr sice umožňuje pracovat s nelineárními modely, z tohoto faktu ale plyne také jeho zásadní nevýhoda: Kvůli procesu linearizace není EKF optimální algoritmus pro odhad stavu systému, a při nevhodně zvolených hodnotách počátečního odhadu systému nebo linearizačního bodu  $\mathbf{r}_{k-1}^{lin}$  může metoda divergovat, je tedy velmi důležité tyto hodnoty zvolit vhodně.

V případě rozšířeného Kalmanova Filtru musíme některé rovnice pozměnit tak, aby mohly pracovat nad nelineárními systémy. V případě, že pracujeme s nelineárním modelem vývoje systému  $\mathbf{f}_k$ , potřebujeme rovnici 3.20 upravit následujícím způsobem:

$$\mathbf{r}_k^- \equiv \mathbf{f}_k(\mathbf{r}_{k-1}^+) \approx \mathbf{f}_k(\mathbf{r}_{k-1}^{lin}) + F_k(\mathbf{r}_{k-1}^+ - \mathbf{r}_{k-1}^{lin}) \quad (3.39)$$

kde  $\mathbf{f}_k$  je **nelineární** model vývoje systému,  $\mathbf{r}_{k-1}^{lin}$  je linearizační bod, typicky zvolený podle minulého stavu systému, a  $F_k$  je Jacobiho matice nelineárního modelu vývoje  $\mathbf{f}_k$ :

$$F_{k(ij)} = \left. \frac{\partial \mathbf{f}_k(\mathbf{r}_{k-1}^+)(i)}{\partial \mathbf{r}_{k-1}^+(j)} \right|_{\mathbf{r}_{k-1}^+ = \mathbf{r}_{k-1}^{lin}} \quad (3.40)$$

Podobně potřebujeme upravit rovnici vztahu mezi měřením a reálným stavem systému (rovnice 3.21) v případě nelineárního modelu měření:

$$\mathbf{m}_k(\mathbf{r}_k^t) \equiv \mathbf{h}_k(\mathbf{r}_k^t) + \boldsymbol{\eta}_k \approx \mathbf{h}_k(\mathbf{r}_k^{lin}) + H_k(\mathbf{r}_k^t - \mathbf{r}_k^{lin}) + \boldsymbol{\eta}_k \quad (3.41)$$

Kde  $H_{k(ij)}$  je Jacobiho matice modelu měření v následujícím tvaru:

$$H_{k(ij)} = \left. \frac{\partial \mathbf{h}_k(\mathbf{r}_k)(i)}{\partial \mathbf{r}_k(j)} \right|_{\mathbf{r}_{k-1}^+ = \mathbf{r}_{k-1}^{lin}} \quad (3.42)$$

V případě nelineárního modelu měření se také změní rovnice pro výpočet rezidua:

$$\boldsymbol{\zeta} = \mathbf{m}_k - (\mathbf{h}_k(\mathbf{r}_k^{lin}) + H_k(\mathbf{r}_k^- - \mathbf{r}_k^{lin})) \quad (3.43)$$

Již jsme zmínili, že rozšířený Kalmanův filtr může mít v některých případech problémy se stabilitou. Z použití rozšířeného Kalmanova filtru vyplývají některé potíže s implementací stabilního a efektivního algoritmu a jeho následnou akcelerací na různých hardwarových platformách. Některými možnými řešeními se budeme dále zabývat v kapitole 4.

<sup>8</sup>EKF – Extended Kalman Filter (rozšířený Kalmanův Filtr)

### 3.3 Metody rekonstrukce trajektorií

Časově nejnáročnější část rekonstrukce trajektorií je jejich rekonstrukce, proto je potřeba vhodná volba sledovacího algoritmu. Náročnost této části analýzy událostí v detektoru vychází z faktu, že v této fázi algoritmus pracuje s hrubými, dosud nijak nezpracovanými daty a objem dat, který musí algoritmus zpracovat nelze nijak omezit. Často se tedy stává, že algoritmus musí v této fázi zpracovávat data až o objemu jednotek TB/s. Zároveň je časté, že v závislosti na konkrétním experimentu vyžadujeme, aby byla data zpracována v reálném čase, což úkol vývoje rekonstrukčního programu dále komplikuje a je tedy velmi důležité takový algoritmus vhodně zvolit. Situaci dále komplikuje fakt, že každý detektor je jiný, je tedy v případě každého detektoru potřeba výběr algoritmu přizpůsobit konkrétním vlastnostem detektoru. V následující sekci shrneme několik takových algoritmů, které můžeme pro rekonstrukci trajektorie zvolit:

- Houghova transformace,
- celulární automaty.

Právě metodou rekonstrukce trajektorií pomocí celulárních automatů se pak budeme podrobněji zabývat v kontextu experimentu CBM.

Důležitým faktorem při výběru vhodného algoritmu je otázka, jak moc je možné problém rekonstrukce trajektorií zjednodušit, tedy jaké zjednodušující předpoklady, které můžeme zavést, usnadní (popřípadě umožní zrychlení např. paralelizací algoritmů tam, kde to u obecného problému bylo nemožné nebo nepraktické) řešení daného problému, aniž by (příliš) snížil přesnost výpočtu. Pro nás je pravděpodobně nejdůležitější vědět, jestli pro rekonstrukci trajektorií daná metoda pracuje se všemi vstupními daty, nebo jestli dokáže pracovat i pouze nad nějakou jejich podmnožinou. Pokud metoda vyžaduje přístup ke všem vstupním datům, říkáme, že mluvíme o **globální** metodě, v opačném případě mluvíme o **lokální** metodě. Pro naše účely je výhodné pracovat s metodami lokálními, protože se přirozeně nabízí možnost metodu do jisté míry paralelizovat. Níže se budeme zabývat Houghovou transformací jako zástupcem globálních metod rekonstrukce částic a metodou celulárních automatů jako zástupcem lokálních metod. Metodou celulárních automatů se budeme také dále zabývat v kontextu rekonstrukce trajektorií.

#### Houghova transformace

Houghova transformace je technika extrakce prvků, původně navržena Paulem V. C. Houghem za účelem strojové analýzy fotografií bublinkových komor [20]. Ve své generalizované formě [10] je hojně používána v analýze obrazu, počítačovém vidění nebo zpracování obrazu. Cílem metody je najít nedokonale vyobrazené objekty (zpravidla objekty stejného charakteru, například přímky nebo kružnice) pomocí takzvaného **hlasování**.

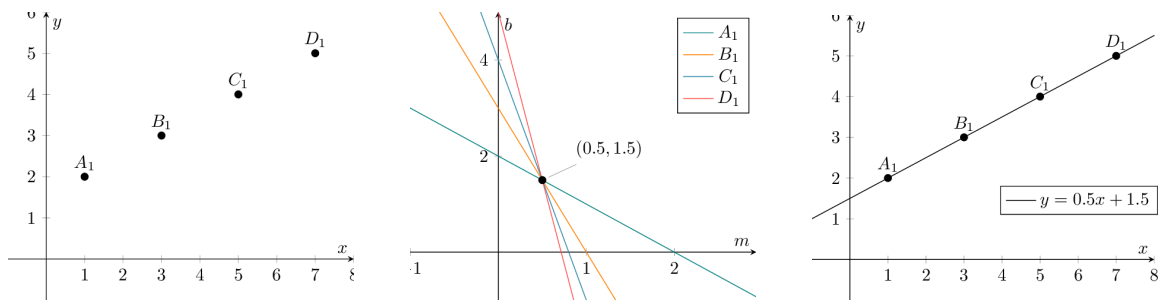
Mějme nejjednodušší verzi Houghovy transformace, a to takovou, která detekuje v obraze přímky: Přímku definujeme jednoduše pomocí rovnice

$$y = mx + b$$

v důsledku čehož můžeme přímku zapsat jako uspořádanou dvojici parametrů  $(m, b)$ , což je informace, která jednoznačně definuje danou přímku jako bod  $(m, b)$  v takzvaném **parametrovém prostoru**<sup>9</sup>.

---

<sup>9</sup>z angl. parameter space



Obrázek 3.4: Příklad Houghovy transformace. Body v **reálném prostoru** (obrázek vlevo) jsou převedeny do **parametrového prostoru** jako přímky. Pokud se všechny body nachází na jedné přímce v reálném prostoru, budou mít jim odpovídající přímky v parametrovém prostoru právě jeden průsečík (obrázek uprostřed). Průsečík v parametrovém prostoru obsahuje parametry přímky, na které se všechny tyto body nachází (obrázek vpravo). Obrázky převzaty z [27].

Uvedme jako příklad hledání přímek pomocí Houghovy transformace následující situaci, znázorněnou na obrázku 3.4: mějme přímku v reálném prostoru, definovanou funkcí  $y = 0.5x + 1.5$ , o které máme informaci v podobě čtyř bodů:  $A_1 = (1, 2)$ ,  $B_1 = (3, 3)$ ,  $C_1 = (5, 4)$ ,  $D_1 = (7, 5)$ . Tak, jako můžeme zapsat jakoukoli přímku jako bod v parametrovém prostoru, můžeme obdobným způsobem pracovat s jednotlivými body. Každý bod z reálného prostoru se promítne do parametrového prostoru jako přímka, která obsahuje body s parametry všech přímek, které v reálném prostoru procházejí daným bodem. Všechny čtyři body tak vyneseme jako přímky v parametrovém prostoru a pozorujeme jejich podobu: Z povahy Houghovy transformace vyplývá, že všechny přímky v parametrovém prostoru, které mají v parametrovém prostoru společný průsečík, leží v reálném prostoru na stejné přímce. Průsečík těchto přímek v parametrovém prostoru nám zároveň sděluje, jaké jsou parametry takové přímky, v případě příkladu na obrázku 3.4 bodu  $(0.5, 1.5)$  v parametrovém prostoru odpovídá přímka  $y = 0.5x + 1.5$ , která prochází všemi body  $A_1$  až  $D_1$ .

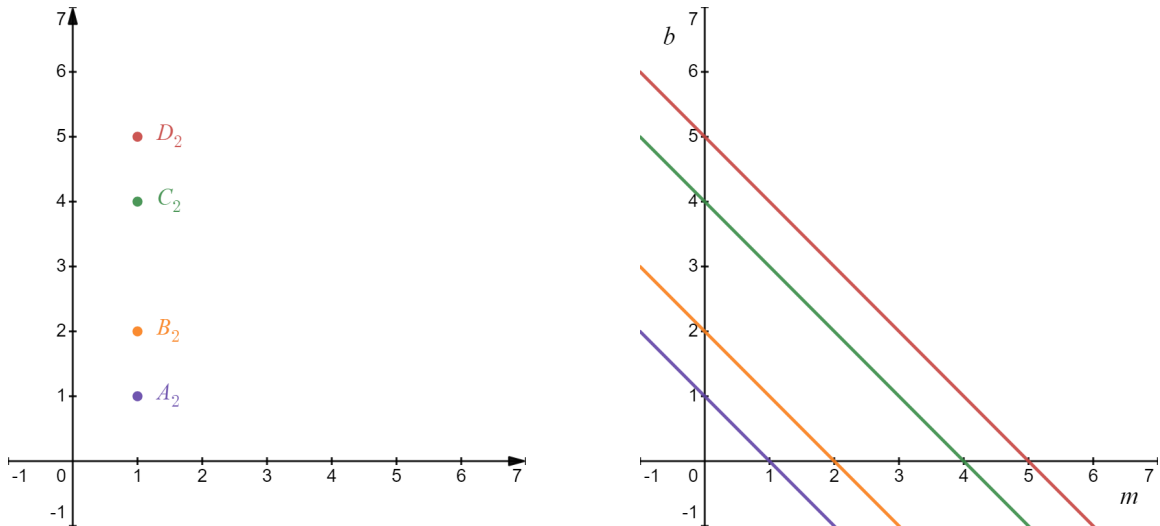
Tento přístup má ale nevýhodu v případě, že chceme takto zapsat svislou přímku, což by mělo za následek parametr  $m$  jdoucí k nekonečnu. Když se pokusíme převést do parametrového prostoru body, které jsou přímo nad sebou (tj. mají stejnou  $x$  souřadnici, viz obrázek 3.5), zjistíme, že jsou všechny odpovídající přímky rovnoběžné a přestože lze všemi body vést jednu přímku, v parametrovém prostoru žádný průsečík nenajdeme.

Tento nedostatek Houghovy transformace můžeme vyřešit tak, že změním způsob, jakým přímku popisujeme: Jedním z řešení je použít pro popis přímky takzvanou **Hesseho normální formu**: Prvním parametrem  $\rho$  udáváme vzdálenost přímky od počátku souřadného systému (jinými slovy délku úsečky, která je ohraničena počátkem souřadného systému a bodem na popisované přímce a která svírá s popisovanou přímku pravý úhel), druhým parametrem  $\theta$  pak popisujeme úhel mezi osou  $x$  a úsečkou, spojující počátek souřadného systému s nejbližším bodem popisované přímky [10]:

$$\rho = x \cos \theta + y \sin \theta \tag{3.44}$$

Uvedme opět příklad se svislou přímku z obrázku 3.5: Vzhledem k tomu, že používáme jinou transformaci, body se již nebudou do parametrového prostoru promítat jako přímky, nýbrž jako sinusovky. V bodě, kde se všechny sinusovky protnou, můžeme analogicky jako





Obrázek 3.5: Příklad nedostatku Houghovy transformace. Obrázek vpravo zobrazuje body na jedné svislé přímce v reálném prostoru, obrázek vpravo zobrazuje tyto body v parametrovém prostoru. Přestože body náležejí jedné přímce, v parametrovém prostoru nemají jednotlivé přímky průsečík. Vytvořeno pomocí nástroje Desmos [9].

u předchozích příkladů vyčíst parametry přímky, na které se body nachází (viz. obrázek 3.7).

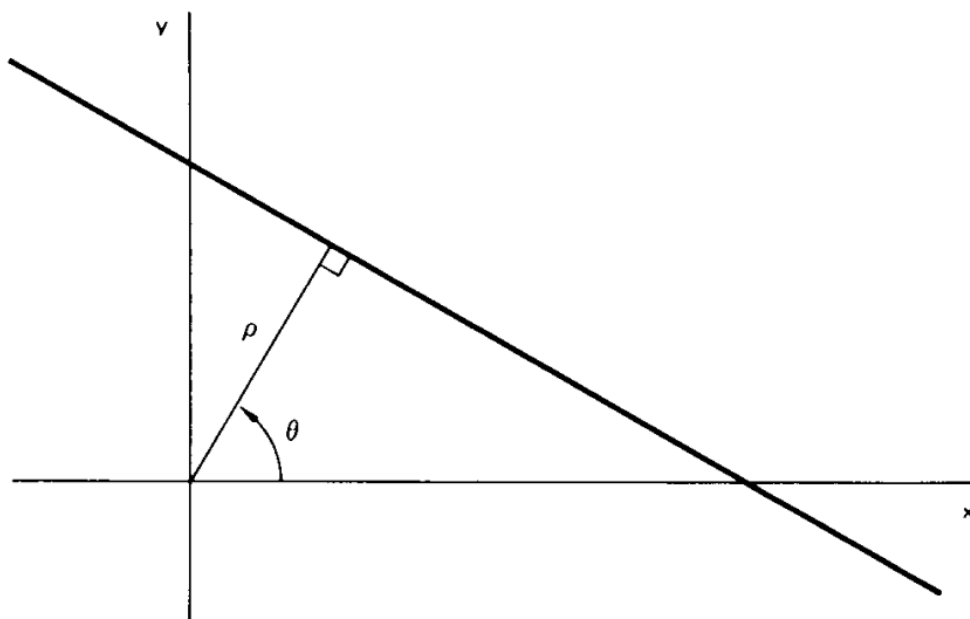
Všimněme si u transformace v normální formě, že se průsečíky jednotlivých sinusovek periodicky opakují: Tento jev pozorujeme z toho důvodu, že přímka se pomocí Hesseho normální formy dá vyjádřit vícero způsoby, což vyplývá z faktu, že úsečka  $\rho$  může s osou  $x$  svírat úhly  $\theta + 2k\pi$  radiánů (první a třetí průsečík v obrázku 3.7), popřípadě  $\theta + (2k + 1)\pi$  radiánů (druhý průsečík, v takovém případě nám ale délka úsečky  $\rho$  vyjde „záporná“).

Generalizovaná forma Houghovy transformace, jak píše Duda a kolektiv [10], může kromě přímek v obraze hledat jakýkoli geometrický útvar, který lze popsat dvěma parametry: pro nás může být relevantní například hledání kružnic: mějme například k dispozici čtyři body, které náležejí jedné kružnici:  $A_4 = (1, 2)$ ,  $B_4 = (2, 1)$ ,  $C_4 = (2, 3)$ ,  $D_4 = (3, 2)$ . Rovnice pro popis kružnice má následující tvar:

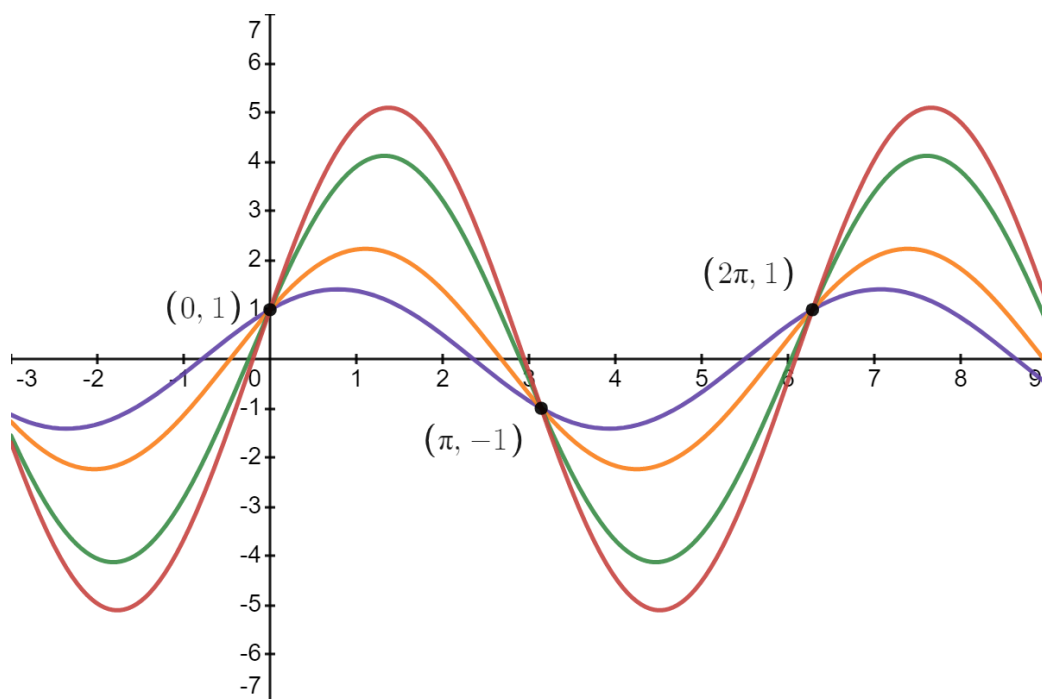
$$r^2 = (x - h)^2 + (y - k)^2 \quad (3.45)$$

kde  $r$  značí poloměr kružnice a bod  $(h, k)$  značí její střed. V tomto případě přichází drobná komplikace, protože kružnici definujeme pomocí tří parametrů. Tento problém obvykle vyřešíme tak, že jeden z hledaných parametrů známe. Pro náš příklad řekněme, že víme, že kružnice má poloměr  $r = 1$ : parametrový prostor tak zůstává dvojrozměrný a můžeme tak promítnout jednotlivé body do parametrového prostoru jako kružnice. V bodě, kde se všechny kružnice v parametrovém prostoru protínají, najdeme parametry pro kružnici v reálném prostoru (konkrétně pak polohu středu kružnice).

