Utilizing emission profile variations in water Cherenkov detectors

2020-02-19 ICEPP Symposium
Lukas Berns, Tokyo Tech

Water-Cherenkov detectors

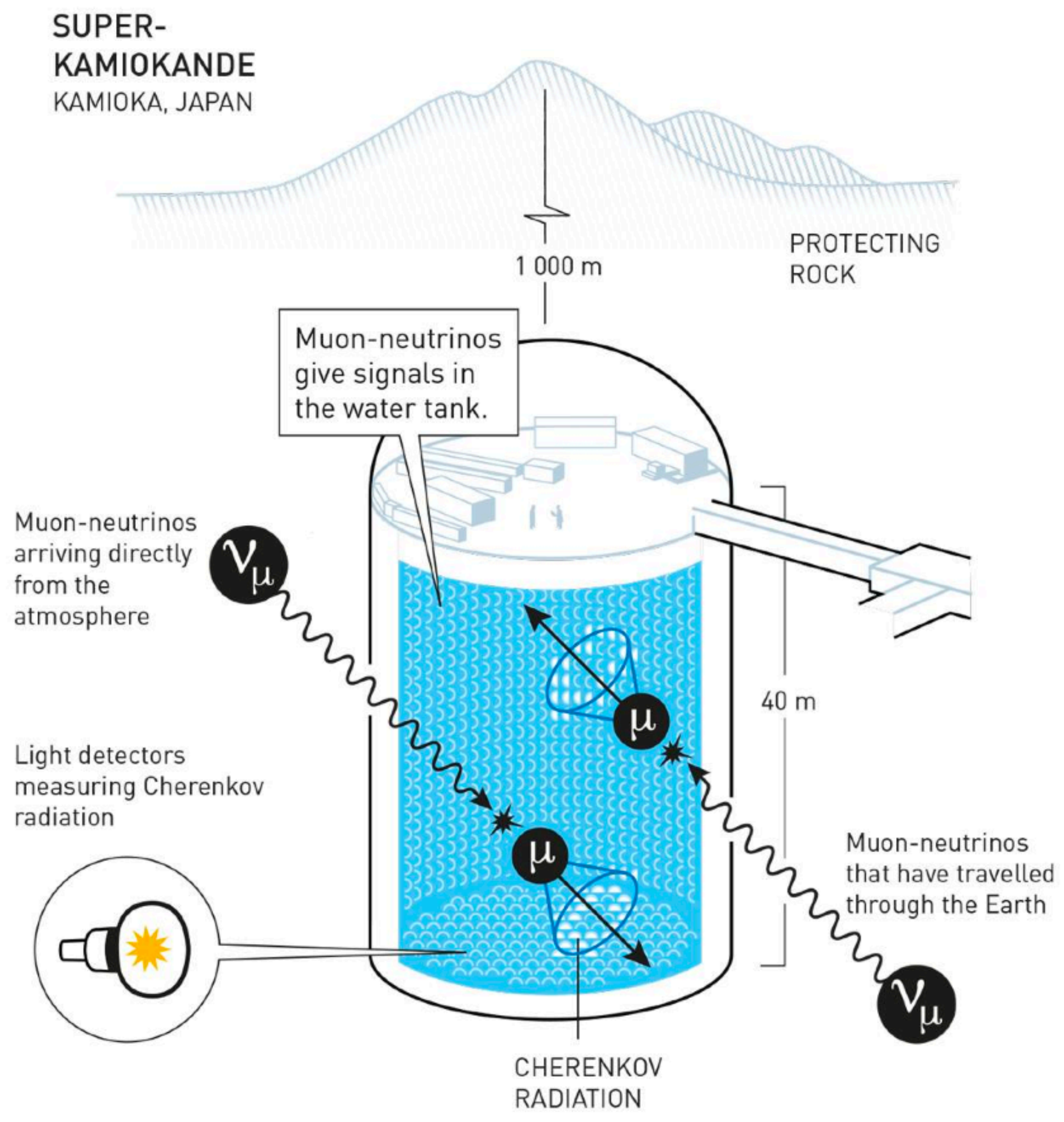
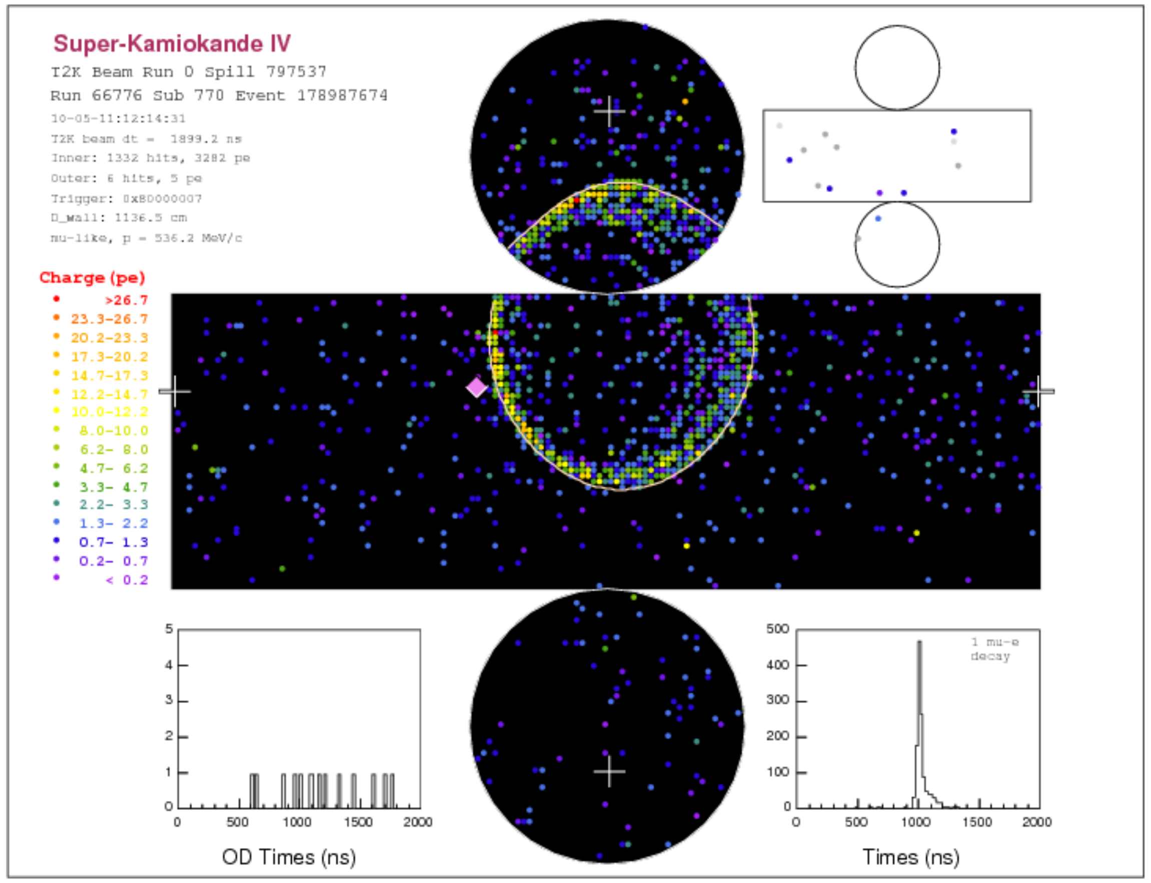

