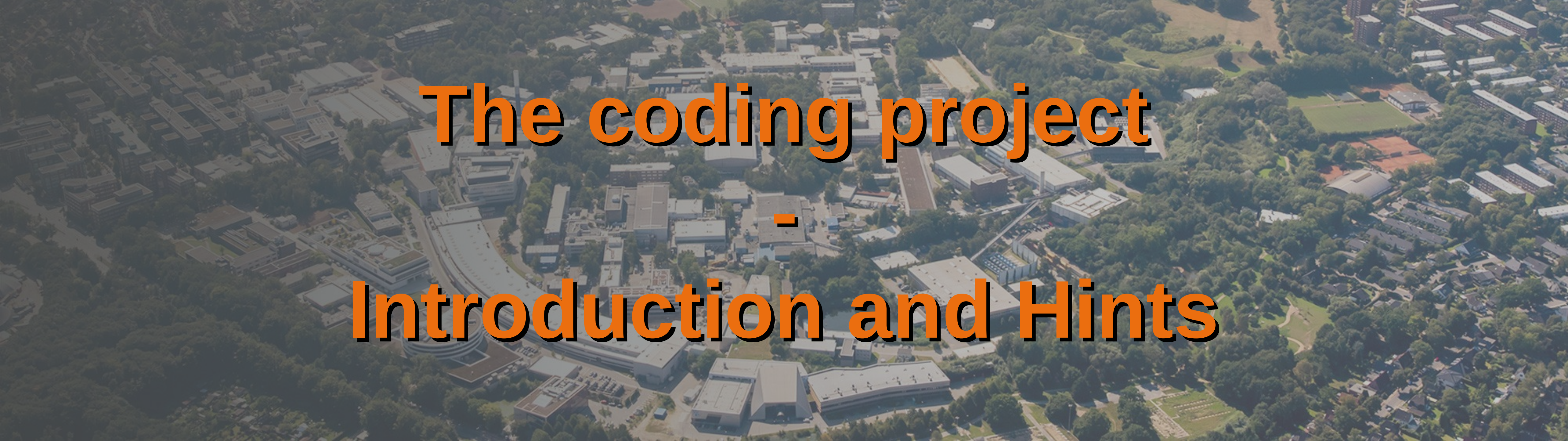


The coding project - Introduction and Hints

Christian Wessel
DESY, 23.05.2023

christian.wessel@desy.de

Introduction to the basf2

scons

- For handling the complex dependencies with the externals (ROOT etc) but also internally, we use sconscripts to compile basf2
- To do so, call “scons -j N” where N is the number of parallel processes used to compile basf2
- You can add options like
  - "--no-debuginfo”: removes additional debugging info from the compiled code
    - Pro: reduces size of the final software
    - Con: if there is an error, you won’t have a fun time to identify it
  - “--sphinx”: builds the sphinx documentation, can be found in the build/html subdirectory
  - “--light”: builds a light release, i.e. only a reduced set of packages is used (much faster)
  - Several others, check the documentation

Introduction to the basf2

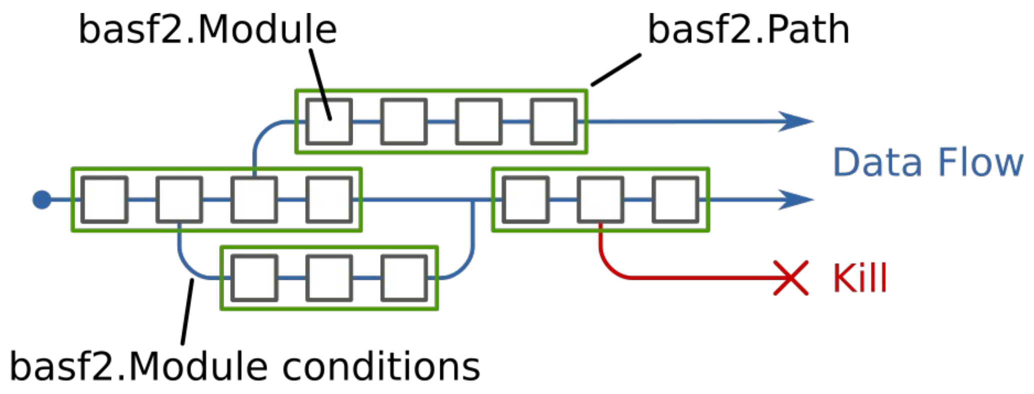
What is a module?