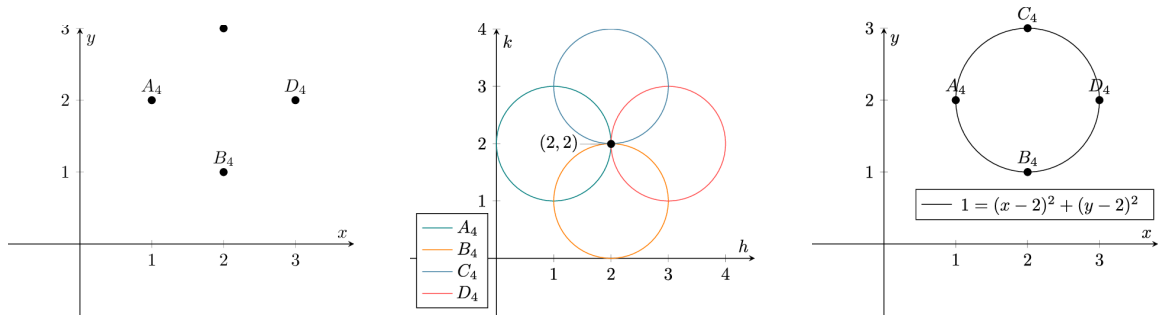
Houghovu transformaci můžeme využít pro rekonstrukci trajektorií následujícím způsobem: V parametrovém prostoru vytvoříme histogram, kde zaznamenáváme, kolik obrazů bodů v parametrovém prostoru prochází každým bodem. Poté postupujeme globálně po celém prostoru a hledáme body s největším počtem procházejících přímek. Z takových bodů postupně vyčítáme parametry pravděpodobných trajektorií a body v reálném prostoru, které se do daného bodu v parametrovém prostoru promítly, vyjme a proces opaku-



Obrázek 3.6: Vizualizace parametrů Houghovy transformace v Hesseho normální formě. Délka úsečky  $\rho$  značí nejkratší vzdálenost přímky od počátku souřadnicového systému,  $\theta$  je úhel mezi úsečkou  $\rho$  a osou  $x$  (viz rovnice 3.44). Převzato z [10].



Obrázek 3.7: Parametrový prostor Houghovy transformace bodů z obrázku 3.5. Vytvořeno pomocí nástroje Desmos [9].



Obrázek 3.8: Verze generalizované Houghovy transformace, hledající kružnice. Body, náležící jedné kružnici (obrázek vlevo) promítneme do parametrového prostoru (obrázek uprostřed). Jejich průsečík udává střed hledané kružnice (vpravo). Převzato z [27].

jeme. Tímto způsobem zároveň potlačujeme šum, protože šumem vzniklé „falešné“ zásahy zpravidla nebudou tvořit v parametrovém prostoru průsečíky.

Tato metoda má pro naše účely jednu zásadní nevýhodu: Houghova transformace pracuje nad celým stavovým prostorem, což v případě sledování částic u CBM znamená značnou paměťovou náročnost a výrazné ztížení paralelizace a akcelerace na hardwarových platformách. Z tohoto důvodu je použití Houghovy transformace sice možné pro účely akcelerace výpočtu ale nepraktické.

## Celulární automat

**Celulární automat** (CA<sup>10</sup>) je souhrnné označení pro typicky diskrétní výpočetní model, studovaný v rámci **teorie automatů**. Jedná se o překvapivě multifunkční jednoduchý nástroj, hojně používaný pro simulace v oblastech dopravy, epidemiologie, fyziky a mnoho dalších.

Celulární automat můžeme zpravidla definovat pomocí těchto čtyř kritérií:

- **pole buněk**, které určuje rozměr celulárního automatu. Pole buněk může mít libovolný počet rozměrů, obvykle jeden nebo dva. Dále rozlišujeme, jestli je pole nekonečné, nebo ohraničené (v případě, že je ohraničené, rozlišujeme, jak jsou definovány stavy hranic pole). V neposlední řadě určujeme tvar buněk v poli (například čtvercové, šestiúhelníkové, krychlové u 3D atd.), což dále ovlivňuje tvar okolí každé buňky,
- **Množina stavů buněk (S)**, automat může počítat s libovolným **konečným** počtem možných stavů, kterých mohou jednotlivé buňky nabývat (v nejjednodušším případě například dvojice stavů 0,1),
- **okolí buněk** určuje, které buňky (a jejich stavy) bude každá buňka brát v potaz, když bude rozhodovat o svém příštím stavu. Rozlišujeme například Mooreovo okolí (čtvercové okolí 8 buněk, majících s buňkou společnou stranu nebo vrchol), rozšířené Mooreovo okolí (stejně, jako Mooreovo, rozšířené o Mooreovo okolí všech osmi sousedů, dohromady tedy 24 buněk v okolí), nebo například Von-Neumannovo okolí (pouze čtyři buňky, které mají společnou stranu s buňkou patří do jejího okolí),

<sup>10</sup>CA – Cellular Automaton (Buněčný automat)

- **přechodová funkce:** pravidla, která určují, jak na základě stavu a okolí buňky vypočítat příští stav buňky. Všechny změny stavu probíhají zároveň na základě posledních stavů pole buněk.

Nejznámější příklad celulárního automatu je hra „Život“ Johna Conweye. Následující odstavec čerpá ze článku Martina Gardnera a skript Jarkka Kari [15, 22]: Hra „Život“ je celulární automat s nekonečným 2D polem buněk o dvou možných stavech (0 – „mrtvá“ a 1 – „živá“) a jednoduchým Mooreovým okolím. Pravidla pro hru „Život“ byla Conwayem navržena tak, aby splňovala následující 3 požadavky:

1. Neměla by existovat žádná počáteční konfigurace, pro kterou by existoval jednoduchý důkaz, že populace (počet „živých“ buněk, nesoucích hodnotu 1) může neomezeně růst.
2. Měly by existovat počáteční konfigurace, které *zřejmě* způsobují neomezený růst populace.
3. Měly by existovat počáteční konfigurace, které rostou a vyvíjejí se po delší dobu, než se doberou k jednomu ze tří možných konců: úplně „vymřou“ (tj. v určitý čas nezbude žádná „živá“ buňka, ať už kvůli příliš vysoké, nebo příliš nízké hustoty „zalidnění“), ustálí se ve stabilním rozložení, které se dále již nezmění, nebo se ustálí v oscilující konfiguraci, během které se s určitou periodou donekonečna opakuje konečné množství konfigurací.

Tyto požadavky mají zařídít, že se samotný celulární automat bude chovat **nepředvídatelně**. Conway tedy navrhl pravidla vývoje populace, která Martin Gardner popsal jako „rozkošně jednoduchá“ [15].

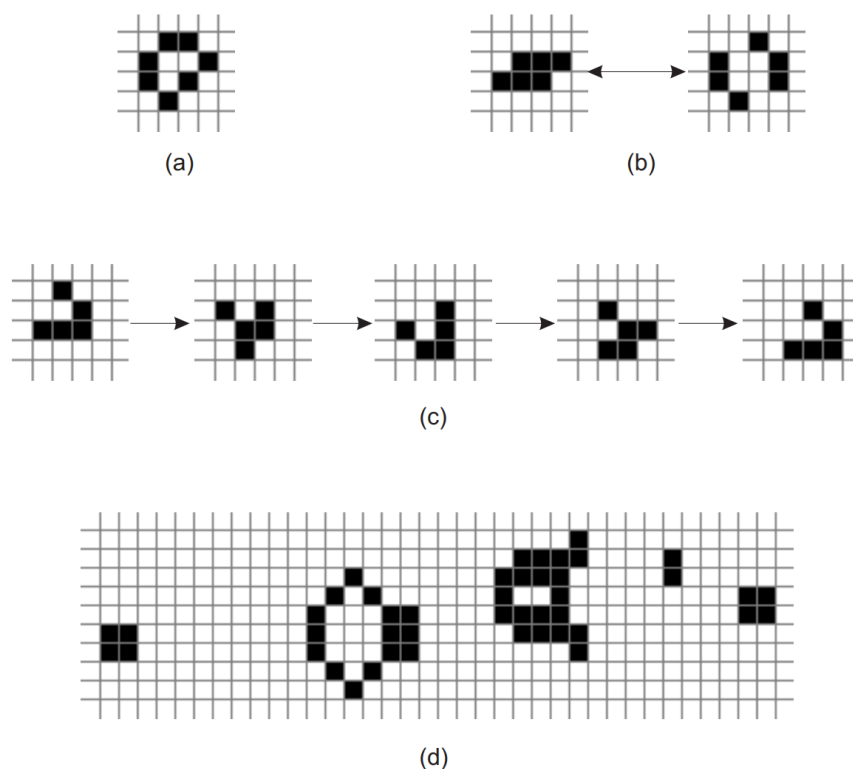
Conwayova hra „Život“ je příklad tzv. **totalistického celulárního automatu**, což znamená, že nezáleží na konkrétní poloze „živých“ a „mrtvých“ buněk v rámci okolí buňky, což snižuje počet pravidel ze  $2^9 = 512$  (počet možných konfigurací Mooreova okolí krát dva, protože pravidla jsou různá pro „živou“ a „mrtvou“ buňku) na  $2 \times 9 = 18$  (výběr pravidla je závislý pouze na stavu buňky a počtu „živých“ buněk).

Pravidla celulárního automatu tak můžeme definovat následovně:

1. Přežití: Pokud je buňka živá a má právě dva nebo tři živé sousedy, přežívá do další evoluce
2. Smrt: Každá živá buňka, která má čtyři a více živé sousedy, zemře kvůli přelidnění. Taktéž každá živá buňka, která má pouze jednoho nebo žádného živého souseda, zahyne samotou.
3. Zrození: Každá „neživá“ buňka, která má **právě** tři živé sousedy, se „narodí“ a v příští evoluci algoritmu bude živá.

V průběhu let hra Život přilákala velké množství nadšenců, kteří postupně shromáždili rozměrnou knihovnu vzorů s různými chováními, které lze kategorizovat pomocí mezi nadšenci uznávané terminologie [22]:

- *still life* (česky *stabilní život*): stabilní vzor, ve kterém se nerodí žádné nové buňky a žádné naopak neumírají, vzor tedy v této konfiguraci, pokud není zvenčí narušen zůstane donekonečna. Nejjednodušší představitel tohoto vzoru je **blok**, čtyři živé buňky, poskládané do  $2 \times 2$  čtverce. Další příklad je ukázán na obrázku 3.9(a).



Obrázek 3.9: Vybrané objekty Conwayovy hry Život: stabilní život (a), oscilátor (b), kluzák (c), kluzákové dělo (d). Převzato z [22].

- *oscilátor*: Periodická konfigurace. Konfigurace se může mezi jednotlivými generacemi měnit, avšak po  $n$  generacích se vrátí do původní konfigurace, říkáme tedy, že vzor má periodu  $n$ . Můžeme také tvrdit, že stabilní život je oscilátor s periodou 1. Nejmenší příklad oscilátoru je takzvaná *blíkačka* (anglicky *blinker*), která se skládá ze tří sousedících buněk v jedné přímce. S každou další evolucí se změní ze svislé na vodorovnou a naopak. Další příklad oscilátoru (takzvaná *žába*) je vyobrazen na obrázku 3.9(b).
- *vesmírné lodě*: Konfigurace, která se s určitou periodou objeví znovu, oproti oscilátorům se liší v tom, že se znovu objeví na jiném místě v poli, působí tak, že „cestuje“. Na obrázku 3.9(c) vidíme nejjednodušší vesmírnou loď, takzvaný *kluzák*.
- *dělo*: oscilující konfigurace, která se stejně jako oscilátor s určitou periodou vrátí do původní podoby, na rozdíl od oscilátoru ale během této periody vygeneruje vesmírnou loď. Na obrázku 3.9(d) můžeme vidět takzvané *kluzákové dělo*.

Navzdory své jednoduchosti vykazuje hra Život natolik sofistikovanou vlastnost jako *sebeorganizace*: I když celulární automat inicializujeme v chaotické počáteční konfiguraci, s postupem času začne metoda vykazovat organizovanou strukturu v podobě (zpravidla) oscilátorů a stabilních vzorů. Tuto vlastnost můžeme využít pro rekonstrukci částic, zejména pro potlačení šumu. Můžeme se na hru Život (a celulární automaty obecně) dívat jako na

lokální verzi **rekurentních neuronových sítí (RNN)**<sup>11</sup>. Navíc, díky jejich lokální povaze, jsou celulární automaty vhodné pro paralelní přístup, kdy v extrémním případě můžeme na každou buňku nahlížet jako na samostatné vlákno programu [1].

Celulární automaty jsou v dnešní době hojně využívány pro rekonstrukci trajektorií v mnoha HEP experimentech, mezi nimi ALICE<sup>12</sup>, HERA-B<sup>13</sup>, K2K<sup>14</sup>, LHCb<sup>15</sup>, NEMO<sup>16</sup>, STAR<sup>17</sup> nebo ARES<sup>18</sup>, který byl prvním experimentem, který používá celulární automaty pro rekonstrukci trajektorií a konfigurace automatu je velmi podobná Conwayově původní hře Život. Každý z experimentů má jiné požadavky na algoritmus rekonstrukce trajektorií, tak i jednotlivé celulární automaty jsou rozdílné. My se budeme dále zabývat celulárním automatem použitým v experimentu CBM.

## Celulární automat pro experiment CBM

Následující podsekcce čerpá z práce Dr. Akishiny [1]. Celulární automat na podobné bázi jako Conwayova hra Život sice funguje dobře pro relativně nízké množství trajektorií, jako je experiment ARES, pro detektory s výrazně vyšším počtem trajektorií jako je CBM budeme ale nuceni zavést velké změny do tohoto algoritmu. V první řadě je potřeba výrazně změnit pohled na pojem buňky automatu: Buňka by měla být schopna vyjadřovat co největší množství parametrů možné trajektorie. Tam, kde detektor experimentu ARES poskytoval nejen informace o pozici, ale i směru zásahu, bude strategie rekonstrukce výrazně odlišná od experimentů, používajících pixelové nebo proužkové detektory (jako v případě CBM), kde máme k dispozici pouze informace o polohách zásahů. Protože neobsahuje informace o směru trajektorie, nenese jeden zásah v hladině detektoru dostatečně velké množství informací, aby mohl popsat trajektorii částice, která ho způsobila. Na druhou stranu, povědomí o parametrech trajektorie můžeme zlepšit, když v rámci jedné buňky automatu zkombinujeme data z vícero po sobě jdoucích zásahů.

Jak je ukázáno na obrázku 3.10, v případě experimentů s vysokou hustotou trajektorií definujeme buňku automatu odlišně od intuitivního přístupu: Zde je každá buňka potenciální segment trajektorie, skládající se ze dvou sousedních zásahů v detektoru. Vstupní informace algoritmu jsou vyčtené zásahy v jednotlivých detektorových hladinách: Zeleně vybarvené značky reprezentují zásahy způsobené jednou částicí, modré značky způsobila částice jiná. Bílá značka značí falešný zásah způsobený šumem. V první fázi algoritmus spojí všechny zásahy na sousedních detektorových hladinách a takto vytvořené segmenty projde a zahodí takové kombinace, které by nikdy nemohly patřit reálné částici. Od této chvíle už algoritmus nikdy nebude pracovat s jednotlivými zásahy, ale vždy s buňkami v podobě dvojic zásahů na sousedních detektorových hladinách.

Jakmile jsou „buňky“ vytvořeny, může začít evoluce celulárního automatu: Metoda hledá pro každou buňku živé sousedy (sousedy v tomto případě definujeme tak, že mají s buňkou společný zásah), kteří mají přibližně stejný směr, přičemž bere v potaz fyzikální model částic (zde nám pomůže výše zmíněný Kalmanův Filtr), protože takoví sousedé mají vyšší pravděpodobnost, že patří ke stejné trajektorii. Zároveň je každé buňce postupně

---

<sup>11</sup>RNN – Recurrent Neural Network (rekurentní neuronová síť)

<sup>12</sup>ALICE – A Large Ion Collider Experiment

<sup>13</sup>HERA – Hadron-Elektron-Ring Anlage (Hadron-Elektronový prstencový urychlovač)

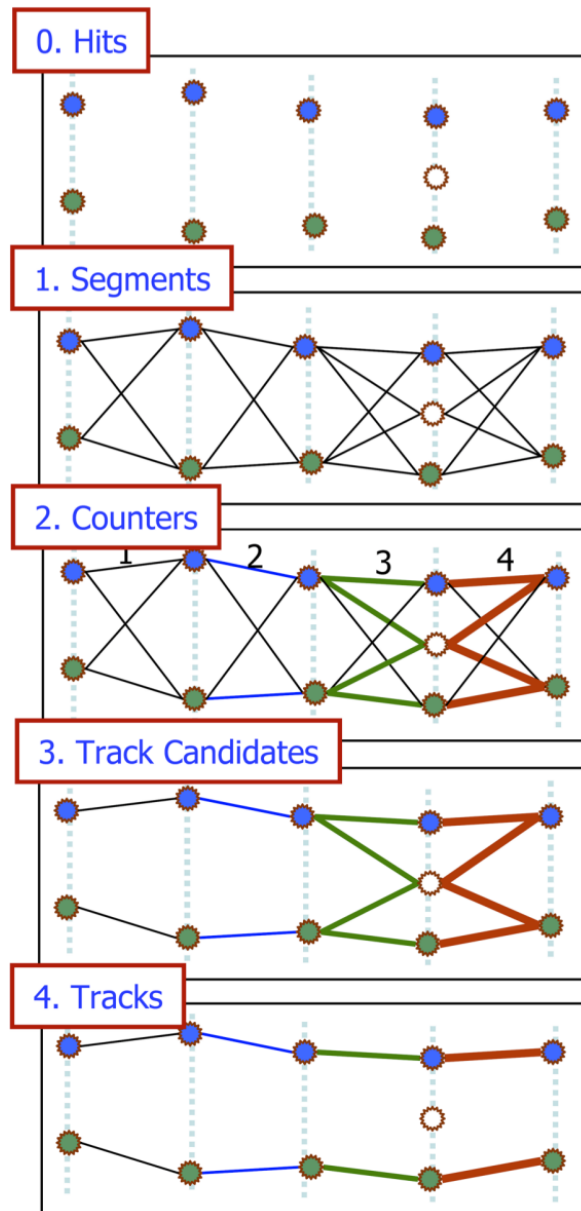
<sup>14</sup>K2K – KEK to Kamioka

<sup>15</sup>LHCb – Large Hadron Collider beauty experiment

<sup>16</sup>NEMO – Neutrino Ettore Majorana Observatory

<sup>17</sup>STAR – Solenoidal Tracker

<sup>18</sup>ARES – Automotive Research Experiment Station



Obrázek 3.10: Zjednodušená ilustrace celulárního automatu, používaného například v experimentu CBM pro rekonstrukci trajektorií částic. Svislé přerušované čáry představují jednotlivé detektorové hladiny, zelené a modré značky značí zásahy způsobené dvěma různými částicemi, bílá značka je falešný zásah způsobený šumem. Segmenty trajektorií jsou zaznamenány jako černé, plné úsečky. Přejato z [1].

přiřazeno číslo, které reprezentuje počet sousedů vlevo (čímž ohodnocujeme délku tohoto kandidáta na trajektorii, čím delší, tím lepší). Tímto způsobem získáváme stromovou strukturu kandidátů na trajektorie. V tomto okamžiku dochází k soutěži jednotlivých kandidátů: Přežít mohou pouze ti nejdelší kandidáti, u kterých Kalmanův Filtr spočítal nejlepší hodnotu  $\chi^2$  a kteří nesdílí žádné společné zásahy s lepšími kandidáty [1].