SUPER-KAMIOKANDE
KAMIOKA, JAPAN
1 000 m
PROTECTING ROCK
Muon-neutrinos give signals in the water tank.
Muon-neutrinos arriving directly from the atmosphere
ν_μ
μ
40 m
Light detectors measuring Cherenkov radiation
Muon-neutrinos that have travelled through the Earth
CHERENKOV RADIATION

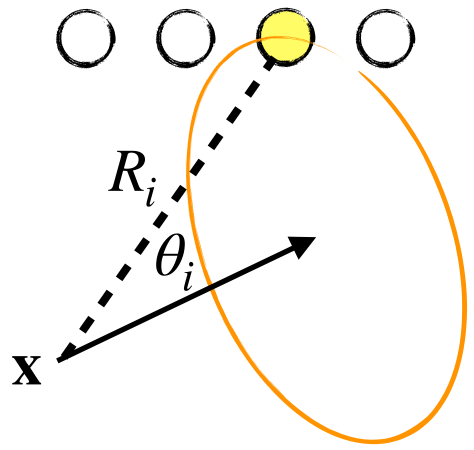
Illustration: © Johan Jarnestad/The Royal Swedish Academy of Sciences

- Rich physics program:
  ν osc., leptonic CPV, proton decay, supernova, …
- Reconstruct charged particle information from charge+time of PMT hits
- Very good e/μ separation

Here using the example of *fiTQun*

Ring-fitting right now





- **Maximum-likelihood** based reconstruction.
- Maximize over track parameters for particle of type α:
  - energy E
  - position $\mathbf{x}$
  - time t
  - direction $\mathbf{n}$

For PMTs i with hit: $\prod_i f_q(q_i) f_t(t_i)$

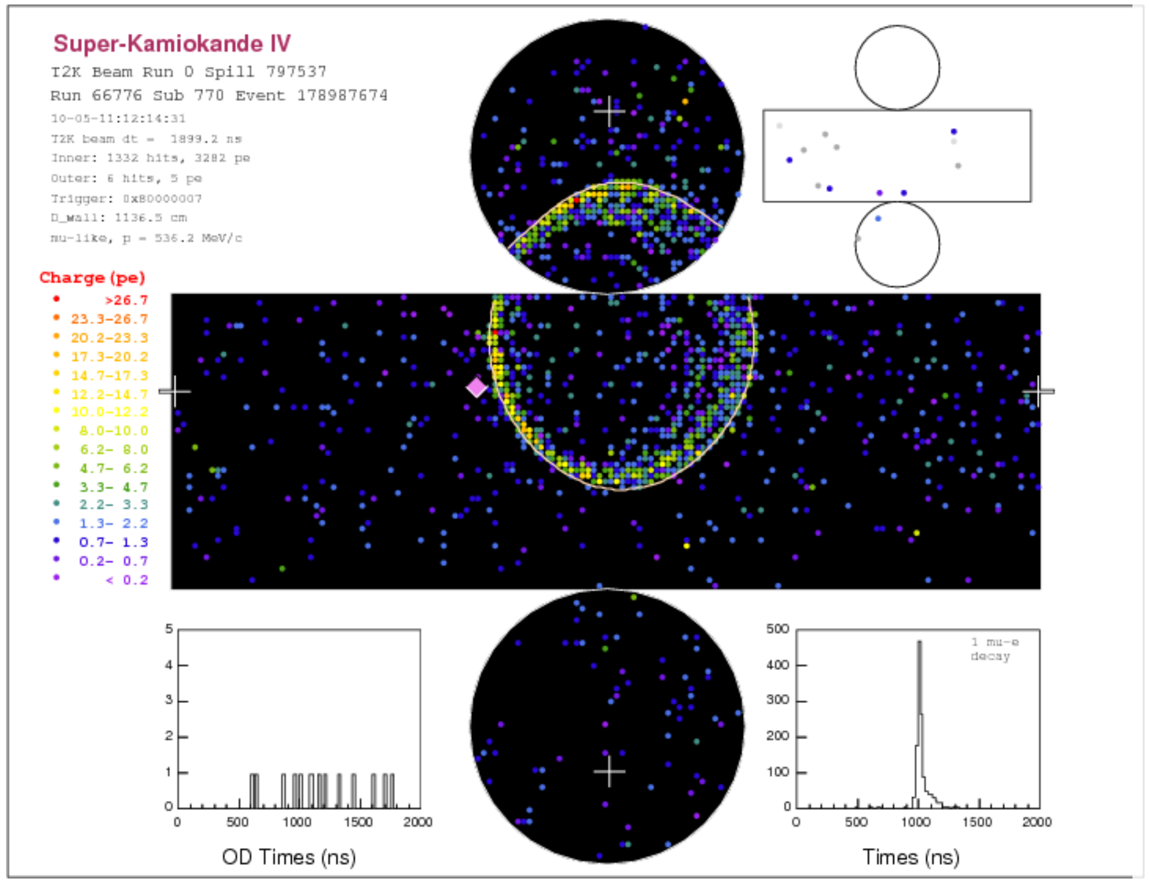
- **charge likelihood** f_q
  Mostly **poisson** with mean #photons as function of opening angle θ_i and distance R_i
- **time likelihood** f_t
  mostly just function of distance R_i