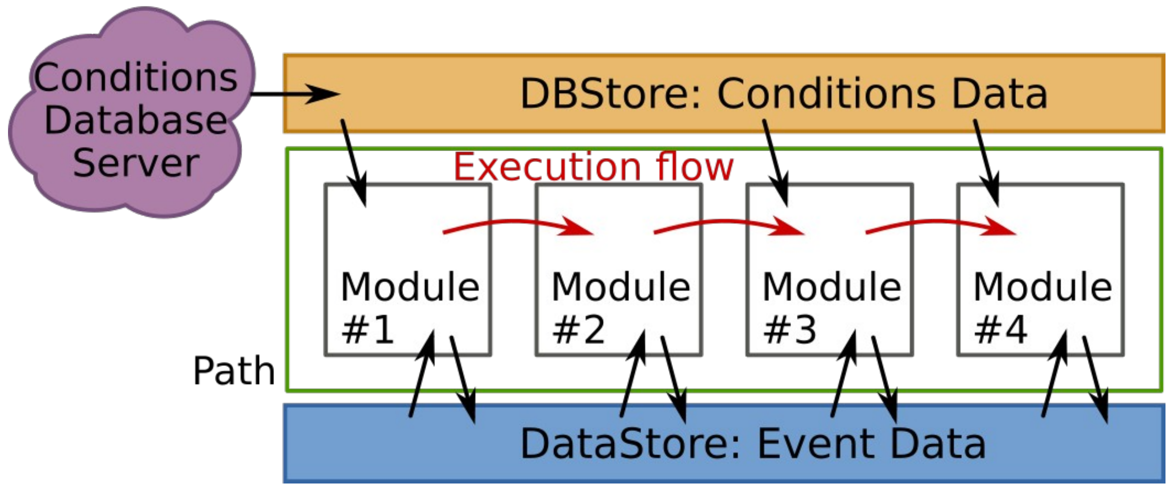
- basf2 is a modular software
- The “base unit” thus is a Module
- We have the Module base class, and all other modules are derived from this
- Functions defined in the base class are
  - initialize(), beginRun(), event(), endRun(), terminate()
  - You don’t have to use all of them if you write your own Module
- As with other classes, you can define additional functions and member variables
- Each module should serve one specified task
  - Don’t try to implement e.g. a full tracking software or ECL reconstruction in one Module, but rather in several
- To add a parameter that you can access from Python, use the “addParam” method in initialize
  addParam(“nameSeenInPython”, m_thisMemberVariable, “Variable description”, defaultValue);
- Due to modularisation, Modules can easily be extended, fixed, replaced in a path

Introduction to the basf2

What is a path?

- basf2 is a modular software
- The “base unit” is a Module
- We stitch together the individual Modules in a path
- The Modules are executed in the order they are in the path
- In a path, Modules can retrieve information from the conditions database
- And they can read from and write to the DataStore
  - Information like detector hits, tracks, clusters
- But how does the DataStore work?

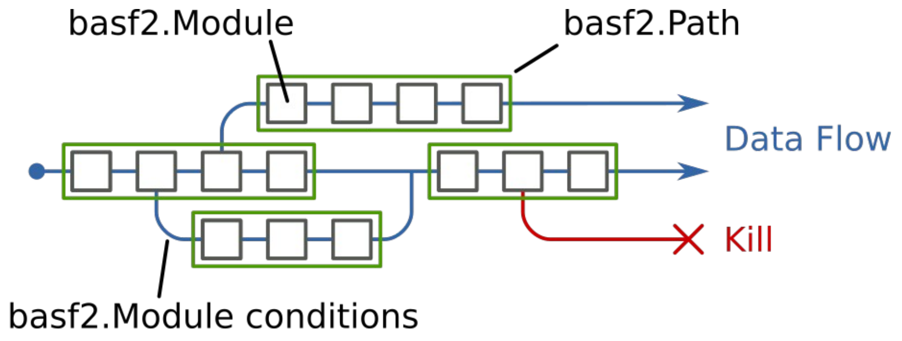


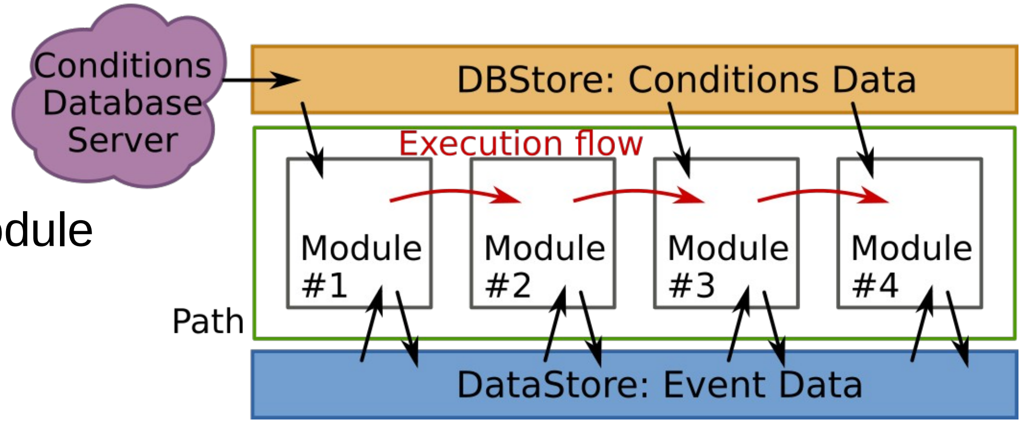


Introduction to the basf2

What is the DataStore?