## Kapitola 4

# Návrh a implementace simulátorů částic v detektoru CBM

Tato kapitola se zabývá podrobným rozбором vlastního algoritmu generování trajektorií částic. Podrobně se tato kapitola bude zabývat návrhem, implementací, verifikací a optimalizací dvou simulátorů drah částic za účelem získání testovacích dat pro účely návrhu a verifikace výsledného algoritmu. Zvláštní pozornost je pak věnována demonstraci vlastností jednotlivých simulačních modelů a porovnání jejich praktičtosti pro generování částic tak, aby vygenerované částice mohly být použity pro verifikaci a měření výkonu sledovacího algoritmu (a jeho součástí). Závěrem se tato kapitola také zabývá možnými problémy jednotlivých modelů, které generování vhodných částic komplikují.

Algoritmus sledování částic, diskutovaný v této práci, využívá do značné míry technologie Kalmanova filtru (respektive jeho rozšířené varianty, **EKF**). Implementace vhodného Kalmanova filtru je komplikovaný proces s mnoha faktory, které mohou drasticky ovlivnit kvalitu jeho výsledku, je proto velmi důležité správně porozumět jeho fungování. Z tohoto důvodu tato práce uvádí za účelem lepšího porozumění algoritmu příklad, který by měl napomoci snazšímu pochopení funkce, kterou Kalmanův filtr vykonává při rekonstrukci trajektorií částic v detektoru.

### 4.1 Simulátor částic v detektoru

Simulační model, který byl použit pro generování trajektorií částic v detektoru, je založen na Lorentzových zákonech pohybu částic v elektromagnetickém poli. Problém generování trajektorií částic v detektoru je složitější, než v případě jiných simulačních modelů z toho důvodu, že (zvláště v případě nehomogenního magnetického a elektrického pole) takový model je nepraktické řešit analyticky. Z tohoto důvodu je výhodnější generovat trajektorie částic pomocí spojitě (respektive kombinovaně) simulace.

Pro účely generování testovacích dat se lze dopustit některých zjednodušení, která výrazně usnadní jak vlastní generování částic, tak vlastní práci s daty. Konkrétně zjednodušíme fyzikální model následovně:

- **homogenní magnetické pole:** Simulátor si klade za cíl modelovat pouze trajektorie v rámci prvního stanoviště detektoru (STS), který se celý nachází uprostřed velkého supravodivého magnetu. Pro experiment CBM se v současné době používá 3D mapa nehomogenního magnetického pole o velikosti téměř 90MB. Pro naše účely je ale taková implementace zbytečně složitá a vzhledem k tomu, že jsou všechny

stanice detektoru STS, který využíváme pro rekonstrukci trajektorií, plně uvnitř tohoto supravodivého magnetu, můžeme (zvláště u prvních detektorových hladin, kde se přesnost magnetického pole projeví nejvíce a které jsou nejbližší středu magnetu, kde je magnetické pole „nejhomogennější“) minimálně uvnitř magnetu pro naše účely předpokládat, že magnetické pole je homogenní. V případě rekonstrukce trajektorií v praxi ale mapa nehomogenního magnetického pole pomůže k dalšímu zpřesnění modelu pro Kalmanův Filtr, díky čemuž je snazší eliminovat šum a celkově zvýšit efektivitu filtrace.

- **nedochází k interakcím mezi částicemi:** Ve chvíli, kdy částice vstoupí do detektoru, předpokládáme, že interagují pouze s detektorem, nikoli mezi sebou. Díky tomuto předpokladu si můžeme dovolit simulovat každou částici zvlášť, a v případě potřeby můžeme generování částic paralelizovat, aniž by bylo nutné jednotlivé procesy simulátoru synchronizovat.
- **všechny částice jsou primární s vysokou energií:** U reálného detektoru předpokládáme výskyt čtyř typů trajektorií, které vyplývají z vlastností a konfigurace experimentu: Částice se liší v energii, což ovlivňuje jejich zakřivení v detektoru a zároveň míru mnohonásobného rozpadu při průchodu detektorem, což výrazně ovlivňuje nejistotu systému. Dále se částice mohou lišit časem a místem vzniku a poločasem rozpadu, kdy částice mohou vzniknout v místě kolize s terčem (pak říkáme, že je částice *primární*), nebo vzniknout v důsledku rozpadu jiné částice uvnitř detektoru (takové částici říkáme sekundární). Nakonec z nedokonalosti detektoru lze předpokládat, že některé průlety částice detektorovou hladinou nejsou zaznamenány. Pro naše účely budeme předpokládat, že všechny částice v detektoru vznikly v místě terče a že se pohybují s vysokou energií. Nepředpokládáme tedy, že některé částice vznikly během průletu detektorem, protože vytvořit takový model by vyžadovalo výrazně hlubší porozumění problematice kvantové chromodynamiky a kvantové fyziky obecně, nehledě na fakt, že se jedná o *cutting edge* odvětví, které je momentálně intenzivně zkoumáno a výzkum takovýchto modelů je jedna z motivací experimentu CBM.
- **všechny částice mají dlouhou střední délku života:** Předpokládáme, že všechny částice, které v rámci detektoru simulujeme, mají dostatečně dlouhou střední délku života, aby stihly ze sledovaného prostoru detektoru vyletět. Toto zjednodušení zavádíme kvůli tomu, aby bylo chování částic konzistentní s ostatními zjednodušeními modelu: Pokud bychom připustili, že některé částice mají kratší délku života, než je potřebné na opuštění prostoru stanoviště STS, a museli bychom v tom případě řešit problém, kdy uvnitř detektoru dochází k sekundárním rozpadům částic, které generují částice nové, sekundární. Tohle chování by bylo ale v rozporu s předchozím předpokladem, proto ho pro naše účely zanedbáváme.
- **neuvažujeme mnohonásobný rozpad:** Pro naše účely nebudeme brát v úvahu mnohonásobný rozpad částic vzhledem k tomu, že bychom museli na místě každé detektorové hladiny přepočítávat nejen trajektorii částice, ale i její energii, což je netriviální úkon, který sice zpřesní simulaci částic, toto zpřesnění ale pro nás během implementace sledovacího algoritmu nemá velký přínos. Zanedbání mnohonásobného rozpadu má pro účely simulace a rekonstrukce trajektorií dva výrazné dopady: Zaprvé zjednoduší simulaci průchodu detektorem, kdy odpadá nutnost výpočtu mnohonásobného rozpadu v každé detektorové hladině (při průchodu každou hladinou detektoru

STS dochází v závislosti na úhlu dopadu kolem 24000 srážek, které ovlivňují výslednou trajektorii částice a její energii), zároveň zjednoduší proces Kalmanovy Filtrace tím, že můžeme fyzikální model definovat přesně a procesní šum modelu je díky tomu zanedbatelný. Zde je třeba zmínit, že i když sestavení matice procesního šumu není příliš komplikovaný proces, samotný výpočet procesního šumu nemá principiální vliv na rekonstrukční algoritmus ani jeho složitost.

- **nulové elektrické pole:** Protože se částice pohybují relativistickými rychlostmi, je výpočet jejího chování výrazně složitější. Přítomnost nenulového elektrického pole by způsobilo neúměrné ztížení práce se simulátorem, přičemž jeho zohlednění přinese pouze minimální zpřesnění modelu (magnetické pole je v tomto případě výrazně dominantní silou). Díky tomuto předpokladu získáme velkou výhodu z toho důvodu, že na rozdíl od magnetického pole elektrické pole ovlivňuje množství energie částice, což v důsledku zákonů speciální relativity způsobuje, že se mj. hmotnost částice stává proměnnou. V opačném případě je možné vlivy speciální relativity vypočítat již během inicializace simulace a během simulace *de facto* pracujeme s klasickou mechanikou.

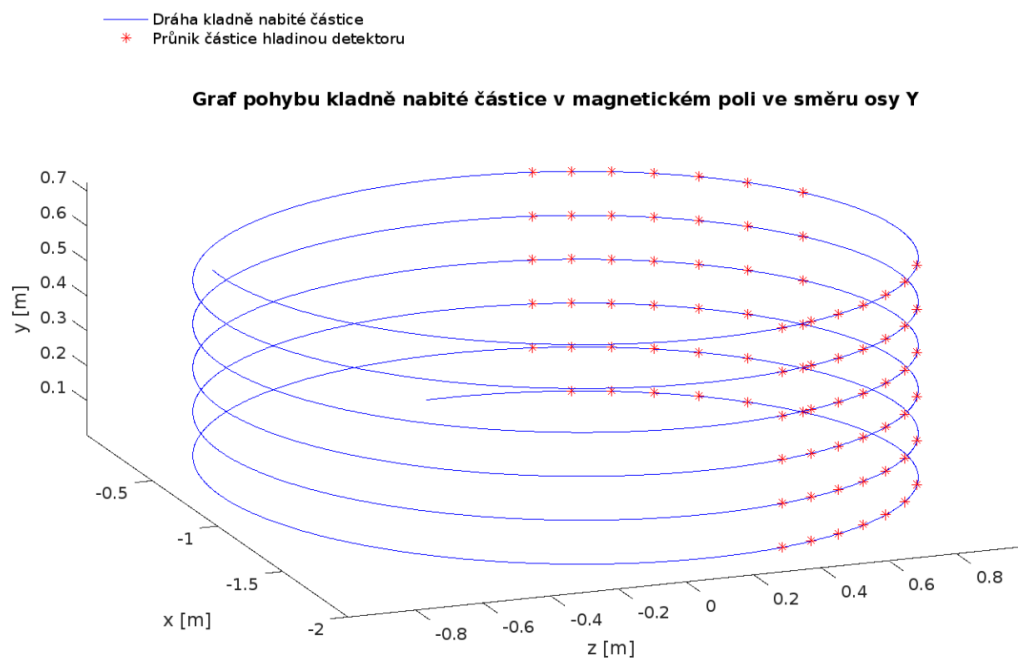
Všechna tato zobecnění nám umožní vytvořit vhodný simulátor, který bude generovat částice vhodné pro vývoj a validaci sledovacího algoritmu. Takový přístup má nevýhodu v tom, že takto vytvořená testovací data nejsou vhodná pro hodnocení výkonu výsledného algoritmu, vývoj takového simulátoru by ale byl nad rámec této práce, zjednodušený simulátor je proto pro tuto práci vyhovující.

## 4.2 Metody řešení diferenciálních rovnic

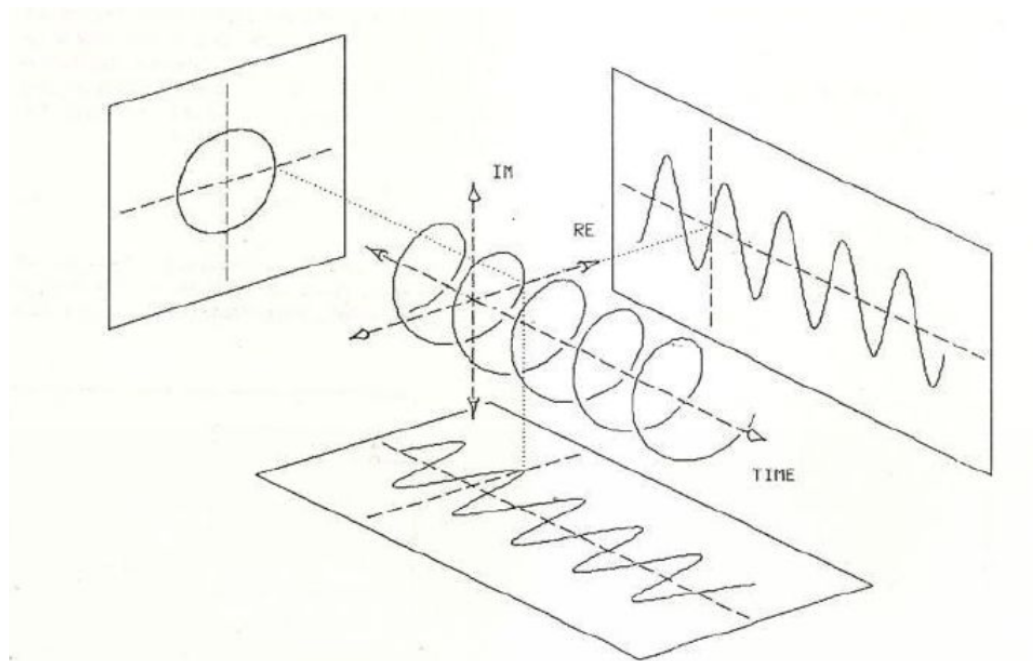
Abychom mohli úspěšně simulovat průlet částic detektorem, je třeba správně implementovat zvolený fyzikální model, definovaný soustavou diferenciálních rovnic vyšších řádů. Proto je nutné správně zvolit metodu řešení diferenciálních rovnic, na které lze potom celý simulační model postavit. Pro účely této práce bylo zvoleno numerické řešení, za zmínku ale stojí i možnost **analytických řešení** trajektorie, přestože pro naše účely nejsou zcela vhodné (simulační model je použitý i v rámci vlastního sledovacího algoritmu, kde již s homogenním magnetickým polem nelze počítat, numerické řešení je zde vhodnější): V případě, že se nabitá částice pohybuje ve směru kolmém na směr magnetického pole, její trajektorie pak nabývá tvaru **krůžnice**. V případě, že se směr částice kolmý není (tj. její má nenulovou složku  $\dot{y}$ , popřípadě  $\theta_y$ ), trajektorie nabývá tvaru **šroubovice**, kterou můžeme vyjádřit jako **komplexní exponenciálu** s osou ve směru magnetického pole (v našem případě ve směru osy  $y$ ):

$$f(y) = R \times e^{i2\pi ky} + w \quad (4.1)$$

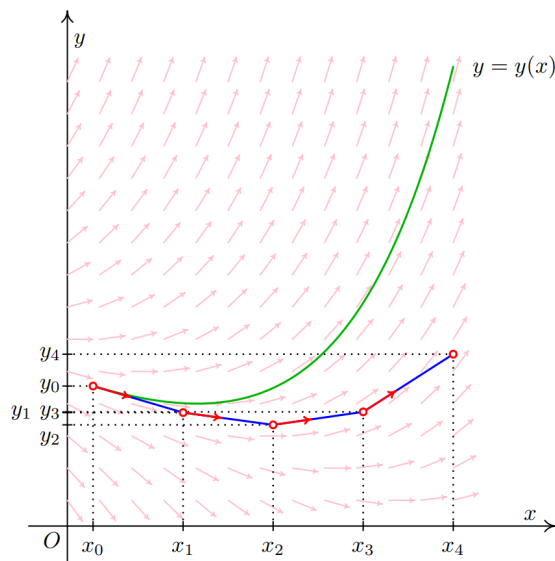
kde  $R$  je poloměr otáčení šroubovice,  $e$  je Eulerovo číslo,  $i$  je imaginární jednotka,  $k$  je perioda exponenciály a  $w \in \mathbb{I}$  je posunutí osy otáčení exponenciály v komplexní rovině, která je kolmá na směr magnetického pole (v našem případě rovina  $xz$ ).  $R$  můžeme také definovat jako poloměr oblouku trajektorie, jeho znaménko pak určuje směr zakřivení, který je určen nábojem částice. Vzhledem k tomu, že simulační model předpokládá, že všechny částice mají počátek v terči, posunutí  $w$  a poloměr  $R$  se stávají lineárně závislými, a tak můžeme nabitou částici v homogenním magnetickém poli vyjádřit pomocí jednoho reálného parametru  $k$  a jednoho komplexního parametru  $w$ , kde parametr  $k$  udává úhel  $\theta_y$  a parametr



Obrázek 4.1: Trajektorie pomalé, kladně nabitě teoretické částice v detektoru s homogenním magnetickým polem ve směru osy Y. Červeně jsou vyznačeny průniky částice s hladinami detektoru, modře pak vlastní trajektorie částice. Trajektorie byla vytvořena pomocí simulačního modelu a výsledný graf byl vytvořen v programu MATLAB.



Obrázek 4.2: Komplexní exponenciála a její projekce do reálné roviny (kosinusovka), imaginární roviny (sinusovka) a komplexní roviny (kružnice). Převzato z [2].



Obrázek 4.3: Graf fungování jednoduché Eulerovy metody. Zeleně je vyznačena reálná funkce, modře jednotlivé kroky Eulerovy metody. Obrázek byl převzat ze skript pro předmět Numerická matematika a pravděpodobnost na Fakultě Informačních Technologií VUT [13].

$w$  osu otáčení komplexní exponenciály. Toto řešení má ale dvě okrajové situace, kvůli kterým je pro účely tohoto simulátoru nevhodné:

- **částice s nulovým nábojem:** v případě, že takto modelovaná částice má nulový náboj, vzniká problém s parametrem  $w$ , který v tomto případě nemá řešení (u přímé trajektorie nelze určit poloměr, proto nelze určit ani bod na komplexní rovině, kterým by procházela osa komplexní exponenciály).
- **částice s nulovým úhlem  $\theta_y$ :** částice, jejíž trajektorie je rovnoběžná s rovinou  $xz$  taktéž nemá v tomto vyjádření řešení, protože taková komplexní exponenciála by měla teoreticky nulovou periodu, narážíme tedy na analogický problém, jako při Houghově transformaci svislé přímky (viz obrázek 3.5).