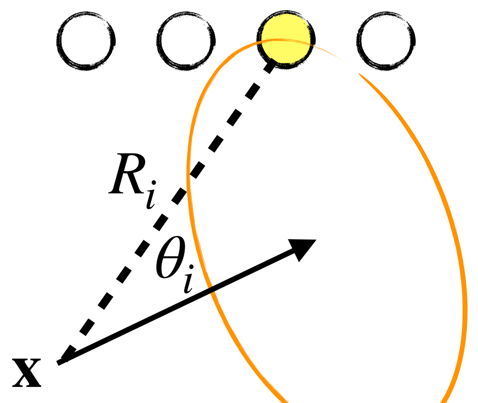
For PMTs i with no hit: $\prod_i P_{\text{unhit}, i}$

… mostly a function of θ_i , R_i

Here using the example of *fiTQun*

Ring-fitting right now





- **Maximum-likelihood** based reconstruction.
  optimal if model pdf = true pdf
- Maximize over track parameters for particle of type α:
  - energy E
  - position $\mathbf{x}$
  - time t
  - direction $\mathbf{n}$

For PMTs i with hit: $\prod_i f_q(q_i) f_t(t_i)$

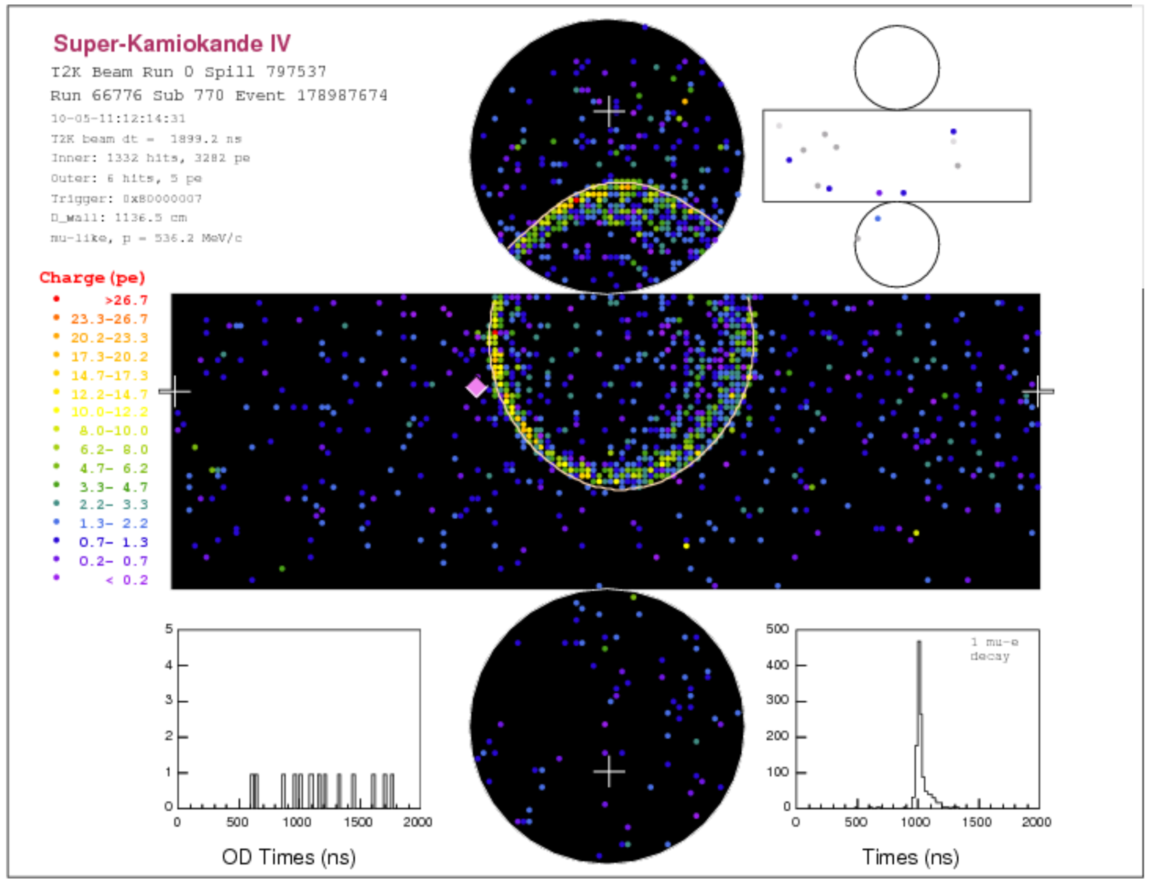
- **charge likelihood** f_q
  Mostly **poisson** with mean #photons as function of opening angle θ_i and distance R_i
- **time likelihood** f_t
  mostly just function of distance R_i