- The DataStore contains StoreArrays or StoreObjPtr of dataobjects
- These information are event level data
  - Detector hits for each detector
  - Tracks and TrackFitResults
  - Clusters (ECL, KLM)
  - …
- The StoreArrays / StoreObjPtrs are used to pass information between the various Modules who perform actions on them
  - E.g. StoreArray<AwesomeDigit> –> AwesomeClusterizerModule –> StoreArray<AwesomeClusters> –> …





Introduction to the basf2

StoreArrays: dos and don’ts

- Do:
  - If a Module introduces a new StoreArray, it has to register it: someStoreArray.registerInDataStore()
  - If a Module requires a StoreArray, it also has to say so: someOtherStoreArray.isRequired()
  - If you want to add an entry to a StoreArray, you have to call “.appendNew(TheNewObject)”
  - This returns a pointer to the newly created object
  - You can loop over a StoreArray with range based for loops (see Giacomo’s slides)
- Don’t:
  - If you just need a temporary container inside an event in your Module that shouldn’t be seen and used outside of your Module, don’t create a StoreArray but e.g. use a std::vector instead

Introduction to the basf2

Logging in basf2 link 1, link 2

- We have different logging and debugging methods build into basf2
- For warnings or errors, use B2WARNING or B2ERROR
- If something happens so that basf2 must stop execution, use B2FATAL
- For just printing some information, use B2INFO
- And for debugging, use B2DEBUG
  - There are different debug levels for different parts of the software
  - In increasing order: MC generators, simulation, reconstruction, analysis level
- It is possible to set a global logging level to e.g. only see B2ERRORs and B2FATALs but suppress everything else
  - Relevant e.g. on the HLT, where we don't want to see every info, but only things that are relevant to operating the detector and making sure that data quality is on a high level

Introduction to the basf2

If in doubt: Check our documentation!

- www.software.belle2.org
- Use one of the recommended sphinx documentations if you want to learn about the software in general, how to use analysis functionalities, etc
- Use the recommended doxygen documentation if you want to learn about the C++ code

- For more general questions on Python, C++, … use google, cppreference.com, cplusplus.com, Python (package) documentation, …

Introduction to the code of the project

Introduction to the code

AcademyTrackingModule

- In the code base you downloaded, you will find a basf2 module skeleton in the folder tracking/modules/AcademyTracking
- In that folder you will find
  - SConscript –> Just ignore it for the project
  - include: folder that contains the header file for the module
  - src: folder that contains the actual implementation

Introduction to the code

AcademyTrackingModule

- The skeleton contains most of the functions that you need
  - Constructor / destructor
  - Functions of the basf2 Module class: initialize, event, terminate (we don’t need beginRun and endRun)
  - Some definitions of important variables:
    - input and output StoreArrays, variables that you will need, …
  - Helper functions for debugging
    - Writing a Python script to create the Hough Space plots
    - Cleaning up the last event
    - Root histograms as examples
  - Checking the number of layers of hits in a Hough Space sector
  - Resolving overlaps between track candidates
  - Writing the final RecoTracks
  - A few (mostly) empty functions for you to fill

Hough Tracking – Reminder

How it works

- Interchanges variables and parameters
- Lines get to be represented by points, and points are represented by lines
- Finding points = finding intersections of lines
- Use Hesse description of a 2D line $\rho(\theta) = x \cdot \cos\theta + y \cdot \sin\theta$
- Convert circles (= tracks) into straight lines using a conformal transformation

$$(x', y') = \frac{2(x, y)}{(x - x_0)^2 + (y - y_0)^2}$$

- In our case, we can savely assume x_0, y_0 to be 0, which simplifies the formula
  - Reminder: ρ represents the curvature of the track, i.e. 1 / radius –> directly linked to p_T
- Binary 2D search using a divide and conquer approach to find intersections of sinusoidals in Hough Space

Your first task – an easy starter

Implement the conformal transformation

`void AcademyTrackingModule::conformalTransformation()`

- **`TODO 1`**`: actually calculate xConformal and yConformal`

- `Compile the code using scons -j4`
- `Run` **`ParticleGunStarter.py`** `to check whether you actually get a Python script called` **`plotHoughTrafo.py`** `containing information about the hits`
- `Run the Python script` **`plotHoughTrafo.py`** `and check the output (HoughTrafoDebug.pdf)`