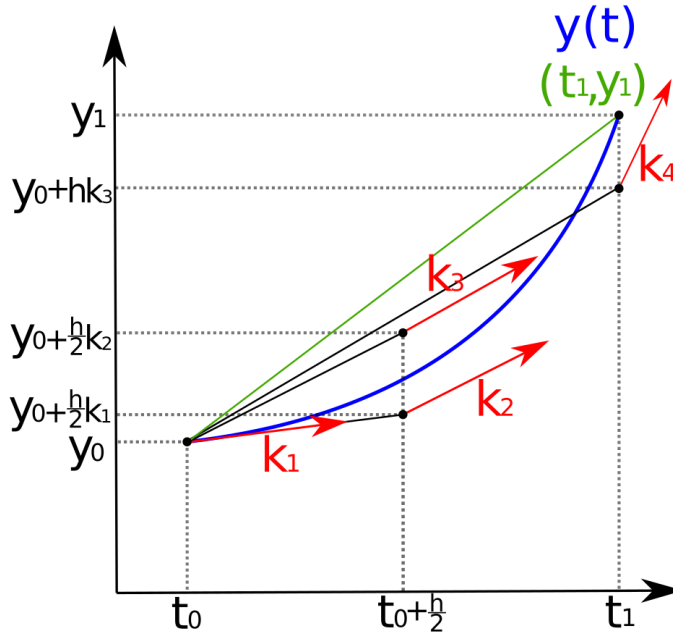
Z tohoto důvodu se tato práce nebude možnostmi analytického řešení trajektorií v detektoru dále zabývat.

## Numerické metody

Po vyloučení použití analytických metod je třeba přistoupit k použití numerických metod. V úvahu jsou brány zejména dvě jednokrokové metody: **Eulerova metoda** a **metoda Runge-Kutta** čtvrtého řádu (dále RK4):

**Eulerova metoda** je nejjednodušší metodou numerického řešení obyčejných diferenciálních rovnic s danými počátečními podmínkami. Mějme obyčejnou diferenciální rovnici prvního řádu:

$$y'(t) = f(t, y(t)), y(t_0) = y_0 \quad (4.2)$$



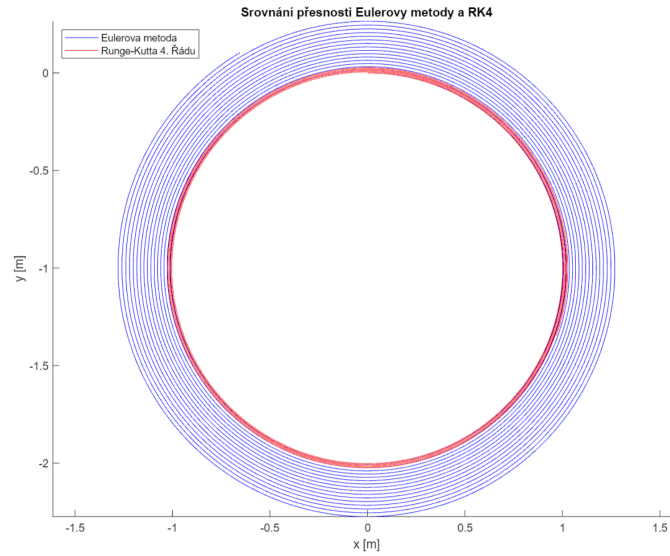
Obrázek 4.4: Graf fungování Runge-Kuttovy metody čtvrtého řádu. Modře je vyobrazená reálná funkce, zeleně je vyobrazená výsledná hodnota  $y_{n+1}$ , červeně jsou zobrazeny jednotlivé koeficienty. Obrázek byl převzat z dokumentace metody RK4 na stránkách *Read the Docs* [26].

jejíž hodnotu chceme numericky spočítat. Zvolíme délku kroku numerické metody  $h$  (pro každé  $t_n$  platí  $t_n = t_0 + nh$ ) a pomocí rekurentního vzorce Eulerovy metody začneme s numerickou integrací:

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n) \\ t_{n+1} &= t_n + h \end{aligned} \quad (4.3)$$

přičemž platí, že počáteční hodnota  $y_0$  v čase  $t_0$  je zadána. Myšlenka Eulerovy metody je, že i když neznáme podobu křivky, kterou zkoumáme, známe počáteční bod, ze kterého vycházíme a z diferenciální rovnice dokážeme vypočítat sklon funkce v tomto počátečním bodě, čímž získáme tečnu, po které se posuneme o krátký časový úsek (krok)  $h$ .

Alternativou Eulerovy metody jsou Runge-Kuttovy metody. Samotná Eulerova metoda odpovídá Runge-Kuttově metodě prvního řádu, pro použití v simulátoru částic byla ale použita nejznámější z Runge-Kuttových metod, konkrétně metoda Runge-Kutty čtvrtého řádu:



Obrázek 4.5: Srovnání propagace chyby numerických metod Runge-Kutta 4. řádu (červeně) a Eulerovy metody (modře). Data byla vygenerována simulátorem částic a zpracována do grafu programem MATLAB.

$$\begin{aligned}
 t_{n+1} &= t_n + h, \\
 y_{n+1} &= y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) h, \\
 k_1 &= f(t_n, y_n), \\
 k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \\
 k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \\
 k_4 &= f(t_n + h, y_n + hk_3),
 \end{aligned} \tag{4.4}$$

Kde  $k_1, \dots, k_4$  jsou koeficienty, definující:

- $k_1$  je sklon funkce v bodě  $y_n$  (Eulerova metoda),
- $k_2$  je sklon funkce ve středu intervalu, v bodě získaným pomocí koeficientu  $k_1$  (metoda **Midpoint**),
- $k_3$  je opět sklon funkce ve středu intervalu, tentokrát bod získává na základě koeficientu  $k_2$ ,
- $k_4$  je sklon na konci intervalu, k jeho výpočtu je potřeba koeficient  $k_3$

## Stabilita numerických metod

Vzhledem k povaze simulačního modelu, kdy se nabitá částice ve směru kolmém na magnetické pole (za předpokladu, že se nemění kinetická energie částice působením elektrického

pole) pohybuje po kružnici (respektive po šroubovici za předpokladu, že složka hybnosti částice ve směru magnetického pole je nenulová), je třeba se pozastavit nad problematikou stability vybraných numerických metod. Bohužel v případě nabitých částic bude jak Eulerova metoda, tak metoda RK4 vždy nestabilní (chování Eulerovy metody a RK4 viz obrázek 4.5). Metoda  $n$ -tého řádu je schopna přesně určit řešení diferenciální rovnice za předpokladu, že přesné řešení diferenciální rovnice je polynom řádu maximálně  $n$ . Eulerova metoda tedy přesně určí řešení, pokud je přesné řešení přímka, u RK4 polynom maximálně čtvrtého řádu atd. Vzhledem k tomu, že pohyb po kružnici (respektive po šroubovici) se jako polynom zapsat nedá, lze přesně spočítat pouze tu složku pohybu, která je rovnoběžná se směrem magnetického pole.

V důsledku tohoto faktu je důležité, aby byla vhodně zvolena délka kroku metody, která sice nezaručí stabilitu, ale poskytne vhodný kompromis mezi rychlostí výpočtu a přesností metody. V tomto ohledu je metoda Runge-Kutta výrazně spolehlivější než Eulerova metoda, přitom poskytuje dobrý kompromis mezi přesností a výpočetní náročností: RK4 je nejvyšší řád Runge-Kuttových metod, kde řád metody odpovídá počtu jejích kroků (RK1/Eulerova metoda: 1 krok, ... RK4: 4 kroky, ale již RK5: 6 kroků), metody vyšších řádů pak rychle nabývají na výpočetní náročnosti a pro nás jsou tak méně vhodné.

### Použití numerických metod pro rovnice vyšších řádů

Přestože jsou Eulerova metoda i metoda Runge-Kutta určeny pro řešení diferenciálních rovnic prvního řádu, můžeme ji použít i pro obyčejné diferenciální rovnice vyšších řádů tak, že převedeme diferenciální rovnici vyššího řádu na soustavu rovnic prvního řádu. Mějme diferenciální rovnici  $N$ -tého řádu:

$$y^{(N)}(t) = f\left(t, y(t), y'(t), \dots, y^{(N-1)}(t)\right) \quad (4.5)$$

zavedeme pomocné proměnné  $z$  následujícím způsobem:

$$\begin{aligned} z_1(t) &= y(t) \\ z_2(t) &= y'(t) \\ &\vdots \\ z_N(t) &= y^{(N-1)}(t) \end{aligned} \quad (4.6)$$

Obyčejnou diferenciální rovnici vyššího řádu pak můžeme přepsat jako následující soustavu diferenciálních rovnic prvního řádu:

$$z'(t) = \begin{bmatrix} z_1'(t) \\ z_2'(t) \\ \vdots \\ z_{N-1}'(t) \\ z_N'(t) \end{bmatrix} = \begin{bmatrix} z_2(t) \\ z_3(t) \\ \vdots \\ z_N(t) \\ f(t, z_1(t), \dots, z_N(t)) \end{bmatrix} = \begin{bmatrix} y'(t) \\ y''(t) \\ \vdots \\ y^{(N-1)}(t) \\ y^{(N)}(t) \end{bmatrix} \quad (4.7)$$



### 4.3 Pohybové rovnice simulačního modelu, první simulační model

První verze simulátoru dráhy částic byl navržen tak, aby co nejpřesněji modeloval dráhu teoretických, „pomalých“ částic. Pro popis dráhy pomalé částice v elektromagnetickém poli slouží jev zvaný **Lorentzova síla**, která popisuje sílu působící na nabitou částici v elektromagnetickém poli.

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \quad (4.8)$$

kde  $\mathbf{E}$  je vektor elektrického pole,  $\mathbf{v}$  je vektor rychlosti částice v elektromagnetickém poli a  $\mathbf{B}$  je vektor magnetického pole. Jak bylo zmíněno výše, v rámci simulátoru je elektrické pole  $\mathbf{E}$  zanedbáno, lze tedy rovnici přepsat pro účely simulátoru následujícím způsobem:

$$\mathbf{F} = q\mathbf{v} \times \mathbf{B} \quad (4.9)$$

Všimněme si, že v tuto chvíli v rovnici nefiguruje hmotnost částice, protože velikost Lorentzovy síly je na ní nezávislá. Hmotnost částice ale potřebujeme znát pro výpočet zrychlení částice, aplikujeme tedy druhý Newtonův zákon  $F = ma$ , díky čemuž dostáváme rovnici:

$$\dot{\mathbf{v}} = \frac{q}{m}\mathbf{v} \times \mathbf{B} \quad (4.10)$$

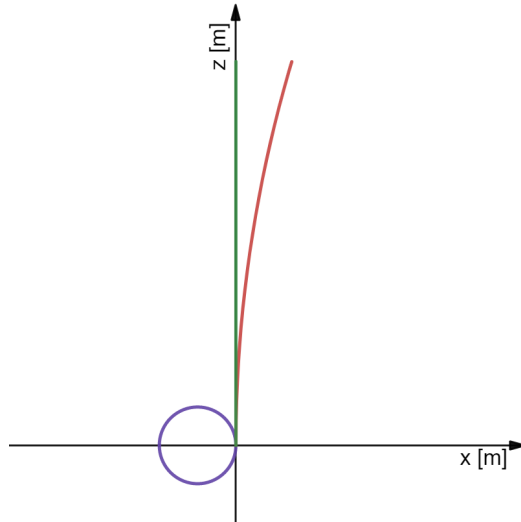
Kde  $\dot{\mathbf{v}} = \mathbf{a}$  je vektor zrychlení částice a  $m$  je její hmotnost. V této rovnici lze pro potřeby simulátoru rozepsat vektory následujícím způsobem:

$$\begin{aligned} \ddot{x} &= \frac{q}{m}(\dot{y}B_z - \dot{z}B_y) \\ \ddot{y} &= \frac{q}{m}(\dot{z}B_x - \dot{x}B_z) \\ \ddot{z} &= \frac{q}{m}(\dot{x}B_y - \dot{y}B_x) \end{aligned} \quad (4.11)$$

kde  $\dot{x}$  je rychlost částice ve směru  $x$ ,  $\ddot{x} = \dot{v}_x = a_x$  značí zrychlení ve směru dané osy,  $B_x$  je velikost intenzita magnetického pole ve směru  $x$  (analogicky pro směry  $y$  a  $z$ ). Tímto způsobem jsme získali soustavu tří obyčejných diferenciálních rovnic, které nám poslouží jako první simulační model. Za pozornost stojí ještě fakt, že v případě homogenního magnetického pole, které má směr rovnoběžný s některou z os je možné model dále zjednodušit: Uvažujme případ detektoru, ve kterém má magnetické pole směr rovnoběžný s osou  $y$ , model lze pak zjednodušit následujícím způsobem:

$$\begin{aligned} \ddot{x} &= -\frac{q}{m}\dot{z}B \\ \ddot{z} &= \frac{q}{m}\dot{x}B \end{aligned} \quad (4.12)$$

Kde  $B = B_y$  je magnetické pole. Všimněme si, že rovnici pro výpočet  $\ddot{y}$  můžeme zcela zanedbat, zrychlení v tomto směru se vždy vyhodnotí jako nulové, v tomto směru se tedy částice budou pohybovat rovnoměrným přímočarým pohybem. Zjednodušení simulátoru tímto způsobem je sice možné, pro účely simulátoru je ale výhodné mít možnost velikost a orientaci magnetického pole nastavovat, bylo tedy učiněno rozhodnutí implementovat obecnější variantu z rovnice 4.11.



Obrázek 4.6: Demonstrace simulace pohybu různých, nerelativistických částic. Červeně je vyznačena dráha těžké / slabě nabité částice s kladným nábojem, modře je vyznačena lehká / silně nabitá částice se záporným nábojem a zeleně je vyznačena dráha částice s nulovým nábojem. Směr magnetického pole je rovnoběžný s osou  $y$ , všechny částice mají shodnou počáteční rychlost  $\mathbf{v} = (0, 0, 1) \text{ ms}^{-1}$ . Graf byl vytvořen za pomoci nástroje Desmos [9].

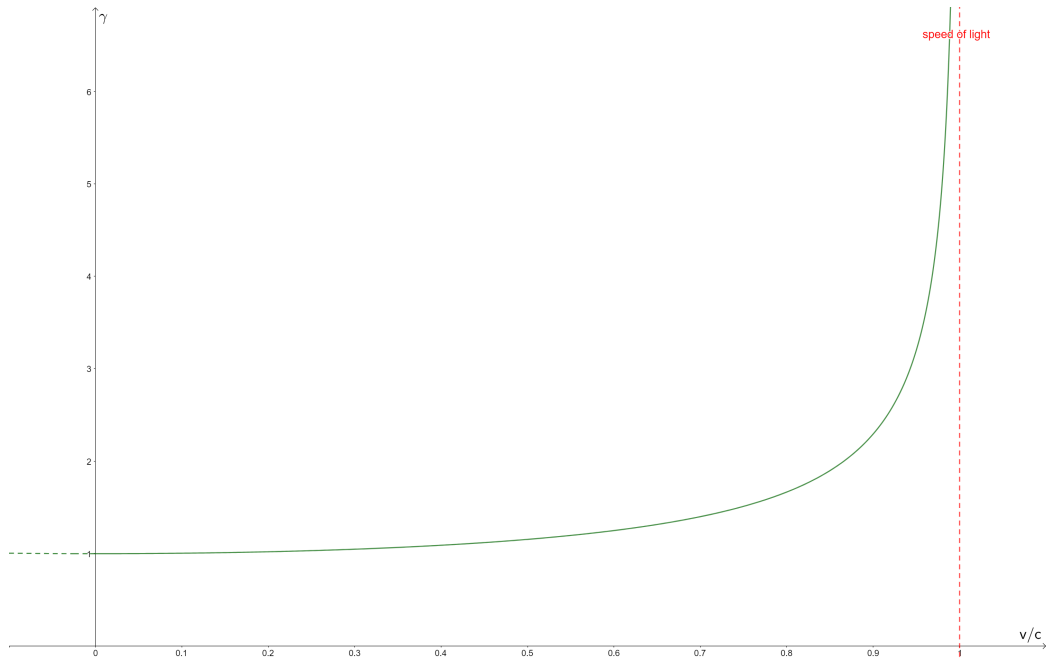
#### 4.4 Speciální relativita v pohybových rovnicích, druhý simulační model

Simulační model popsany výše má zásadní nedostatek, který je patrný ve chvíli, kdy pomocí tohoto simulátoru vygenerujeme první částice s úkolem „modelovat“ reálné částice (například protony či tauony): Protože používáme pro výpočet zrychlení klasické zákony mechaniky, nastává problém ve chvíli, kdy simulátor začne generovat částice při relativistických rychlostech. Protože model nebere zákony speciální relativity v potaz, simulátor může generovat částice, které v praxi neexistují a vzhledem k tomu, že u částic v detektoru CBM očekáváme rychlosti přes  $0.9c$ , spoléhat se na klasickou mechaniku je nevhodné. Další z nedostatků je, že simulátoru teoreticky nic nebrání generovat částice s rychlostmi přesahujícími rychlost světla.

Abychom mohli simulační model upravit tak, aby zohledňoval relativistické chování částic, musíme nejprve zavést takzvaný **Lorentzův faktor**:

$$\gamma \equiv \frac{dt}{d\tau} = \frac{c}{\sqrt{c^2 - v^2}} = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}} \quad (4.13)$$

Kde  $c$  značí rychlost světla ve vakuu,  $\tau$  je takzvaný **vlastní čas**. Pohybové rovnice předchozího modelu (rovnice 4.11), po korekci pro speciální relativitu mají pak následující tvar:



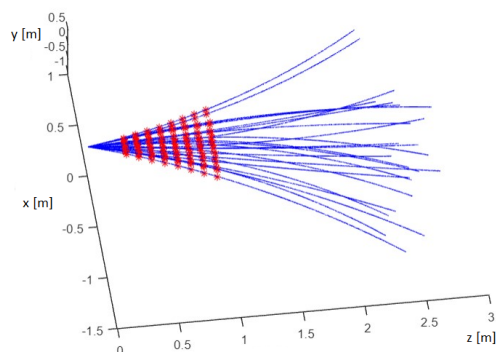
Obrázek 4.7: Graf funkce rychlosti a Lorentzova faktoru. Při rychlostech blízcích se rychlosti světla Lorentzův faktor stoupá k nekonečnu. Graf byl vytvořen pomocí nástroje Desmos [9].

$$\begin{aligned}
 \ddot{x} &= \frac{q}{\gamma m} (\dot{y}B_z - \dot{z}B_y) \\
 \ddot{y} &= \frac{q}{\gamma m} (\dot{z}B_x - \dot{x}B_z) \\
 \ddot{z} &= \frac{q}{\gamma m} (\dot{x}B_y - \dot{y}B_x)
 \end{aligned}
 \tag{4.14}$$

Zavedení Lorentzova faktoru do pohybových rovnic znamená, že dosud konstantní hmotnost částice  $m$  se stává proměnnou, závislou na rychlosti částice. Rozhodnutí předpokládat v simulátoru homogenní magnetické pole a nulovou intenzitu elektrického pole se zde projeví jako výhodné z důvodu, že bez vlivu elektrického pole velikost rychlosti částice  $v$  zůstává konstantní, proto zůstává konstantní i její celková hmotnost, kterou pouze při inicializaci simulace upravíme na základě klidové hmotnosti a Lorentzova faktoru. V případě druhého, pro speciální relativitu upraveného simulátoru již dokážeme efektivně modelovat částice s relativistickou rychlostí.