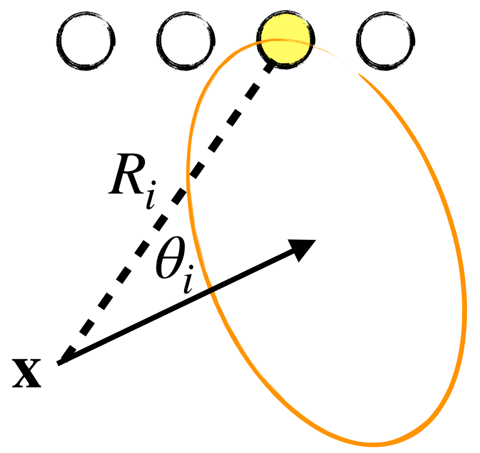
For PMTs i with no hit: $\prod_i P_{\text{unhit}, i}$

… mostly a function of θ_i , R_i

Here using the example of *fiTQun*

Ring-fitting right now





- **Maximum-likelihood** based reconstruction.
  optimal if model pdf = true pdf
- Maximize over track parameters for particle of type α:
  - energy E
  - position $\mathbf{x}$
  - time t
  - direction $\mathbf{n}$

Each PMT is assumed to have independent errors

no way!

For PMTs i with hit: $\prod_i f_q(q_i) f_t(t_i)$

- **charge likelihood f_q**
  Mostly **poisson** with mean #photons as function of opening angle θ_i and distance R_i
- **time likelihood f_t**
  mostly just function of distance R_i