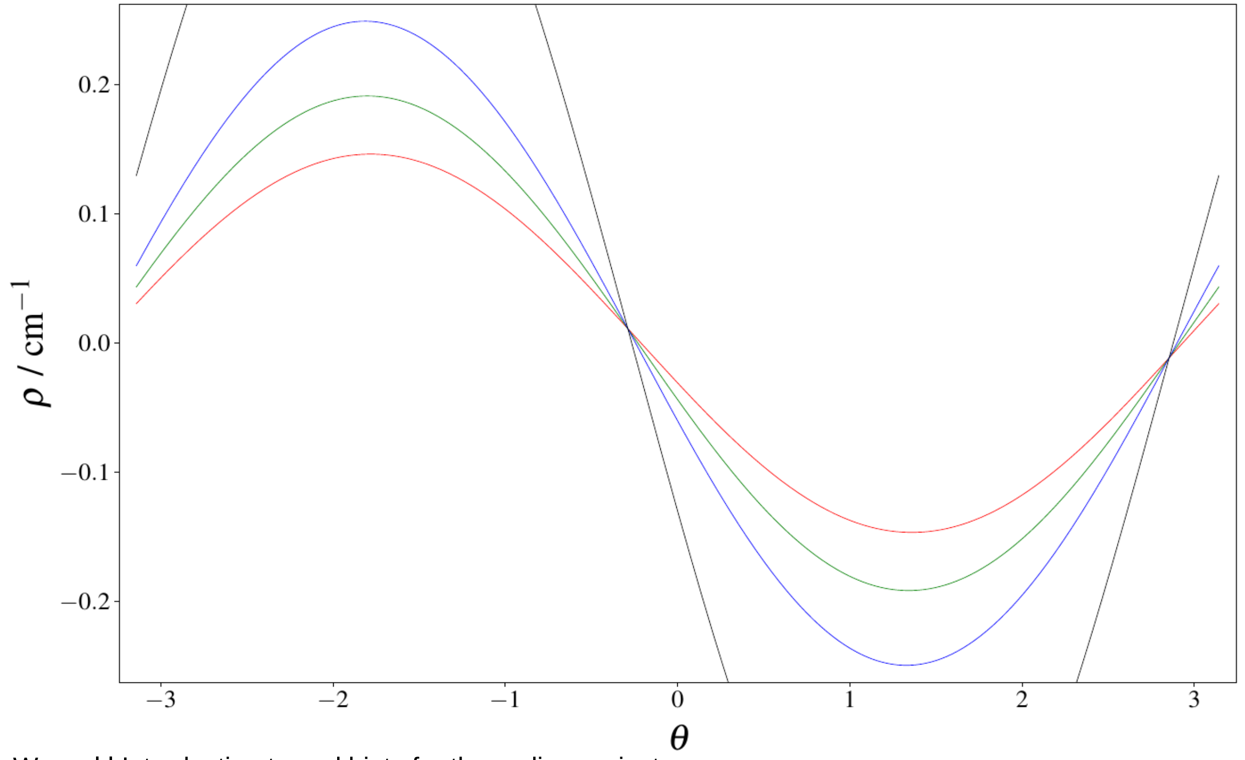
Your first task – an easy starter

Implement the conformal transformation

`void AcademyTrackingModule::conformalTransformation()`

- **`TODO 1`**`: actually calculate xConformal and yConformal`

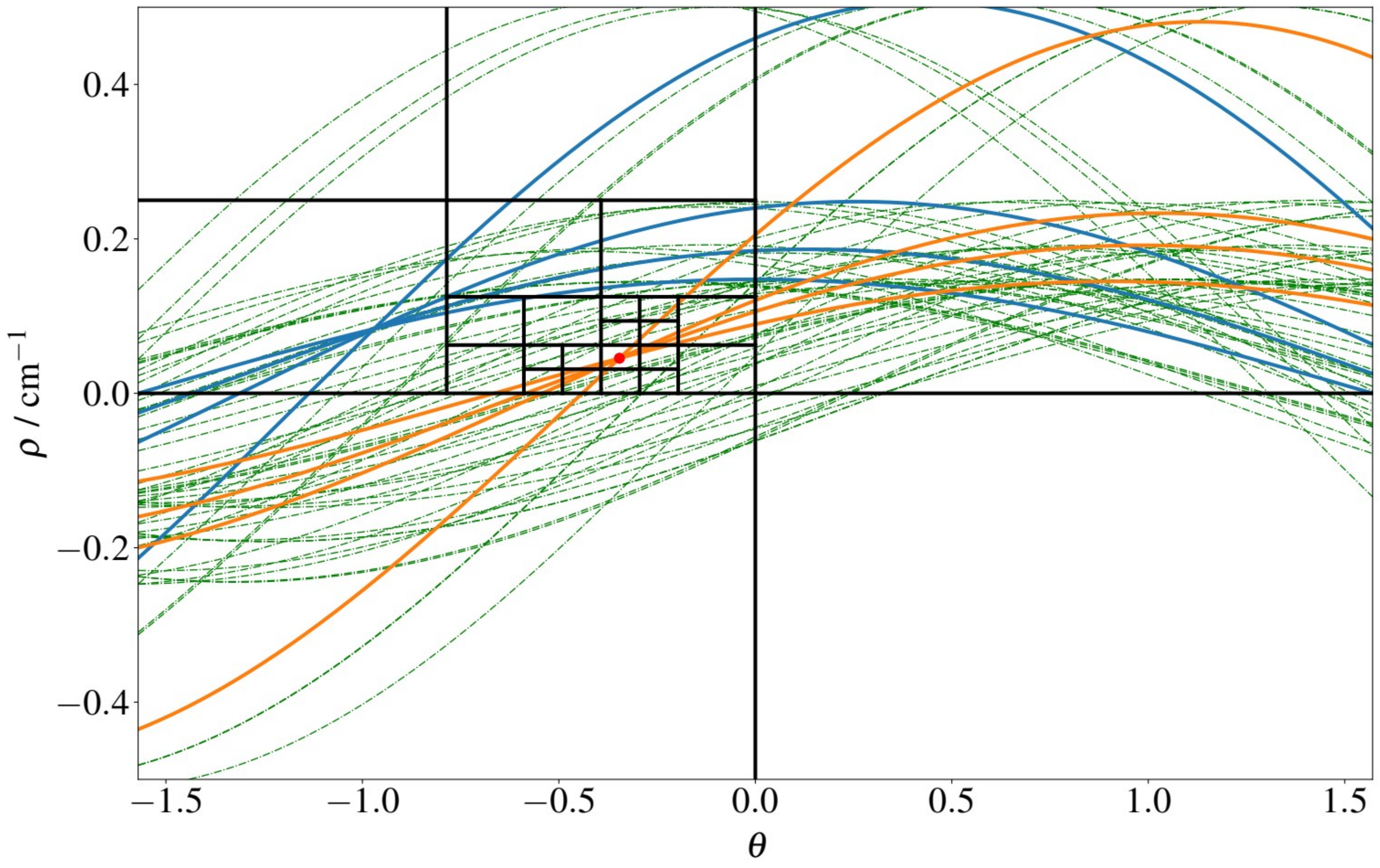
- `Compile the code using scons -j4`
- `Run` **`ParticleGunStarter.py`** `to check whether you actually get a Python script called` **`plotHoughTrafo.py`** `containing information about the hits`
- `Run the Python script` **`plotHoughTrafo.py`** `and check the output (HoughTrafoDebug.pdf)`
- `It should look like this:`