## 4.5 Kombinovaný simulátor částic s logováním zásahů v detektoru

Generování cvičných částic bylo implementováno pro demonstrační účely ve formátu Jupyter notebook, původně v jazyce Python 3.9.4 (za účelem rychlého prototypování). V průběhu práce byl simulátor přepsán v jazyce Julia 1.6.1. díky podobnostem s jazykem Python a výrazně vyšší rychlosti zpracování [4]. Simulátor byl také implementován i v jazyce C pro porovnání rychlostí jednotlivých implementací.

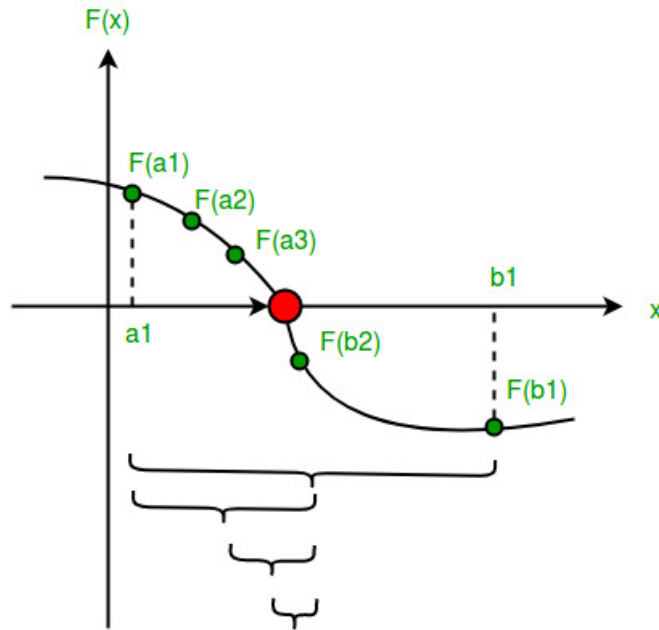


Obrázek 4.8: Graf 25 náhodně vygenerovaných částic, vlastnostmi odpovídajících částicím  $p^+$ ,  $\Lambda_c^+$ ,  $\Sigma^-$  a  $\Sigma_b^-$ . Modře jsou naznačeny vygenerované trajektorie, červeně jsou vyznačeny zásahy částic v hladinách detektoru. Data byla vygenerována pomocí simulátoru částic a graf byl vytvořen pomocí programu MATLAB.

Původně byl simulátor navržen pro spojitou simulaci, která probíhá ve vlastním čase každé částice od času  $t_0 = 0s$  do času  $t_{max} = \frac{1.25}{v}$  sekund, kde 1.25 je délka části dráhy částice, která je ovlivněna supravodivým magnetem CBM. Mimo oblast magnetu není nutné trajektorii částice simulovat, protože oproti oblasti uvnitř magnetu, kde lze magnetické pole *téměř* považovat za homogenní, je výhodnější předpokládat, že mimo magnet je intenzita magnetického pole nulová než se pokoušet magnetické pole jinak aproximovat. Navíc algoritmus pro sledování trajektorií v detektoru používá ve své první fázi pouze data z detektoru STS, který je celý uvnitř magnetu, tvar trajektorie ve vzdálenosti od terče větší než 1 metr pro nás tedy již není zajímavá.

Spojité simulátor používá Runge-Kuttovu metodu pro simulaci trajektorie, jednotlivé kroky pak zapisuje do výstupního `.csv` souboru. Pro tento účel je sice současný návrh simulátoru postačující, takový simulátor ale není schopen přesně určit, kde částice prošla hladinou detektoru. Z tohoto důvodu byl simulátor rozšířen na **kombinovaný simulátor**, díky čemuž jsme schopni obsluhovat *stavové události* (události, které nelze naplánovat, v případě simulátoru částic okamžiky, kdy částice proletí hladinou detektoru).

V okamžiku, kdy se změní stavová podmínka (v případě simulátoru částic taková změna souřadnice  $z$ , která ukazuje na průlet hladinou detektoru mezi jednotlivými měřeními), je potřeba najít čas, kdy ke změně stavové podmínky (a tedy ke stavové události) došlo. Toho lze docílit například pomocí metody **půlení intervalu**. Mějme numerickou metodu s krokem  $h$ , minimálním krokem  $h_{min}$  a stavovou podmínkou: krok metody opakujeme tak dlouho, než stavová podmínka změní hodnotu (tj. dojde ke stavové události), v takový okamžik krok zmenšíme na poloviční a obnovíme stav systému z okamžiku před provedením kroku, který stavovou událost vyvolal. Krok metody a postupné půlení délky kroku provádíme tak dlouho, dokud je délka kroku než  $h_{min}$ , v tu chvíli víme, že při vyvolání stavové události byl nalezen okamžik jejího vyvolání s přesností  $\pm h_{min}$ . Existuje varianta tohoto algoritmu, která neobnovuje původní stav systému, ale provádí zmenšený krok „pозpátku“. Tato verze metody sice konverguje rychleji, z programátorského hlediska bylo ale snazší implementovat původní metodu. Tímto způsobem simulátor nachází okamžiky, ve kterých ke stavovým událostem došlo a umožňuje její další zpracování. Stavová podmínka je v tu chvíli potvrzena, vyhodnotí se stavová událost (například zapsání hodnot stavu



Obrázek 4.9: Metoda půlení intervalu: Metoda provede krok a vyhodnotí, jestli se *změnila hodnota stavové podmínky* a v závislosti na hodnotě stavové podmínky buďto provede další krok, nebo podělí délku kroku dvěma a provede krok znovu. Obrázek byl převzat z webu GeeksforGeeks [5]

systému nebo vyhodnocení procesu mnohonásobného rozptylu, který sice tento simulátor zanedbává, při použití ve vlastním sledovacím algoritmu je ale jeho vyhodnocení na místě).

Stavové události jsou vnitřně zpracovávány pomocí jedné funkce, která při každém kroku porovná polohu částice před a po provedení kroku numerické metody. Funkce má uloženy  $z$  souřadnice všech hladin detektoru, a pokud je některá z těchto hladin **mezi** dvěma hodnotami  $z$  souřadnice před a po provedení kroku, vyvolá stavovou událost. Stavové události jsou, podobně jako data o trajektorii částice, ukládány do samostatného `.csv` souboru, který bude po anonymizaci (program do `.csv` souboru zapisuje jednoznačné *id* částice, která zásah způsobila) a zašumění dat možné použít jako testovací data pro vývoj sledovacího algoritmu.

### Struktura částice v detektoru

Každá částice je v simulátoru definována jako struktura, nesoucí data o všech vlastnostech a stupních volnosti, potřebných k její jednoznačné definici:

- jednoznačný identifikátor částice, sloužící ke kontrole správnosti rekonstruovaných částic (lze tímto způsobem kontrolovat, jestli zásahy detektoru, které algoritmus spojil jako trajektorii jedné částice, skutečně byly vyvolány stejnou částicí),
- počáteční čas simulace částice  $t$ , který,
- koncový čas simulace  $t_{max}$ , který říká simulátoru, kdy má ukončit práci s danou částicí (ať už díky tomu, že částice opustila detektor, nebo došlo k sekundárnímu rozpadu),
- klidovou hmotnost částice  $m_0$  na základě které dopočítáváme Lorentzův faktor částice pro úpravu vlastností při relativistickém pohybu,

- náboj částice  $q$ ,
- stavový vektor částice s informacemi o poloze a rychlosti částice,
- Lorentzův faktor  $\gamma$ ,
- Konstantu  $C_{\gamma qm} = \frac{q}{\gamma m}$ , kterou použijeme při výpočtu zrychlení.

### simulace průletu částice detektorem

Průlet částice detektorem je simulován tak, že nejprve inicializujeme stavový vektor částice, počáteční a koncový čas simulace, její náboj a klidovou hmotnost. Pro zjednodušení generování nových částic jsou v rámci souboru `physics.h` definovány základní potřebné konstanty jako  $\pi$ , rychlost světla  $c$  nebo elementární náboj  $e$ . Dále je v souboru definován katalog klidových hmotností a kvantových nábojů (náboj částice  $q$  získáme tak, že její kvantové číslo náboje vynásobíme konstantou elementárního náboje) známých, pozorovatelných částic na základě kterých lze následně dopočítat Lorentzův faktor. Po zapsání hodnot částice je struktura částice předána funkci `calculate_gqm()`, která vypočítá a uloží hodnotu Lorentzova faktoru. V tuto chvíli je částice připravena k simulaci.

Zmiňme ještě funkci `simulation_loop()`, která zprostředkovává vlastní proces numerické simulace. Funkce má na starosti logování dat simulace (za tím účelem jako parametry přijímá dva souborové deskriptory, do kterých data zapisuje: první soubor je určen logování trajektorie, druhý soubor ukládá zásahy v hladinách detektoru), kontrolu (a obsluhu) stavových podmínek a vlastní krokování vybrané numerické metody. V zájmu obecnosti implementace má funkce parametr `num_method`, ukazatel na funkci, pomocí kterého můžeme při volání funkce `simulation_loop()` určit, jakou metodu si přejeme pro simulaci použít (implementovány jsou Eulerova metoda a metoda RK4).

Implementace simulátoru v Julia je velmi podobná implementaci v C, s tím rozdílem, že Julia poskytuje za účelem fyzikálních simulací modul `PhysicalConstants.jl`, který obsahuje všechny informace o důležitých fyzikálních konstantách, zároveň definuje jednotky těchto konstant a umožňuje při programování v jazyce Julia pracovat s fyzikálními jednotkami, čímž usnadňuje vývoj a verifikaci fyzikálních výpočtů. Pro práci s logovanými daty trajektorií je pak použit modul `DataFrames.jl`, který umožňuje rychlou a výpočetně nenáročnou práci s těmito daty (`DataFrames.jl` implementuje sadu funkcí nad datovými rámci podobným způsobem jako v jazyce Python knihovna `Pandas` s tím, že všechny *element-wise* operace jsou implicitně vektorizovány pro rychlejší zpracování).

### Odvození fyzikálního modelu pro Kalmanův Filtr

Protože Kalmanův filtr, který je nezbytný k fungování sledovacího algoritmu, potřebuje vhodný fyzikální model k tomu, aby správně odhadoval příští stav systému, bylo by výhodné použít stejný model, který již používáme pro generování testovacích dat. Fyzikální model odvodíme tak, jak bychom model odvozovali (a dále budeme odvozovat, v případě dalších modelů) v případě použití rozšířeného Kalmanova Filtru (EKF):

Mějme soustavu devíti rovnic, vycházejících ze soustavy 4.14:

$$\begin{aligned}
x_k &= x_{k-1} + \Delta t \dot{x}_{k-1} + \frac{\Delta t^2}{2} \ddot{x}_{k-1} \\
\dot{x}_k &= \dot{x}_{k-1} + \Delta t \ddot{x}_{k-1} \\
\ddot{x}_k &= \frac{q}{\gamma m} (\dot{y}_{k-1} B_z - \dot{z}_{k-1} B_y) \\
y_k &= y_{k-1} + \Delta t \dot{y}_{k-1} + \frac{\Delta t^2}{2} \ddot{y}_{k-1} \\
\dot{y}_k &= \dot{y}_{k-1} + \Delta t \ddot{y}_{k-1} \\
\ddot{y}_k &= \frac{q}{\gamma m} (\dot{z}_{k-1} B_x - \dot{x}_{k-1} B_z) \\
z_k &= z_{k-1} + \Delta t \dot{z}_{k-1} + \frac{\Delta t^2}{2} \ddot{z}_{k-1} \\
\dot{z}_k &= \dot{z}_{k-1} + \Delta t \ddot{z}_{k-1} \\
\ddot{z}_k &= \frac{q}{\gamma m} (\dot{x}_{k-1} B_y - \dot{y}_{k-1} B_x)
\end{aligned} \tag{4.15}$$

Nyní můžeme definovat vektorově ohodnocenou funkci  $\mathbf{f}_k(\mathbf{r}_k^+) : \mathbb{R}^9 \rightarrow \mathbb{R}^9$ , která obsahuje pravé strany rovnice 4.15:

$$\mathbf{r}_k^- = \mathbf{f}_k(\mathbf{r}_{k-1}^+) = \begin{bmatrix} x_{k-1} + \Delta t \dot{x}_{k-1} + 0.5\Delta t^2 \ddot{x}_{k-1} \\ \dot{x}_{k-1} + \Delta t \ddot{x}_{k-1} \\ q\gamma^{-1}m^{-1}(\dot{y}_{k-1}B_z - \dot{z}_{k-1}B_y) \\ y_{k-1} + \Delta t \dot{y}_{k-1} + 0.5\Delta t^2 \ddot{y}_{k-1} \\ \dot{y}_{k-1} + \Delta t \ddot{y}_{k-1} \\ q\gamma^{-1}m^{-1}(\dot{z}_{k-1}B_x - \dot{x}_{k-1}B_z) \\ z_{k-1} + \Delta t \dot{z}_{k-1} + 0.5\Delta t^2 \ddot{z}_{k-1} \\ \dot{z}_{k-1} + \Delta t \ddot{z}_{k-1} \\ q\gamma^{-1}m^{-1}(\dot{x}_{k-1}B_y - \dot{y}_{k-1}B_x) \end{bmatrix} \tag{4.16}$$

Protože v tomto modelu předpokládáme, že je magnetické pole homogenní a elektrické pole nulové, fyzikální model je lineární (za povšimnutí stojí i fakt, že v důsledku nulového elektrického pole se nemění celková energie částice, tedy ani její hmotnost, můžeme tedy proměnnou hmotnosti  $m$  nahradit konstantou klidové hmotnosti částice  $m_0$ ). Díky tomuto faktu není nutné systém linearizovat a můžeme rovnou sestavit propagační matici  $F_k$ :

$$F_k = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{qB_z}{\gamma m_0} & 0 & 0 & -\frac{qB_y}{\gamma m_0} & 0 \\ 0 & 0 & 0 & 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & -\frac{qB_z}{\gamma m_0} & 0 & 0 & 0 & 0 & 0 & \frac{qB_x}{\gamma m_0} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & \frac{qB_y}{\gamma m_0} & 0 & 0 & -\frac{qB_x}{\gamma m_0} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.17}$$

Zde je na místě poznamenat, že v případě modelu, který by ve své propagační matici používal stavové proměnné, bylo by již nutné pro použití Kalmanovým Filtrem Matici linearizovat podle vhodně zvoleného  $\mathbf{r}_k^{lin}$ .

Jak bylo zmíněno výše, pro homogenní magnetické pole, které je rovnoběžné s některou z os, je možné model ještě mírně zjednodušit. Mějme homogenní magnetické pole, působící ve směru osy  $y$ . Pohybová složka ve směru  $y$  se tím značně zjednoduší, protože se vždy bude jednat jen o rovnoměrný přímočarý pohyb, což mimo jiné znamená, že můžeme zcela vynechat proměnnou  $y_k$  a zmenšit tak velikost stavového vektoru na 8. Výsledná predikční matice pak vypadá následovně:

$$F_k = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{qB_y}{\gamma m_0} & 0 \\ 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & \frac{qB_y}{\gamma m_0} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.18)$$

### Nevýhody simulačních modelů s časovým rozměrem

Simulátor částic je sice funkční a spolehlivě modeluje i speciální teorii relativity, při pokusu o vytvoření Kalmanova filtru pro použití v rekonstrukčním algoritmu ale vyvstává další problém: Ačkoliv pro naše účely je v pořádku předpokládat, že magnetické pole je konstantní a dokonale rovnoběžné s osou  $y$ , v reálném detektoru tento předpoklad zavést nelze. Situaci dále komplikuje fakt, že reálné magnetické pole je natolik komplexní, že se nevyplatí ho popisovat analyticky. To je v současné době řešeno tak, že rekonstrukční program drží v paměti velmi detailní 3D mapu směru a intenzity magnetického pole, se kterým potom pracuje. Z tohoto důvodu v reálném rekonstrukčním programu není pro predikci příštího stavu použit prostý výpočet  $\mathbf{r}_k^- = \mathbf{f}_k(\mathbf{r}_{k-1}^+)$ , ale následující odhad je získán pomocí numerické simulace od detektorové hladiny  $a$  k detektorové hladině  $b$ . Simulátor částic tedy využijeme i v rámci vlastního rekonstrukčního algoritmu. Simulátor ale vzhledem k tomu, že pracuje v čase, nedokáže předem určit, kdy simulaci zastavit a pro získání příští predikce musí čekat na stavovou událost. Kontrola stavových podmínek je výpočetně zbytečně náročná (a v případě devítisložkového stavového vektoru její náročnost dále roste), proto pokud to není nutné, je žádoucí se v rámci vlastního rekonstrukčního programu kombinovaným simulátorům vyhnout.

Z tohoto důvodu je třeba zavést třetí simulační model, tentokrát postavený tak, aby nesimuloval pohyb částice v závislosti na čase, ale na **souřadnici z**.

## 4.6 Pohyb částice v závislosti na vzdálenosti od terče detektoru, třetí simulační model

V reakci na problémy s implementací simulátoru zmíněné výše byl na základě práce Dr. Akishiny [1] zaveden poslední fyzikální model a implementován simulátor, tentokrát zásadně odlišný od předchozích verzí. Simulátor vychází z faktu, že částici v magnetickém poli lze jednoznačně popsat pomocí pěti proměnných (říkáme, že stavový prostor má pět stupňů volnosti):

$$\mathbf{r} \{x, y, t_x, t_y, q/p\} \quad (4.19)$$



Kde  $x$  a  $y$  jsou souřadnice částice pro danou souřadnici  $z$ ,  $t_x \equiv \tan \theta_x$  a  $t_y \equiv \tan \theta_y$  označují sklon směru částice vůči rovinám ZX ( $t_y$ ) a ZY ( $t_x$ ) a  $q/p$  označuje poměr mezi nábojem a hybností částice. Rovnice pro výpočet Lorentzovy síly přepíšeme následujícím způsobem: [1]

$$\begin{aligned}\frac{dt_x}{dz} &= c(q/p) t_r (t_y (B_z + t_x B_x) - B_y (1 + t_x^2)) \\ \frac{dt_y}{dz} &= c(q/p) t_r (B_x (1 + t_y^2) - t_x (B_z + t_y B_y)) \\ t_r(z) &= \sqrt{t_x^2 + t_y^2 + 1}\end{aligned}\tag{4.20}$$

Kde  $c$  je rychlost světla a  $B$  je intenzita magnetického pole. Všimněme si, že vzhledem k tomu, že tento model nepoužívá derivace v čase, neznáme rychlost částice, v důsledku čehož je obtížné zjišťovat vliv relativity na dráhu konkrétní částice. Výhodou tohoto přístupu ale je, že i když nevíme, jakou rychlostí částice detektorem proletěla, z podstaty modelu ani není třeba rychlost částice znát, protože částice bude mít stejnou trajektorii nezávisle na tom, jestli je částice těžká a pomalá, nebo lehká a rychlá, protože se bude lišit pouze čas, za který částice detektorem proletí, a ten vzhledem k povaze a vzorkovací frekvenci detektoru nejsme schopni změřit (z pohledu detektoru proletí každá rozpoznatelná částice všemi hladinami detektoru STS ve stejný čas).