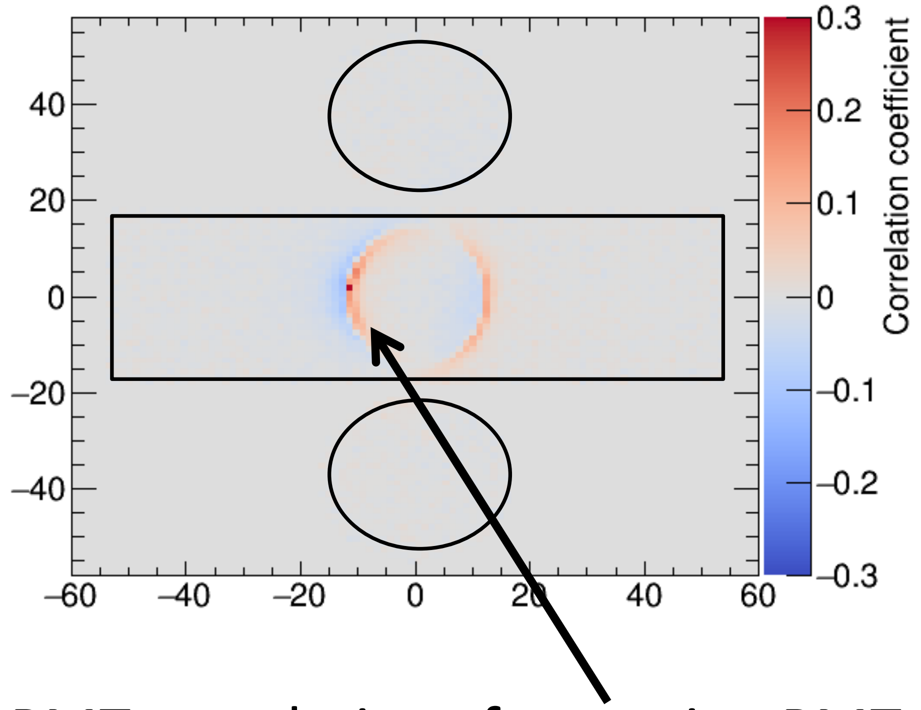
For PMTs i with no hit: $\prod_i P_{\text{unhit},\, i}$

… mostly a function of θ_i, R_i

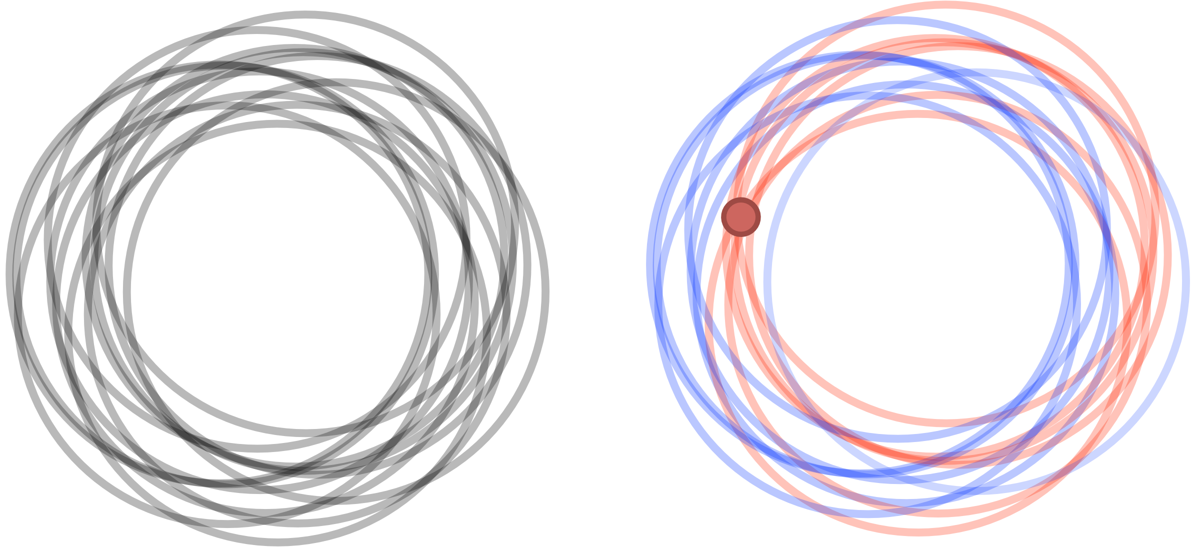
PMT-correlations
their origins and magnitude

- Simulate many events with 500 MeV electron of same position, direction etc. and study correlation of charges for PMT pairs.