Intersection finding

Probing the Hough Space

- Binary search using a divide and conquer approach



Your second task – part 1

Implement the 2D binary search algorithm

```
void AcademyTrackingModule::fastHoughTransformation(const std::vector<SpacePointAndHitPair> hitsToCheck, float minX,
                                                    float maxX, float minY, float maxY, ushort currentRecursionLevel)
```

- Carefully check the TODOs
- **TODO 2**: You need to calculate the left and right boarders of the current sector, the following should help you (for x, y):
  - delta = (max – min) / 2
  - lowerBoarder = min + delta * step
  - upperBoarder = min + delta * (step + 1)
- **TODO 3**: Implement the sinusoidal lines
- **TODO 4**: We only care about lines with positive slope –> you should know how to calculate it ;)
- **TODO 5**: You need to check whether a line passes through a given sector
  - Make clear for yourself what it means if a line passes a rectangle in a cartesian coordinate system, i.e. what are the conditions for the line to be fully outside of a rectangle, to pass it horizontally (fully contained between minY and maxY), vertically (fully contained between minX and maxX), or whether it "cuts a corner"
  - If a sinusoidal has negative slope, or does not pass the current sector defined by the boarders we calculated above, we don't want to process it further
- **TODO 6**: If it does pass the sector, we want to fill **hitsInLayers** accordingly at the layer number
- **TODO 7**: And also add the current hitPair to **containedHitPairs** if you want to use it

Your second task – part 2

Implement the 2D binary search algorithm

```
void AcademyTrackingModule::fastHoughTransformation(const std::vector<SpacePointAndHitPair> hitsToCheck, float minX,
                                                    float maxX, float minY, float maxY, ushort currentRecursionLevel)
```