Výhoda takového modelu je v tom, že simulátor založený na takovém modelu můžeme spustit v libovolném místě v detektoru a v libovolném místě detektoru můžeme simulaci zase zastavit. Tento fakt nám mimo jiné umožní provádět simulace z bodu  $a$  do bodu  $b$ , což je výhodné pro potřeby rekonstrukčního algoritmu, který nyní může provést pomocí Runge-Kuttovy metody odhad dráhy částice od jedné detektorové hladiny k druhé, ale i pro obecný Kalmanův filtr, jehož predikční matice nyní není závislá na časovém intervalu  $\Delta t$ , který se mezi hladinami detektoru může lišit, ale na místním intervalu  $\Delta z$ , který je v experimentu CBM konstantní (hladiny detektoru jsou rozestavěné po intervalech deseti centimetrů).

Tento přístup má svá vlastní omezení, konkrétně zmíníme dvě:

- vzhledem k tomu, že používáme souřadnici  $z$  jako časovou složku, některé možné dráhy částic nelze tímto způsobem simulovat. Konkrétně se jedná o částice, které z detektoru vyletí pod pravým úhlem k ose  $z$ , a o částice, které mají dostatečně nízkou energii na to, aby se v detektoru „točily“ (viz částice na obrázku 4.1). Intuitivně existují tato omezení z toho důvodu, že v rámci simulace se nemůžeme „vracet v čase“, z matematického hlediska je to pak z toho důvodu, že hodnoty stavových proměnných  $t_x$  a  $t_y$  mají v případě pravého úhlu k ose  $z$  funkční hodnotu limitně v  $\pm \infty$ . Pravděpodobnost výskytu natolik pomalé částice je každopádně velmi nízká. Navíc, v důsledku mnohonásobného rozptylu uvnitř detektorových hladin, jehož vliv je silnější pro částice s nižší energií, je prakticky nemožné, aby byl detektor schopen takovou částici zrekonstruovat.
- protože neznáme rychlost částice, vliv speciální relativity na částici je „skryt“ v proměnné  $p/q$ . To sice pro potřeby filtrace není problém, v případě, že chceme simulovat dráhu reálné částice, výpočet vlivu relativity na ni je netriviální. Nicméně díky faktu, že model nebere v úvahu čas, není toto omezení nijak závažné.

Simulátor byl implementován na velmi podobné bázi, jako první verze simulátoru pro nerelativistické částice, a díky jeho povaze bylo možné upustit od kombinované simulace a simulovat místo toho každý segment trajektorie zvlášť (vzhledem k tomu, že derivujeme podle osy  $z$ , můžeme částici „odstartovat“ v terči a jako  $z_{max}$  nastavit první hladinu detektoru ve 30cm, vyčíst hodnoty jeho stavového vektoru v  $z_{max} = 30$  a inicializovat simulaci znovu, s hodnotami shodnými, jako na konci simulace, počátečním  $z$  v první hladině detektoru a  $z_{max}$  nastavíme na druhou hladinu detektoru ve 40 cm atd.), což je nejen výhodné pro vlastní rekonstrukční algoritmus, ale i výpočetně efektivní pro vlastní simulaci částic.

### **Problém nalezení počátečního stavového vektoru třetího simulátoru**

Z použití třetího simulátoru vyplývá další problém, který značně komplikuje práci s ním: V případě, že nenalezneme vhodné hodnoty pro počáteční stavový vektor částice v simulátoru  $\mathbf{r}_0$ , může se stát, že částice během průletu detektorem nabere sklon devadesáti stupňů k ose  $z$ , tedy ke směru paprsku. V takovém případě takto sestrojený simulátor selže, protože dále není schopen dráhu částice simulovat. Z experimentování s takovým simulátorem dále vyplývá, že nalézt takový stavový vektor (s výjimkou triviálních případů, jako například částice s nulovým nábojem nebo s nulovým počátečním úhlem  $t_x$ ) není jednoduché a pro naše potřeby generování velkého množství testovacích částic je nepraktický. Z tohoto důvodu bylo nakonec ze snahy o sjednocení modelů pro Kalmanův Filtr a generování částic upuštěno a zatímco Kalmanův Filtr používá model závislosti na souřadnici  $z$  (třetí model), trajektorie částic jsou generovány kombinovaným simulátorem s časovým modelem a úpravou Lorentzových rovnic pro relativistické rychlosti v homogenním elektromagnetickém poli.

## Kapitola 5

# Návrh a implementace rekonstrukčního algoritmu

Tato kapitola se podrobně zabývá návrhem a implementací rekonstrukčního algoritmu pro sledování částic v experimentu CBM. Stručně jsou zde shrnuty použité nástroje a programovací jazyky, současně navržené řešení pro platformu *Intel Xeon Phi* profesora Kisela a doktorky Akishiny, velký důraz je kladen na vlastní rekonstrukční algoritmus a jeho rozdíly oproti původnímu návrhu. Dále se v kapitole budeme zabývat možnostmi akcelerace tohoto algoritmu na hardwarových platformách, zejména pak paralelizací výpočtů pomocí protokolu MPI. V závěru kapitoly diskutujeme dosažené zrychlení v důsledku použité metody.

### 5.1 Sekvenční verze rekonstrukčního algoritmu

V této sekci se budeme zabývat plně sekvenční, nijak neparalelizovanou verzí rekonstrukčního algoritmu. V tomto algoritmu také identifikujeme výpočetně nejnáročnější součásti, na které se následně při paralelizaci v dalších sekcích zaměříme.

Rekonstrukční algoritmus, jak již bylo zmíněno, sdružuje dohromady dvě první části rekonstrukce částic v detektorech: **hledání stop** (track finding) a **prokládání trajektorií křivkou** (track fitting). Tyto dvě části sdružujeme kvůli podstatě Kalmanova Filtru, který k hledání stop používáme: v rámci evoluce Kalmanova Filtru počítáme hodnotu  $\chi^2$  (viz rovnice 3.32), která značí, jak moc odpovídá poslední naměřený stav systému poslednímu odhadu Kalmanova Filtru. Pomocí této hodnoty pak měříme, jak moc je pravděpodobné, že série vybraných zásahů patří ke stejné trajektorii. Zároveň však protože hlavní výstup Kalmanova filtru je co nejlepší odhad stavového vektoru, ve chvíli, kdy tímto způsobem nalezneme sadu zásahů, u kterých je nejpravděpodobnější, že byly vytvořeny stejnou trajektorií, jsme zároveň provedli druhou část rekonstrukce, tedy fitování trajektorie. Protože ale fyzikální model pohybu částice v detektoru není lineární, není možné použít klasický Kalmanův Filtr, který je optimální a stabilní, ale jeho rozšířenou podobu (EKF, viz 3.2), která již optimální ani nutně stabilní není. Navíc, z důvodu šetření paměti a zvýšení rychlosti výpočtů, bylo rozhodnuto v původní implementaci algoritmu pro CBM používat při reprezentaci reálných čísel bitovou délku pouze 32 bitů, což dále potenciálně zvyšuje riziko divergence rozšířeného Kalmanova Filtru.

## Inicializace algoritmu

Vstupem rekonstrukčního algoritmu je neseřazené pole všech zásahů. Každý zásah je struktura dvou čísel s plovoucí desetinnou čárkou a jednoho celého čísla, které určuje hladinu detektoru:

```
struct Hit
  x::Float32
  y::Float32
  z::UInt8
end
```

protože algoritmus používá pouze 8 detektorových hladin detektoru STS, které jsou rozestaveny v pravidelných intervalech, nabývá hodnota souřadnice  $z$  ve strukturách zásahů hodnoty 3-10, díky čemuž nám stačí ji reprezentovat pouze jako celé číslo s nejmenší bitovou šířkou, jakou nám program dovolí použít, tedy 8 (jeden byte). V další části algoritmu budeme postupně všechny zásahy sdružovat do trojic a vyhodnocovat jejich  $\chi^2$  hodnoty, podle kterých budeme dále vyhodnocovat nejpravděpodobnější vlastnosti jednotlivých trajektorií.

Nejdůležitější část této části algoritmu je určení první aproximace stavového vektoru  $\mathbf{r}_0^+$  a kovarianční matice  $C_0^+$ . Určení kovarianční matice je triviální (viz rovnice 3.24), vezmeme jednotkovou matici o velikosti 5 a vynásobíme arbitrárním velkým číslem  $\mathbf{Inf}^2$ . V případě  $\mathbf{r}_0^+$  již ale postup triviální není. Můžeme například:

- inicializovat první hodnotu  $\mathbf{r}_0^+$  jako přímočarou trajektorii neutrálně nabitě částice. V takovém případě inicializujeme  $\mathbf{r}_0^+ = \{0, 0, t_x, t_y, 0\}$  tak, že  $t_x = x_0/z$  a  $t_y = y_0/z$ , kde  $x_0, y_0$  jsou souřadnice  $x$  a  $y$  prvního měření, tedy prvního zásahu potenciální trojice.
- inicializovat počáteční hodnoty  $x$  a  $y$  na 0 a  $t_y$  inicializovat jako v prvním příkladu,  $t_x$  pak získat metodou nejmenších čtverců tak, že pohyb trajektorie ve směru  $x$  proložíme parabolou. Hodnota  $q/p$  je záporná v případě, že je první hodnota  $x$  také záporná, jinak je kladná (neuvažujeme případ, kdy první hodnota  $x$  je 0, protože středem detektoru prochází paprsek a detektorová hladina je v tomto intervalu prázdná).

V první fázi tedy algoritmus pro každý zásah od první po šestou hladinu detektoru (tedy pro hodnoty  $z \in \langle 3, 8 \rangle, z \in \mathbb{N}$ , další hodnoty  $z$  neuvažujeme, protože z nich nelze vytvořit trojice) generujeme první odhad stavového vektoru  $\mathbf{r}_0^+$ . V této práci byl nakonec použit první zmíněný postup, s jednou změnou: Experimentálně bylo zjištěno, že v případě předpokládaného nulového náboje se Kalmanův filtr brzy pokusí aproximovat pouze přímočarými trajektoriemi (tedy ustálit hodnotu  $q/p$  jako 0). Tomuto fenoménu zabráníme tak, že určíme počáteční hodnotu  $q/p$  jako  $\pm 10^{-19}$  v závislosti na tom, jestli první naměřený bod má hodnotu  $x$  kladnou, nebo zápornou.

## Hledání trojic zásahů

Nejdůležitější (a časově nejnáročnější) součástí algoritmu je hledání trojic zásahů, které nejpravděpodobněji patří ke stejné částici. Dr. Akishina ve své práci zmiňuje, že v závislosti na počtu částic, které v rámci daného snímku proletěly detektorem, zabírá fáze hledání trojic až 90.4% celkového výpočetního času [1], a tedy vykazuje největší potenciál pro akceleraci.

Uvedme pro demonstraci této části algoritmu hypotetickou situaci, kdy detektorem prolétly tři částice, které zanechaly perfektní stopy (žádná částice neproletěla detektorem nezaznamenána) a nezanechaly šum. Pro každý bod, pro který hledáme nejvhodnější trojice,

použijeme bod  $[0, 0]$  jako výchozí a počáteční odhad  $\mathbf{r}_0^+$ , který jsme získali v rámci předchozí fáze programu.

V tento okamžik provede algoritmus první Kalmanovskou Filtraci nad počátečním odhadem  $\mathbf{r}_0^+$  a s prvním „měřením“  $\mathbf{m}_1$ . Protože Filtr není lineární, je třeba nejprve získat správnou hodnotu propagační matice  $F_k$ . protože máme sestrojený simulační model, díky kterému dokážeme odsimulovat propagaci částice. Experimentálně bylo zjištěno, že většina částic, které detektorem proletí, je jen velmi málo ovlivněno magnetickým polem a jejich trajektorie je tak víceméně přímá. Z tohoto důvodu příští hodnotu propagační matice  $F_k$  získáme tak, že za pomoci bodu  $\mathbf{m}_1$  a odhadu  $\mathbf{r}_0^+$  inicializujeme Runge-Kuttovu metodu 4. řádu s krokem 10cm, díky čemuž získáme stav příštího odhadu  $\mathbf{r}_1^{lin}$  a dopočítáme podle nich  $F_0$  podle následující rovnice:

$$\mathbf{r}_1^{lin} = \mathbf{r}_0^+ F_0 \quad (5.1)$$

Zde je třeba poznamenat, že přestože k získání nového odhadu  $\mathbf{r}_1^-$  nepotřebujeme linearizovanou propagační matici  $F_1$ , protože jsme ji získali pomocí Runge-Kuttovy metody zmíněné výše, propagační matici potřebujeme také proto, abychom mohli správně upravit kovarianční matici  $C_1^-$  podle rovnice:

$$C_1^- = F_0 C_0^+ F_0^T + Q_0 \quad (5.2)$$

kde  $F_0$  je lineární propagační matice,  $C_0^+$  je původní kovarianční matice a  $Q_0$  je matice procesního šumu, která je pro naše účely nulová, protože simulátor ani rekonstrukční algoritmus se šumem, vyplývajícím z mnohonásobného rozpadu při průchodu detektorovou hladinou nepočítají. V případě, že bychom chtěli pracovat s procesním šumem, který v praxi vzniká vlivem mnohonásobného rozpadu, ke kterému dochází při průchodu částice jednotlivými proužky detektoru, modelovali bychom ho následujícím způsobem: [1]

$$Q_k = \sigma_k^2(\theta) \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (t_x^2 + 1)t_r^2 & t_x t_y t_r^2 & 0 \\ 0 & 0 & t_x t_y t_r^2 & (t_y^2 + 1)t_r^2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (5.3)$$

kde  $\theta$  je úhel dopadu částice na hladinu detektoru.  $\sigma_k^2$  značí směrodatnou odchylku normálního rozložení, určujícího míru změny směru trajektorie v důsledku mnohonásobného rozpadu. Takovou směrodatnou odchylku určíme následujícím způsobem: [1]

$$\sigma(\theta) = \frac{13.6 \text{MeV}}{\beta c p} q \sqrt{L/X_0} [1 + 0.038 \ln(L/X_0)] \quad (5.4)$$

Přestože výpočet směrodatné odchylky pro procesní šum není nemožný, narazíme na hned několik problémů, jejichž řešení na první pohled není zřejmé, a pro naše účely zbytečně komplikované:

- neznáme hodnotu rychlosti částice  $\beta$ , protože námi používaný fyzikální model s rychlostními vektory schválně nepočítá, abychom se vyhnuli problémům s časovou proměnnou v Kalmanově Filtraci.
- nemáme k dispozici materiálové vlastnosti jednotlivých proužků detektoru, což výpočet mnohonásobného rozpadu dále komplikuje.

Z tohoto důvodu bylo od zahrnutí mnohonásobného rozpadu v detektoru jak ze simulačního, tak rekonstrukčního modelu vypuštěno.

Zbytek Kalmanovy filtrace je totožný s postupem zmíněným výše v práci (viz 3.2). Na konci této filtrace má algoritmus k dispozici „dvojice“ bodů, kde první bod je střed terče a druhý bod je výchozí bod pro stavbu trojice zásahů. Tento postup opakujeme tak, že pro každý výchozí bod vyzkoušíme dále Kalmanovu filtraci s každým bodem z následující detektorové hladiny jako další měření. Celý tento proces pak zopakujeme ještě jednou tak, že na konci této fáze programu vznikne  $a * b * c$  možných trojic zásahů, kde  $a$  je počet zásahů na detektorové hladině  $i$ ,  $b$  je celkový počet zásahů na detektorové hladině  $i + 1$  a  $c$  je celkový počet zásahů na detektorové hladině  $i + 2$ . Každá z těchto trojic má v jazyce Julia následující strukturu:

```
struct Hit
    x::Float32
    y::Float32
end

struct Triplet
    point1::Hit
    point2::Hit
    point3::Hit
    chi_sq::Float32
end
```

Zde je dobré poznamenat, že ne všechny kombinace zásahů jsou dále používány. Díky hodnotě  $\chi^2$  je na první pohled zřejmé, že některé trojice (i dvojice) zásahů není možné, aby tvořily dráhu jedné trajektorie. Z tohoto důvodu lze experimentálně nastavit práh, který nesmí hodnota  $\chi^2$  překročit, nebo bude daná dvojice či trojice zahozena a dále s ní není pracováno. Tímto způsobem se vyhneme nutnosti zpracovávat již tak velké množství dat vyloženě kombinatoricky a přispějeme tak k výraznému zrychlení.

## Vytvoření a evoluce celulárního automatu

V okamžiku, kdy program má k dispozici všechna data o možných trojicích zásahů, je možné zkonstruovat celulární automat, díky kterému budeme moci spojit jednotlivé trojice zásahů do kompletních kandidátů na trajektorie.

Narozdíl ale od intuitivního pojetí celulárního automatu, kdy každý pixel reprezentuje jedna buňka automatu a okolí buňky je obdobně složeno z okolních pixelů detektoru, modelujeme celulární automat následujícím způsobem:

- každá buňka automatu reprezentuje možnou trojici zásahů, vygenerovanou předchozím krokem programu. Jakmile se program dostane do této fáze, již jednotlivé zásahy nezpracovává, a pouze skládá nejnadějnější trojice zásahů.
- Okolí jednotlivých buňek určujeme následovně: pokud dvě trojice mají společné dva zásahy (a neliší se prostředním z nich, potom jsou trojice konkurenti) a podobné hodnoty  $q/p$ , jsou označeny jako sousední.