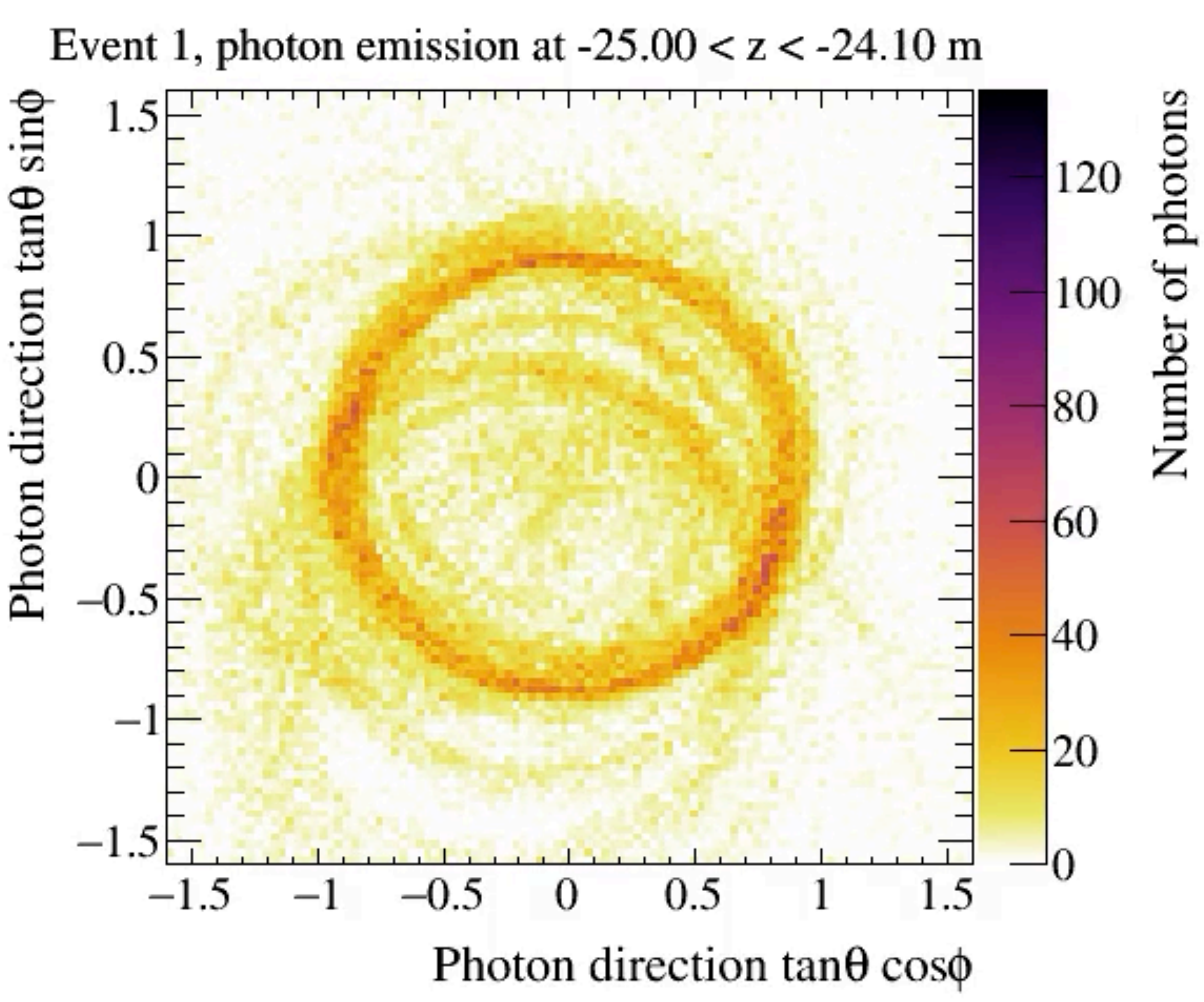