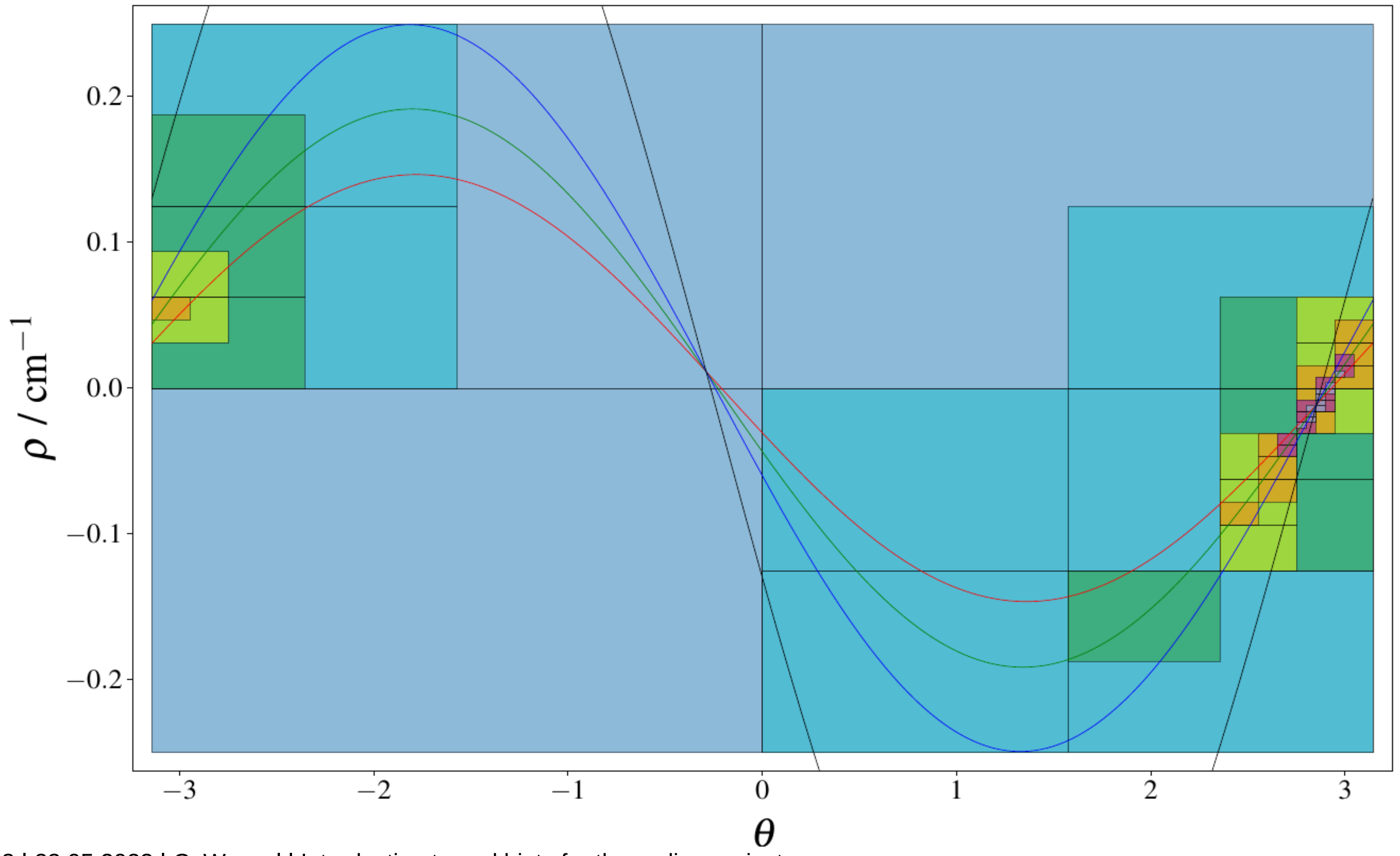
- Now we are at the **if (checkLayersHit(hitsInLayers))** part
- **TODO 8**: Give the correct arguments to **writeRectangleDebugOutput**
- **TODO 9**: Give the correct arguments to the recursive call of **fastHoughTransformation**
  - The arguments depend on whether or not you are making use of the list of hits that passed through the current sector, containedHitPairs
  - You don’t have to care about the condition breaking the recursion, it’s already there ;)
- **TODO 10**: We want to create cluster from the “active” Hough Space sectors later on. For this, we need some information. Since you should get more active slowly, less help is provided. But you should implement something like
  - a new **TrackCandidate** variable
  - a (range based) for loop over **containedHitPairs** in which you add the first value of each **HitPair** to the **TrackCandidate** you created in the last step
  - add the **TrackCandidate** you created to **m_trackCandidatesFromHoughSpace**
- **TODO 11**: Fill **m_activeSectors** and **m_HoughSpaceArray**
  - To calculate the index of each Hough Space sector, think about how 2D arrays usually work in a computer, and how you would represent this in a cartesian coordinate system. And also which things are different between the two.

Your second task – validation

Implement the 2D binary search algorithm

```
void AcademyTrackingModule::fastHoughTransformation(const std::vector<SpacePointAndHitPair> hitsToCheck, float minX,
                                                    float maxX, float minY, float maxY, ushort currentRecursionLevel)
```

- If you did everything correctly and re-run **ParticleGunStarter.py** followed by **plotHoughTrafo.py**, you should again get a PDF as before, which now should look like this:

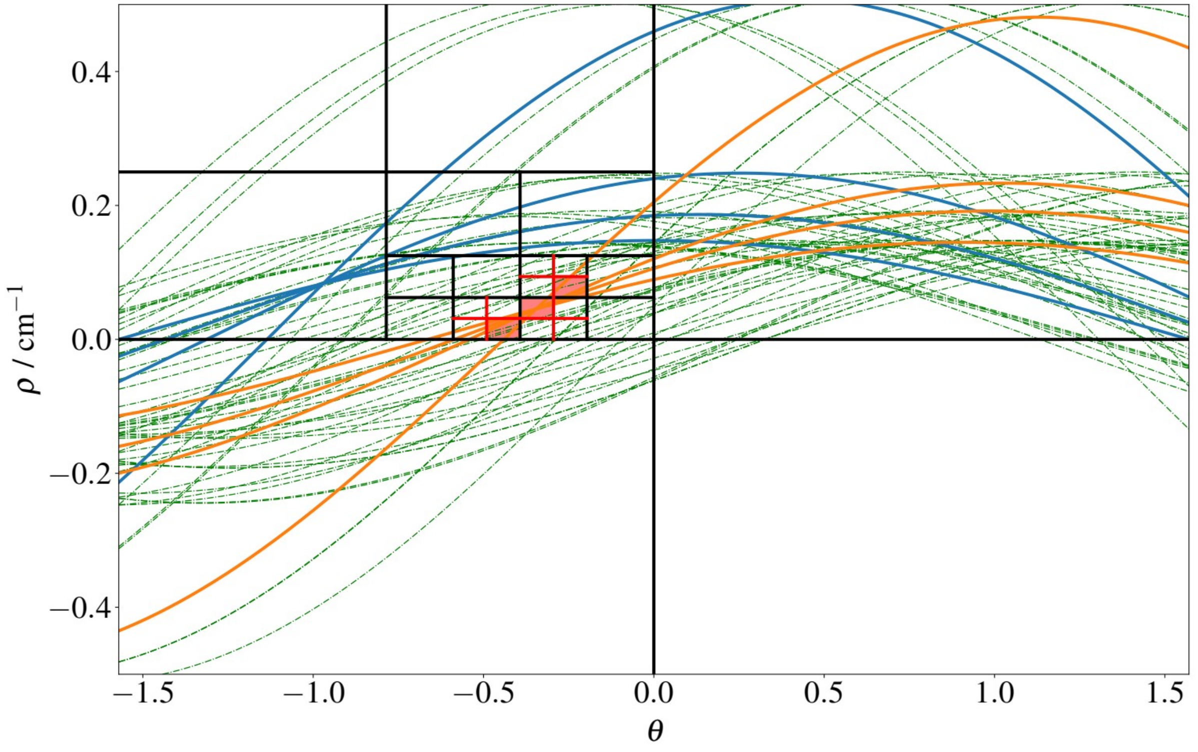


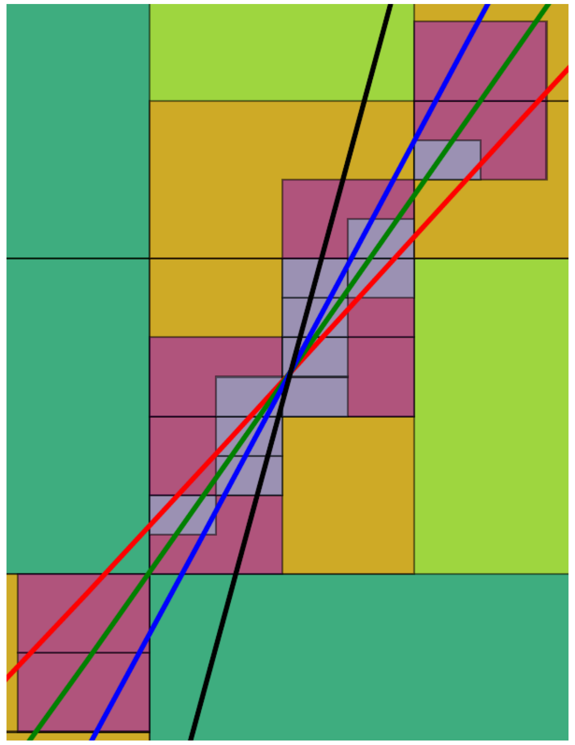
Your third task

Implement the 2D clustering algorithm

`void AcademyTrackingModule::findHoughSpaceClusters()`

- **`TODO 12:`** `Now you have to find clusters in the 2D Hough Space`
- `You can do whatever you want here! Be creative!!`
- `There are some hints in the comments in the code`
- `The only requirements: your input are` **`m_trackCandidatesFromHoughSpace`** `and your output are` **`m_rawTrackCandidates`**
- `While creating the clusters, you should make sure that you collect all the hits that belong to a cluster and create a new` **`TrackCandidate`** `with them, which is then stored in` **`m_rawTrackCandidates`**