Jakmile vytvoříme síť sousedů, vzniká nám jakási stromová struktura, se kterou pracujeme dále: každé buňce (tj. trojici) přiřadíme „počítadlo“, které počítá nejdelší řetězec

sousedů směrem k terči (tj. určujeme nejdelší kandidáty trajektorií). Ve chvíli, kdy máme inicializované „buňky“ se správně přiřazenými počítadly, můžeme začít vybírat nejlepší kandidáty na částice. Výběr je nyní již jednoduchý: vzhledem k tomu, že jako sousedy trojice uvažujeme pouze tehdy, kdy jsou jejich stavové vektory a hodnoty  $\chi^2$  dostatečně podobné, díky čemuž dokážeme vyloučit většinu náhodných kombinací trojic, v tento okamžik vybíráme nejlepší kandidáty podle dvou kritérií:

- od nejdelší trajektorie po nejkratší: Pravděpodobnost, že náhodná množina zásahů nereprezentuje reálný průlet částice exponenciálně klesá s každým dalším zásahem, který by mohl trajektorii odpovídat. Z tohoto důvodu algoritmus upřednostňuje delší čtrajektorie před kratšími.
- od nejnižší hodnoty  $\chi^2$  po nejvyšší: Čím vyšší hodnota  $\chi^2$ , tím méně sada vybraných zásahů odpovídá chování reálné potenciální částice. Z tohoto důvodu upřednostňuje program kandidáty s nižšími hodnotami  $\chi^2$  před vyššími.
- pokud trojice zásahů již byla v trajektorii dříve použita, podruhé již použita být nesmí. Z tohoto důvodu, než program definitivně reportuje nalezenou trajektorii, zkontroluje, jestli některá z použitých trojic nebyla již použita. Pokud ano, tento kandidát není použit a je zahozen.

## Výběr nejlepších kandidátů a příští evoluce celulárního automatu

Výstupem předchozí fáze programu je seznam kandidátů na „nejnadějnější“ trajektorie, seřazený sestupně podle délky trajektorie a hodnoty  $\chi^2$ , sestupně v tomto pořadí. V této fázi program projde sekvenčně všechny kandidáty a pokud jsou všechny body, které kandidát obsahuje, dosud nepoužity, zapíše je na výstup, v opačném případě je zahodí. Jakmile program takto projde celým seznamem, tato fáze programu je ukončena, použité zásahy jsou ze seznamu odstraněny a celulární automat je inicializován znovu pro další evoluční fázi, tentokrát s jinými parametry. Dále algoritmus probíhá stejným způsobem, tentokrát pro trajektorie s nízkou energií, poté pro sekundární trajektorie a nakonec pro trajektorie, u kterých kvůli nedokonalosti detektoru STS došlo k „vypadnutí“ jednoho zásahu (pravděpodobnost, že v jedné trajektorii vypadne více než jeden zásah je menší než 0.05%, což je vzhledem k vlastnostem detektoru zanedbatelné). Vzhledem ale k tomu, že se v rámci této práce zabýváme pouze primárními trajektoriemi s vysokou energií (a pro tyto účely vytvořený simulátor vytváří pouze takové trajektorie) a další iterace celulárního automatu jsou k té první analogické, dále se jimi nebudeme zabývat.

## 5.2 Rozdíly v implementaci oproti původnímu algoritmu

Jak již bylo zmíněno výše, pro tuto práci vyvinutý algoritmus je oproti původnímu algoritmu z práce Dr. Akishiny [1] zjednodušený tak, aby zachoval podstatu programu spolu s jeho nejnáročnějšími součástmi, byl ale zjednodušen tak, aby se nemusel zabývat pokročilými fyzikálními fenomény, které se jsou z programátorského hlediska sice de facto analogické, ale z fyzikálního hlediska výrazně nad rámec této práce. Jak jsme již zmínili výše, simulátor i rekonstrukční algoritmus vyvinutý v rámci této práce se dopouští zjednodušení zejména v oblasti procesního šumu, který ani simulátor, ani rekonstrukční algoritmus neuvažují, tento fakt ale z důvodu, že je zanedbán již ve fázi generování trajektorií nemá na funkci algoritmu vliv (můžeme říci, že model použitý pro rekonstrukční algoritmus je pouze přesnější).

Další rozdíl je oproti původní implementaci je v omezení implementace pouze na primární trajektorie s vysokou energií, zčásti proto, že jejich rekonstrukce je de facto analogická, ale také proto, že zejména v případě sekundárních trajektorií nastává velký problém při simulaci takových rozpadů, protože takový simulační model by obnášel výrazně složitější model, který vyžaduje hluboké znalosti z oblasti částicové fyziky, což opět přesahuje rámec této práce.

Jako poslední významný rozdíl oproti původní implementaci zmiňme nástroje a technologie při implementaci použité, a technologický kontext práce. V rámci této práce vznikl rekonstrukční program, který jako hlavní způsob akcelerace využívá zejména distribuované počítání, zejména pomocí protokolu MPI<sup>1</sup> a modulu Distributed.jl jazyka Julia, který oproti MPI<sup>2</sup> programátorsky zjednodušuje „jednostranné“ posílání zpráv pro facilitaci distribuovaných výpočtů. Oproti tomu původní algoritmus pro rekonstrukci částic v experimentu CBM byl vytvořen tak, aby co nejlépe využíval platformy *Intel Xeon Phi*. Silnou stránkou této platformy bylo velké množství CPU jader s instrukční sadou rodiny **x86** (Intel Xeon Phi 7295, původně vydaný v roce 2017, disponoval 72 jádry o čtyřech vláknech na jádro, tedy 288 vláken [7]) a podpora 512-bitových SIMD<sup>3</sup> instrukcí AVX-512. Z tohoto důvodu se původní algoritmus zabýval paralelizací na úrovni vláken a vektorizací výpočtů pomocí SIMD instrukcí, které byly silnou stránkou této platformy. Vývoj této platformy byl ale v roce 2019 (částečně kvůli problematické litografické 10nm technologii společnosti Intel a velkou konkurencí ze strany výrobců GPGPU) ukončen a platforma označena jako *End of Life*. [8]

### 5.3 Paralelní rekonstrukční algoritmus

V této podkapitole se budeme podrobně zabývat paralelizací algoritmu, jehož části byly v této práci již podrobně rozebrány. V této práci se zaměříme primárně na paralelizaci náročných částí programu pomocí distribuovaných výpočtů z toho důvodu, že distribuovaný přístup umožňuje nejen paralelizovat výpočty na jednotlivých jádrech procesoru, ale zejména *distribuovat* výpočty mezi různými počítači na síti. Tento přístup je dále podpořen faktem, že datová centra mají zpravidla velmi dobře optimalizovanou infrastrukturu pro distribuované výpočty (dnešní počítače lze například pomocí technologie InfiniBand propojit za přenosových rychlostí až 1200Gb/s s latencí nižší než 600 nanosekund [24]), v důsledku čehož je možné využít velkého množství výpočetního výkonu pro danou aplikaci, což by v extrémních případech, jako je on-line rekonstrukce trajektorií částic v HEP experimentech, často nepraktické kvůli velké režii odesílání dat po síti pomocí protokolů jako MPI.

#### Analýza paralelizovatelných součástí programu

Abychom byli schopni vysoce náročný program efektivně akcelarovat, je nejprve potřeba provést rozbor jeho nejnáročnějších částí. Programovací jazyk Julia pro usnadnění profilování zdrojového kódu interaktivní *plamenové grafy*<sup>4</sup>, graf vykreslující stav zásobníku volání v závislosti na čase. Modul `FlameGraphs.jl` navíc pomáhá nalézt kritické části programu, které mohou způsobovat problémy například z důvodu neudání explicitních typů

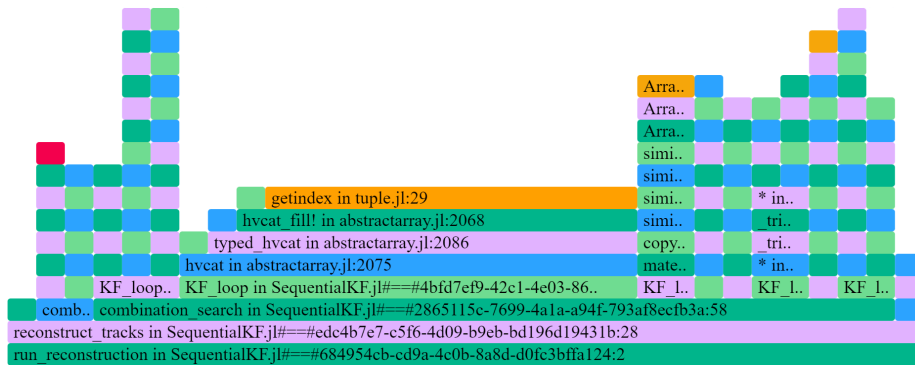
<sup>1</sup>MPI – Message Passing Interface (rozhraní pro předávání zpráv)

<sup>2</sup>Zde rozumíme standard MPI-1, který neposkytuje mechanismy pro jednostrannou komunikaci RMA

<sup>3</sup>SIMD – Single Instruction, Multiple Data (jedna instrukce, více dat)

<sup>4</sup>z angl. flame graphs





Obrázek 5.1: Plamenový graf funkce `run_reconstruction()` nad datasetem `cbm_10_particles`, vygenerovaným simulátorem s relativistickým časovým modelem. Za povšimnutí stojí červeň vybarvená oblast z volání funkce nad datovými rámci, kde jsme se nevyhli problémům s dynamickým typováním, časová ztráta způsobená touto chybou je ale vedle ostatních součástí programu zanedbatelná.

proměnných, což může do značné míry omezit schopnost kompilátoru optimalizovat kód před spuštěním.

Z grafu 5.1 můžeme vyčíst, že zdaleka nejnáročnější součást programu je vlastní smyčka Kalmanova Filtru. Výrazně tak převáží i časovou náročnost evoluce celulárního automatu i ostatní části programu, jako například operace úprav datasetu pro potřeby programu. Zároveň stojí za povšimnutí, že nejnáročnější část funkce není ani samotné maticové násobení, protože (podobně jako v případě vektorizovaných operací nad datasety) samotné maticové výpočty Julia při dodržení základních pravidel správného návrhu velmi dobře optimalizuje a výrazně tak snižuje atraktivitu manuální vektorizace kódu na tomto místě. Protože je ale funkce navržena tak, aby mohla být spouštěna mnohokrát s na sobě vzájemně nezávislými argumenty, je zřejmé, že paralelizovat bude výhodnější na úrovni o jedna vyšší, tedy ve funkci `combination_search()`.

Zároveň je třeba poznamenat, že časová náročnost s počtem zásahů v detektoru stoupá u části vypočítávající odhady Kalmanova Filtru výrazně rychleji, než u ostatních částí programu, jak vyplývá z kombinatorické podstaty algoritmu konstrukce trojic. Dr. Akishina ve své práci zmiňuje, že konstrukce trojic zásahů zabírá přes 90% výpočetního času programu [1], a i když v našem případě tento údaj není tak markantní z důvodů omezení, které blíže popisujeme v podkapitole 4.1 (zejména pak nehomogenní magnetické pole, se kterým současné řešení počítá, výrazně zvyšuje dobu, kterou stráví ve smyčce Kalmanova Filtru, konkrétně pak u propagace  $\mathbf{r}^t$  Runge-Kuttovou metodou), stále se jedná o nejnáročnější součást výpočtu a tedy i o součást, které věnujeme při paralelizaci největší pozornost.

## Paralelizace rekonstrukčního programu

V předchozí kapitole jsme zjistili, že největší potenciál pro zrychlení má u rekonstrukčního programu smyčka Kalmanova Filtru. Tento fakt nám radí, že pro vhodnou paralelizaci potřebujeme najít způsob, jakým rozdělit „stavový prostor“ problému tak, abychom minimalizovali množství dat, se kterými jednotlivé uzly bezpodmínečně potřebují pracovat. V případě paralelizace na úrovni vláken nebo logických procesorů by nám pomohla možnost pracovat se sdílenou pamětí, ať už v podobě vyrovnávací paměti *cache* nebo RAM. V případě distribuovaného počítání je ale nutné se takovému paradigmatu pokud možno vy-

```

BenchmarkTools.Trial: 1880 samples with 10 evaluations.
Range (min ... max):  1.816 ms ... 16.123 ms
Time (median):        2.085 ms
Time (mean ± sig):    2.655 ms ± 1.880 ms
GC (min ... max):    0.00% ... 79.34%
GC (median):         0.00%
GC (mean ± sig):     9.32% ± 11.36%
Memory estimate: 2.64 MiB, allocs estimate: 29229.

```

Obrázek 5.2: Výstup benchmarku nástroje `BenchmarkTools` jazyka Julia pro sekvenční implementaci rekonstrukčního algoritmu. Jako testovací data byl použit dataset `cbm_3_particles`, vygenerovaný simulátorem s relativistickým časovým modelem. Modul `BenchmarkTools` v rámci makra `@benchmark` vyhodnotí rychlost provádění měřené funkce a vygeneruje statistiky a histogram jednotlivých iterací.

hnout: jednotlivé uzly (většinou, existují výjimky, viz [24]) nesdílí paměť mezi sebou, datová komunikace probíhá pomocí zpráv, které jsou oproti komunikaci mezi např. jednotlivými jádry procesoru velmi pomalé, a tedy drahé. Z tohoto důvodu je žádoucí, aby každý uzel v rámci výpočetního *clusteru* fungoval pokud možno co nejvíce samostatně a teprve na konci své práce odevzdal vedoucímu uzlu výsledek.

V rámci bližšího zkoumání implementovaného sekvenčního algoritmu byly identifikovány čtyři různé body, potenciálně vhodné k paralelizaci:

- Dva v rámci funkce `combination_search()` (dvě náročné, vnořené `for` smyčky) a
- Dvě o stupeň v zásobníku volání níž ve funkci `reconstruct_tracks` vnořené `for` smyčky.

V rámci hledání optimálního řešení byly tedy implementovány všechny čtyři potenciální paralelizace. Pro paralelizaci na bázi distribuovaného počítání bylo třeba provést následující kroky:

1. **Načtení modulu `Distributed`:** Než nám bude umožněno jakkoli manipulovat s dalšími procesy, musíme nejdříve v mateřském procesu načíst tento modul. Tento modul nám poskytne dvě extrémně pro nás důležitá makra: `@everywhere`, které akci, kterou chceme provést, vynutí na všech uzlech *clusteru*, a `@distributed`, makro, které nám umožní bez velkých zásahů do kódu náročné výpočty distribuovat.
2. **Vytvoření nových procesů:** Pokud jsme nespustili prostředí Julia pomocí přepínače `julia -p N`, kde `N` značí počet procesů, které se k tomu hlavnímu vytvoří **navíc**, je třeba tak učinit nyní. Učiníme tak prostě pomocí funkce `addprocs(N)`, kde `N` opět značí počet dalších procesů, které je třeba vytvořit (zde sice hovoříme o procesech, protože je instanciujeme na stejném počítači, implementace `Distributed` ale počítá i s komunikací jednotlivých uzlů po síti).
3. **Nastavení prostředí nových procesů:** Julia při vytváření nového procesu nekopíruje proměnné prostředí, proces je tak inicializován *de facto* prázdný. Proto je potřeba

dceřinným procesům předat všechny důležité informace předtím, než program spustíme a začneme měřit jeho výkon. Zde nám pomůže makro `@everywhere`, díky kterému standardní operace pro načtení knihovny `include("CBMReconstructor.jl")` nemusí být nijak speciálně upravena ani nemusíme nijak měnit strukturu původního programu, makro problém nových uzlů vyřeší za nás. Kromě knihovny `CBMReconstructor` je také potřeba informovat uzly, aby načetly modul `DistributedArrays`. Tento modul řeší problém, kdy jednotlivé procesy potřebují pracovat nad stejnou proměnnou (například ukládat struktury do stejného pole).

- 4. Paralelizace náročné smyčky pomocí `@distributed`:** Po tom, co jsme úspěšně inicializovali cluster a identifikovali nejperspektivnější části programu pro paralelizaci, vlastní rozdělení práce mezi uzly je opět triviální. Pomocí makra `@distributed` program rozdělí všem uzlům práci a implicitně pokračuje ve vykonávání programu (pokud nepoužijeme explicitně makro vynucující synchronizaci, což pro naše potřeby není nutné), dokud nepotřebuje pracovat s daty, o kterých ví, že zpracovává jiný proces a dosud ji nevrátil (v takovém případě uzel čeká).
- 5. Vrácení výsledků paralelních bloků a změření rychlosti programu:** V okamžiku, kdy ukončíme provádění paralelní části programu, provedeme zbytek programu stejně, jako v případě sekvenční verze programu. V zájmu dalšího zrychlení projektu byl navržen algoritmus pro paralelní/distribuované řazení vygenerovaných kandidátních trajektorií. Navržena byla implementace distribuovaného *merge sortu*, po profilování programu a vygenerování plamenového grafu bylo ale od pokusu implementovat distribuovaný *merge sort* upuštěno ze dvou důvodů: Zaprvé Julia implementuje velmi dobře optimalizovanou *in-place* funkci, zadruhé u funkce, která není prováděna dostatečně dlouho na to, aby se paralelizace vyplatila, nevhodné vynucení distribuovaného počítání mohou rychlost zpracování naopak dramaticky zpomalit.

Po tom, co jsme úspěšně vytvořili snadno distribuovatelný program, zbývá jen změřit a vyhodnotit výsledky a zjistit tak, která z možností paralelizace je nejvýhodnější.

## Porovnání výkonu paralelních algoritmů

Z tabulky 5.3 můžeme vyčíst několik zajímavých faktů. Zaprvé, jak vyplývá z plamenového grafu, jednotlivé cykly, které jsme se snažili paralelizovat, jsou vnořené jedna do druhé (i když je tento fakt zakryt faktem, že se objevují v různých funkcích. Zatímco sekvenční algoritmus je neefektivní a pro praktické využití nepoužitelný kvůli své vysoké náročnosti na paměť a malé rychlosti, pokus paralelizovat „nejvnitřnější“ smyčku z těchto tří způsobí, že nedostatek výpočetně náročných instrukcí, které by použití distribuovaného počítání ospravedlnilo a relativně vysoká režie posílání zpráv jiným uzlům způsobil, že program se s vyšším počtem jader stává výrazně pomalejším a „hladovějším“ po paměti. Není třeba zdůrazňovat, že tento postup není vhodný.

Vnější cyklus v téže funkci naproti tomu už se chová daleko víc, jak bychom si představovali, s výrazně nižšími nároky na paměť a až čtyřnásobné zrychlení oproti sekvenčnímu algoritmu. Nicméně stojí za zmínku, že ve chvíli, kdy uzel přestane být plně využíván, rychle přestává být užitečný. V případě vnější smyčky funkce `combination_search()` takový jev sledujeme u měření s dvaceti čtyřmi logickými procesory: řádově vyšší spotřeba paměti a nižší rychlost, než kdybychom použili pouze šest jader. Znovu se tedy prokazuje, že distribuované počítání je výhodné pouze tehdy, máme-li dobrý důvod ho použít.

		Proc memory (KB)	allocations	median_time
sequential_search	1	7730000	83969879	8,201
inner_combination_search_2	2	890000	13274641	19,909
inner_combination_search_6	6	1670000	26762989	28,965
inner_combination_search_12	12	3380000	55330367	43,656
outer_combination_search_6	6	26300	478466	2,365
outer_combination_search_8	8	43880	708563	2,3
outer_combination_search_24	24	147090	2418930	2,429
outer_reconstruct_tracks_8	8	127,28	2233	1,467
outer_reconstruct_tracks_12	12	167,77	2804	1,756
inner_reconstruct_tracks_2	2	223,53	3021	6,04
inner_reconstruct_tracks_4	4	368	5729	2,574
inner_reconstruct_tracks_8	8	692,53	11827	1,582
inner_reconstruct_tracks_12	12	1050	18828	1,298
inner_reconstruct_tracks_16	16	1752	30028	1,257
inner_reconstruct_tracks_24	24	2200	41316	1,235

Obrázek 5.3: Tabulka měření jednotlivých verzí distribuovaného rekonstrukčního programu. Z dat můžeme vyvést závěr, že distribuce vnitřní smyčky funkce pro rekonstrukci trajektorie je pro zrychlení (a šetření systémovou pamětí). Zatímco některé z implementačních verzí se dle očekávání zrychlovaly, jiné naopak výrazně na výkonu ztrácely i vůči naivní, sekvenční verzi programu.

Vnější cyklus vnější funkce (tedy „nejvnějšejší“ cyklus) je další dobrý příklad této teorie: Protože je v této smyčce přesně dané, že bude iterovat pouze v intervalu  $[3 : 8]$ . Z toho vyplývá, proč se zvyšujícím se množstvím dostupných jader se výkon algoritmu zhoršuje. Za povšimnutí každopádně stojí velmi nízký počet alokací v programu, což je šetrné k paměti.

Poslední (a nejlepší) varianta řešení distribuce výpočtu rekonstrukčního algoritmu je vnitřní smyčka vnější funkce. Z důvodů, které jsme zmínili výše, byla tato varianta jediná, která při spuštění na plných 24 logických procesorech byla schopná pojmout plný výkon procesoru, díky čemuž poskytuje nejvyšší zrychlení téměř 800%. Opět ale stojí za zmínku, že přestože má s vyšším počtem jader stále vyšší výkon, neznamená to, že můžeme paralelizovat „donekonečna“: již na rozdílu mezi časem naměřeným pro 16 a 24 jader je vidět, že s vyšším počtem jader je každé další jádro pro výpočet méně přínosné.

## 5.4 Alternativy k použití distribuovaného počítání

V raných fázích této fáze bylo uvažováno nad akcelerací zmiňovaného algoritmu na platformě FPGA. Tento přístup se ale ukázal být nepraktický z několika důvodů, ze kterých zmiňme dva nejmarkantnější:

- **Paměťová náročnost.** Jeden z hlavních problémů rekonstrukčního programu je jeho paměťová náročnost. Již při rekonstrukci 100 částic pracuje program s vysokými desítkami gigabajtů dat, což způsobuje problém, kdy provoz potenciálního FPGA akcelérátoru vyžaduje časté I/O (jakýsi „page miss“) operace, které dále zpomalují výpočet. Tento problém platformy FPGA je těžké odstranit a uvážíme-li, že reálný program je

kvůli přítomnosti nehomogenního magnetického pole ještě výrazně paměťově náročnější, alternativa FPGA přestává být atraktivní.

- **Prostorová náročnost.** Z podstaty maticového násobení, které je fundamentální pro rekonstrukční algoritmy založené na Kalmanově Filtru, vyplývá, že jeho implementace na hradlových polích by byla velmi náročná. Při násobení matic bychom pro každý výpočet matic  $5 \times 5$  potřebovali 125 násobiček a neexistuje způsob, jak takové množství zmenšit. Když navíc uvažíme, že takových násobení je **v každé iteraci** Kalmanova Filtru dvanáct, začne být zřejmé, že jakýkoli vysoce specializovaný obvod bude zabírat tolik místa na hradlovém poli, že je jeho implementace nepraktická. Navíc, specializované akcelerátory jako *GPGPU*<sup>5</sup>, poskytují vysoký výkon pro maticové operace (zejména pak násobení a sčítání) a relativně mírnou učitelskou křivku pro jejich programování.

Stejně jako algoritmus zvolený pro tuto práci je současné řešení rekonstrukce trajektorií postavené na konceptech celulárního automatu a rozšířeného Kalmanova filtru. Konkrétně řešení navržené pro experiment CBM bylo navrženo pro platformu *Intel Xeon Phi* a využívá paralelního zpracování pomocí vláknové paralelizace a vektorizace SIMD instrukcemi. Tento postup má oproti akceleraci na FPGA velkou výhodu, a tou je jeho **operační paměť**: Největší zřejmá slabina hradlových polí v této situaci je velmi malé množství operační paměti.

Naopak velmi perspektivní je v tomto odvětví použití grafických procesorů. Z podstaty Kalmanova Filtru vyplývá, že využívá zejména maticového sčítání a násobení, dvě operace, na které jsou moderní grafické procesory velmi dobře vybaveny. V současné době v Evropském Centru pro Jaderný Výzkum (CERN) pracuje experiment ALICE<sup>6</sup>, jeden ze čtyř velkých experimentů na Velkém Hadronovém Urychlovači (LHC) a experiment, jehož výpočetní přístup je tomu v CBM velmi blízko (implementace na bázi celulárního automatu a Kalmanova Filtru). Rozdíl oproti CBM je v tom, že výpočty při ALICE jsou akcelerovány pomocí GPGPU [25] a vzhledem k tomu, že v rámci této práce nebylo optimalizačně nijak zasahováno do algoritmu Kalmanova Filtru nabízí otázku, zda by nebylo možné akcelarovat tento výpočet dále pomocí kombinace distribuovaných výpočtů a urychlení `KF_loop` na GPU.

---

<sup>5</sup>GPGPU – General Purpose Graphics Processing Unit (obecný grafický akcelerátor)

<sup>6</sup>ALICE – A Large Heavy Ion Experiment (velký experiment s těžkými ionty)

# Kapitola 6

## Závěr

Cílem této práce bylo seznámit se s moderními experimenty na půdě částicové a kvantové fyziky, podrobně prostudovat použité technologie a algoritmy využívané pro rekonstrukci trajektorií v detektorech částic, navrhnout a implementovat akcelerovatelný rekonstrukční algoritmus a tuto akceleraci následně také implementovat. Vzhledem ke rozsáhlosti tématu a matematicko-fyzikálnímu aparátu mimo rámec informačních technologií si tato práce dává také za cíl problematiku rekonstrukce částic v moderních fyzikálních experimentech co nejlépe osvětlit..

V rámci práce jsem se podrobně seznámil s jednotlivými fázemi zpracování dat z detektorů částic, různými technologiemi pro získávání dat z nich a algoritmy pro následnou rekonstrukci trajektorií částic, zejména pak pro detektor CBM při výzkumném středisku FAIR v Darmstadtu. Pro podrobnější studium jsem zvolil dvě výpočetně nejnáročnější (a tedy nevhodnější pro studium možností jejich akcelerace) části zpracování fyzikálních dat z detektorů: rekonstrukce trajektorií částic a odhad (*fitování*) vlastností takto rekonstruovaných trajektorií. Pro akceleraci těchto dvou fází jsem vybral rekonstrukční algoritmus na bázi celulárních automatů a Kalmanových filtrů a po prostudování stávajících řešení pro detektor CBM a detektory mu konceptuálně podobné jsem se rozhodl podrobně zabývat možnostmi akcelerace algoritmu pomocí distribuovaného počítání.

Vzhledem k neexistenci vhodných testovacích dat (ať už reálných, či simulovaných) bylo třeba vhodná testovací data vytvořit. Z tohoto důvodu byly v průběhu práce pro různé účely postupně navrženy a implementovány čtyři různé verze simulačního modelu chování částic uvnitř detektorů se silným magnetickým polem. Více modelů bylo implementováno z důvodu, že samotný sledovací algoritmus potřebuje používat simulační model pro výpočet predikčního kroku rozšířeného Kalmanova filtru, a bylo tedy potřeba simulační model navrhnout tak, aby potřebám algoritmu co nejlépe odpovídal. Z tohoto důvodu jsem v různých fázích studia rekonstrukčního algoritmu průběžně podobu modelu upravoval tak, aby nejenže vhodně simuloval dráhy částic v detektoru, ale aby umožňoval i „částečnou“ simulaci (tj. aby bylo možné simulovat pouze zadaný úsek dráhy částice). Navzdory snaze sjednotit fyzikální modely tak, aby simulace trajektorií částic i propagační model Kalmanova Filtru fungovaly na stejném matematickém modelu, bylo od této snahy zdůvodu nestability q/p modelu při simulaci ostrých úhlů upuštěno a pro potřeby práce jsou používány dva.

V rámci studia rekonstrukčních algoritmů byl algoritmus implementován v jazyce Julia, nejdříve jako čistě sekvenční program a následně ve čtyřech verzích paralelizován pomocí distribuovaného počítání. Pro dataset nedokonalých částic s chybovým detektorem program oproti sekvenčnímu vykonává výpočet na 24 logických jádrech s téměř osmisetprocentním zrychlením. Jako další pokračování práce se nabízí implementace matice procesního šumu

pro výpočet mnohonásobného rozpadu, prozkoumání možností dalšího zrychlení akcelerací distribuovaného rekonstrukčního algoritmu a Kalmanova Filtru za pomoci GPGPU, srovnání výkonu rekonstrukčního kódu v Jazyce Julia vůči například C/C++, popřípadě další zdokonalení vlastního algoritmu, ať už zahrnutím nehomogenního magnetického pole, dělení detektorových stanic podle počtu zásahů nebo implementace rekonstrukce nově vzniknuvších nebo rozbitých částic (s chybějícím zásahem na jedné detektorové stanici).

# Literatura

- [1] AKISHINA, V. *Four-dimensional event reconstruction in the CBM experiment*. 2016. Disertační práce. Johann Wolfgang Goethe-Universität, Fachbereich Informatik und Mathematik. Dostupné z: [inis.iaea.org/search/search.aspx?orig\\_q=RN:49093434](https://inis.iaea.org/search/search.aspx?orig_q=RN:49093434).
- [2] DUNCAN, A. *The analytic impulse* [online]. Květen 1988 [cit. 2022-7-20]. Dostupné z: [www.andrewduncan.net/air/](http://www.andrewduncan.net/air/).
- [3] AUMANN, T. Prospects of nuclear structure at the future GSI accelerators. *Progress in Particle and Nuclear Physics*. 2007, sv. 59, č. 1, s. 3–21. DOI: [doi.org/10.1016/j.pnpnp.2006.12.018](https://doi.org/10.1016/j.pnpnp.2006.12.018). ISSN 0146-6410. International Workshop on Nuclear Physics 28th Course. Dostupné z: [www.sciencedirect.com/science/article/pii/S0146641006000974](http://www.sciencedirect.com/science/article/pii/S0146641006000974).
- [4] BEZANSON, J., EDELMAN, A., KARPINSKI, S. a SHAH, V. B. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*. 2017, sv. 59, č. 1, s. 65–98. DOI: [10.1137/141000671](https://doi.org/10.1137/141000671). Dostupné z: [doi.org/10.1137/141000671](https://doi.org/10.1137/141000671).
- [5] *Program for Bisection Method* [online], 06. dubna 2021 [cit. 2022-7-27]. Dostupné z: [www.geeksforgeeks.org/program-for-bisection-method](http://www.geeksforgeeks.org/program-for-bisection-method).
- [6] CURRY, E., GUYON, B., SHERIDAN, C. a DONNELLAN, B. Developing an Sustainable IT Capability: Lessons From Intel’s Journey. *MIS Quarterly Executive*. Červen 2012, sv. 11, s. 61–74.
- [7] CUTRESS, I. *Intel Lists Knights Mill Xeon Phi on ARK: Up to 72 cores at 320W with QFMA and VNNI* [online]. AnandTech, 19. prosince 2017 [cit. 2023-04-04]. Dostupné z: <https://www.anandtech.com/show/12172/intel-lists-knights-mill-xeon-phi-on-ark-up-to-72-cores-at-320w-with-qfma-and-vnni>.
- [8] CUTRESS, I. a SHILOV, A. *The Larrabee Chapter Closes: Intel’s Final Xeon Phi Processors Now in EOL* [online]. AnandTech, 07. května 2019 [cit. 2023-04-04]. Dostupné z: <https://www.anandtech.com/show/14305/intel-xeon-phi-knights-mill-now-eol>.
- [9] *Desmos Graphing Calculator* [online]. 2015 [cit. 2022-10-07]. Dostupné z: [www.desmos.com](http://www.desmos.com).
- [10] DUDA, R. O. a HART, P. E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. jan 1972, sv. 15, č. 1, s. 11–15. DOI: [10.1145/361237.361242](https://doi.org/10.1145/361237.361242). ISSN 0001-0782. Dostupné z: [doi.org/10.1145/361237.361242](https://doi.org/10.1145/361237.361242).



- [11] *The accelerator facility of FAIR and GSI* [online]. Facility for Antiproton and Ion Research in Europe [cit. 2022-17-04]. Dostupné z: [www.fair-center.eu/overview/accelerator](http://www.fair-center.eu/overview/accelerator).
- [12] *The universe in the lab* [online]. Facility for Antiproton and Ion Research in Europe [cit. 2022-17-04]. Dostupné z: [www.fair-center.eu/overview](http://www.fair-center.eu/overview).
- [13] FAJMON, B., HLAVIČKOVÁ, I., NOVÁK, M. a VÍTOVEC, J. *Numerická matematika a pravděpodobnost: Informační technologie* [online]. VUT v Brně, 2014 [cit. 2022-17-07]. Dostupné z: [matika.umat.feec.vutbr.cz/inovace/texty/INM/CZ/INM\\_plna\\_verze\\_CZ.pdf](http://matika.umat.feec.vutbr.cz/inovace/texty/INM/CZ/INM_plna_verze_CZ.pdf).
- [14] FRIMAN, B., HÖHNE, C., KNOLL, J., LEUPOLD, S., RANDRUP, J. et al. *The CBM Physics Book: Compressed Baryonic Matter in Laboratory Experiments*. Leden 2011. ISBN 978-3-642-13292-6.
- [15] GARDNER, M. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*. 1970, sv. 223, s. 120–123, [cit. 2022-10-07]. Dostupné z: [web.stanford.edu/class/sts145/Library/life.pdf](http://web.stanford.edu/class/sts145/Library/life.pdf).
- [16] GORBUNOV, S., KEBSCHULL, U., KISEL, I., LINDENSTRUTH, V. a MÜLLER, W. Fast SIMDized Kalman filter based track fit. *Computer Physics Communications*. 2008, sv. 178, č. 5, s. 374–383. DOI: [doi.org/10.1016/j.cpc.2007.10.001](https://doi.org/10.1016/j.cpc.2007.10.001). ISSN 0010-4655. Dostupné z: [www.sciencedirect.com/science/article/pii/S0010465507004262](http://www.sciencedirect.com/science/article/pii/S0010465507004262).
- [17] *Green IT Cube becomes research and transfer center* [online], 25. dubna 2022 [cit. 2022-2-07]. Dostupné z: [www.gsi.de/en/start/news/details?tx\\_news\\_pi1%5Baction%5D=detail&tx\\_news\\_pi1%5Bcontroller%5D=News&tx\\_news\\_pi1%5Bnews%5D=5343&cHash=2b65707d6cffc3d40119b25ea374574e](http://www.gsi.de/en/start/news/details?tx_news_pi1%5Baction%5D=detail&tx_news_pi1%5Bcontroller%5D=News&tx_news_pi1%5Bnews%5D=5343&cHash=2b65707d6cffc3d40119b25ea374574e).
- [18] *Green IT Cube: supercomputing center for GSI and FAIR* [online], 01. června 2021 [cit. 2022-2-07]. Dostupné z: [www.gsi.de/en/researchaccelerators/research\\_an\\_overview/green-it-cube](http://www.gsi.de/en/researchaccelerators/research_an_overview/green-it-cube).
- [19] *GSI - About us* [online]. GSI Helmholtzzentrum für Schwerionenforschung [cit. 2022-17-04]. Dostupné z: [www.gsi.de/en/about\\_us](http://www.gsi.de/en/about_us).
- [20] HOUGH, P. V. C. Machine Analysis of Bubble Chamber Pictures. *Conf. Proc. C*. 1959, sv. 590914, s. 554–558. Dostupné z: [inspirehep.net/files/53d80b0393096ba4afe34f5b65152090](http://inspirehep.net/files/53d80b0393096ba4afe34f5b65152090).
- [21] KABEL, C. *Großbaustelle in Darmstadt: Rohbau des Fair-Tunnels ist fertig* [online]. Frankfurter Rundschau, 7. června 2021. Aktualizováno 10.6.2021 [cit. 2022-18-04]. Dostupné z: [www.fr.de/rhein-main/darmstadt/grossbaustelle-in-darmstadt-rohbau-des-fair-tunnels-ist-fertig-90792726.html](http://www.fr.de/rhein-main/darmstadt/grossbaustelle-in-darmstadt-rohbau-des-fair-tunnels-ist-fertig-90792726.html).
- [22] KARI, J. *Cellular Automata*. University of Turku, 2013. 133 s. Dostupné z: [www.cs.tau.ac.il/~nachumd/models/CA.pdf](http://www.cs.tau.ac.il/~nachumd/models/CA.pdf).
- [23] KOLLEGGER, T. *Green Cube GSI: FAIR Tier 0* [online], 30. března 2015 [cit. 2022-27-04]. Dostupné z: [indico.cern.ch/event/325439/contributions/756690/attachments/631354/868855/20150330-TKollegger-GreenCubeGSI.pdf](http://indico.cern.ch/event/325439/contributions/756690/attachments/631354/868855/20150330-TKollegger-GreenCubeGSI.pdf).

- [24] *Mellanox ConnectX-6 VPI Adapter Card: Product brief* [online]. 2016. Dostupné z: [https://www.chipict.com/wp-content/uploads/2018/09/PB\\_ConnectX-6\\_VPI\\_Card.pdf](https://www.chipict.com/wp-content/uploads/2018/09/PB_ConnectX-6_VPI_Card.pdf).
- [25] ROHR, D., GORBUNOV, S. a LINDENSTRUTH, V. GPU-accelerated track reconstruction in the ALICE High Level Trigger. GPU-accelerated track reconstruction in the ALICE High Level Trigger. *J. Phys.: Conf. Ser.* 2017, sv. 898, č. 3, s. 032030. DOI: 10.1088/1742-6596/898/3/032030. 8 pages, 7 figures, contribution to CHEP 2016 conference. Dostupné z: <http://cds.cern.ch/record/2297452>.
- [26] SCHMIEDEL, B. *Runge-Kutta 4th order scheme* [online]. Read the Docs, Inc, 2019 [cit. 2022-07-24]. Dostupné z: [lowebms.readthedocs.io/en/latest/code/rk4.html](https://lowebms.readthedocs.io/en/latest/code/rk4.html).
- [27] TOQUICA CÁCERES, H. Hough and Watershed Transforms Algorithms Brief Review. Březen 2017, s. 1–11. DOI: 10.13140/RG.2.2.35718.98889. Dostupné z: [www.researchgate.net/publication/315495322\\_Hough\\_and\\_Watershed\\_Transforms\\_Algorithms\\_Brief\\_Review](https://www.researchgate.net/publication/315495322_Hough_and_Watershed_Transforms_Algorithms_Brief_Review).