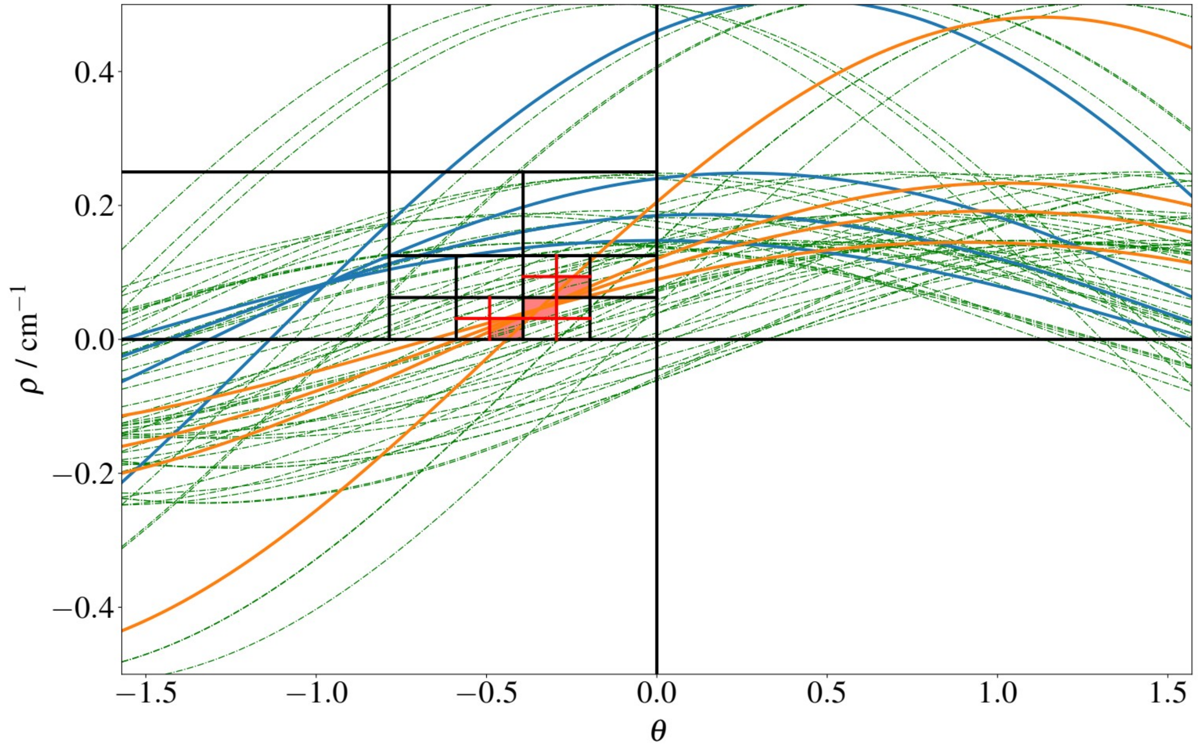


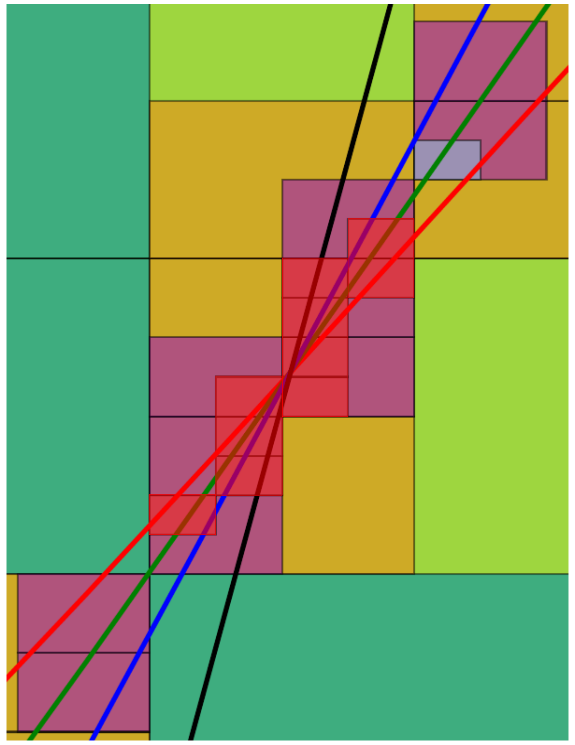
Your third task

Implement the 2D clustering algorithm

`void AcademyTrackingModule::findHoughSpaceClusters()`

- **`TODO 12:`** `Now you have to find clusters in the 2D Hough Space`
- `You can do whatever you want here! Be creative!!`
- `There are some hints in the comments in the code`
- `The only requirements: your input are` **`m_trackCandidatesFromHoughSpace`** `and your output are` **`m_rawTrackCandidates`**
- `While creating the clusters, you should make sure that you collect all the hits that belong to a cluster and create a new` **`TrackCandidate`** `with them, which is then stored in` **`m_rawTrackCandidates`**





Your fourth task

Clean the m_rawTrackCandidates

- **TODO 13:** Now you have to clean the raw track candidates
- You can do whatever you want here! Be creative!!
- There are some hints in the comments in the code
- Your input are the m_rawTrackCandidates created in **TODO 12**, and the output are a vector of **SpacePointTrackCandidates** (SPTC) **m_SpacePointTrackCands**

- The raw track candidates can still contain some contaminations from hits from other tracks (or background hits, if we used background, c.f. the dashed green lines in the left figure on the last slide)
- And there for sure also are raw track candidates that are just made up of random combinations of hits
- Think about a single track:
  - What are its properties?
  - What do the hits in a track have in common?
  - How many hits do you need for a track? I.e. do you want to keep track candidates with 1, 2, 3, 4, … hits?
    - Hint: the minimum required number of degrees of freedom (NDF) for a track is 5, and from each 2D hit you get 2 NDF (or 1 NDF per SVDCluster, but we anyway have 2 SVDCluster per SVDSpacePoint)
  - …
- If you are happy with your selection of hits, you have to create a SPTC for each track candidate
  - This is just a class that contains a vector of SpacePoints
  - They should be sorted in ascending order

Congratulations!

That‘s it – You have written a track finding algorithm!

- But of course that’s only part of the story
  - Optimisation of parameters
  - Validation with and without background
    - Goal: high track finding efficiency and low fake rate
  - For different event types (BB, cc, uds, tau, …)
  - Optimisation of execution time
  - …
- Especially dealing with background becomes really difficult
- As well as optimising the execution time

Thank you for your attention!

Contact

Deutsches Elektronen-
Synchrotron DESY

www.desy.de

Christian Wessel
Belle II
christian.wessel@desy.de