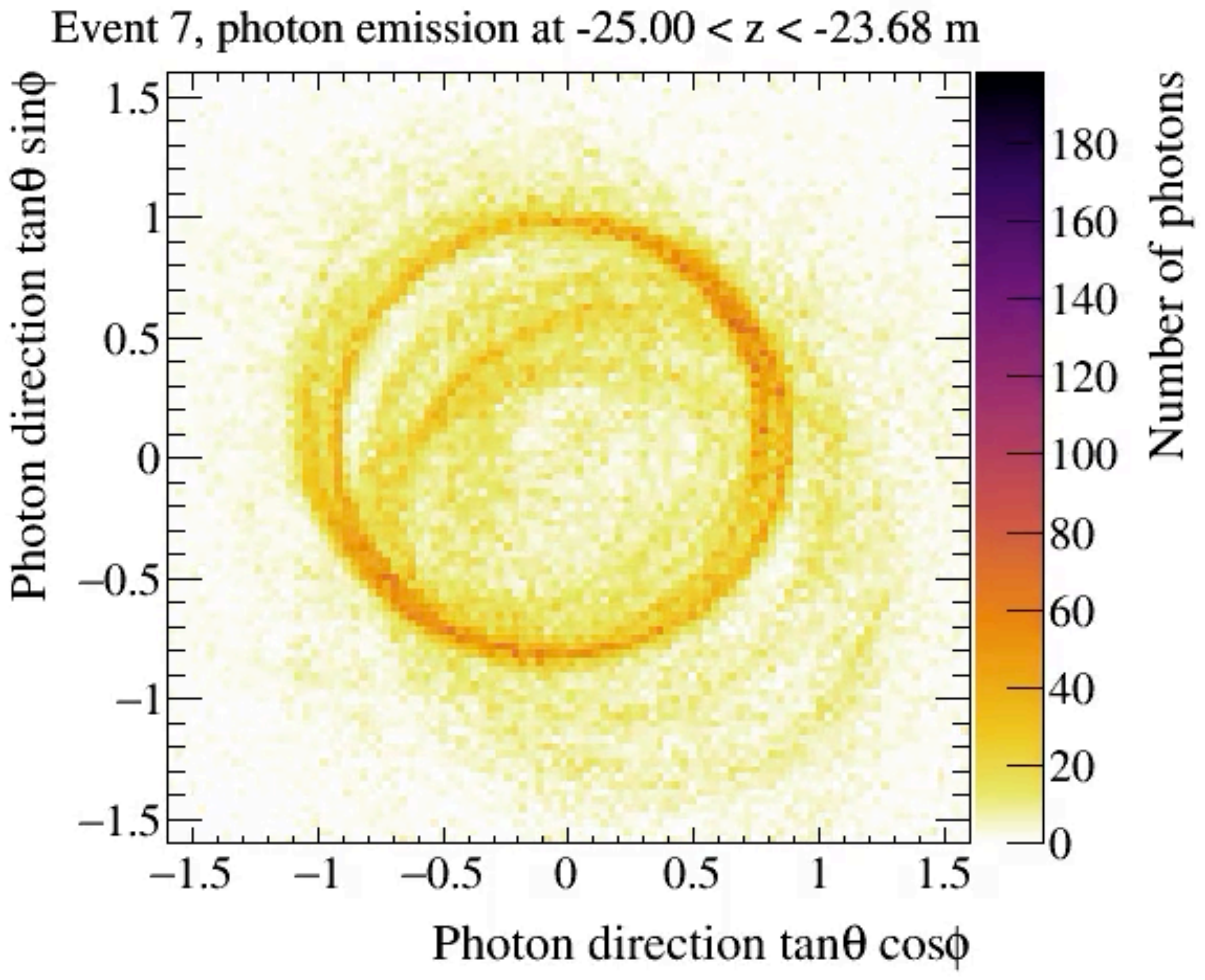
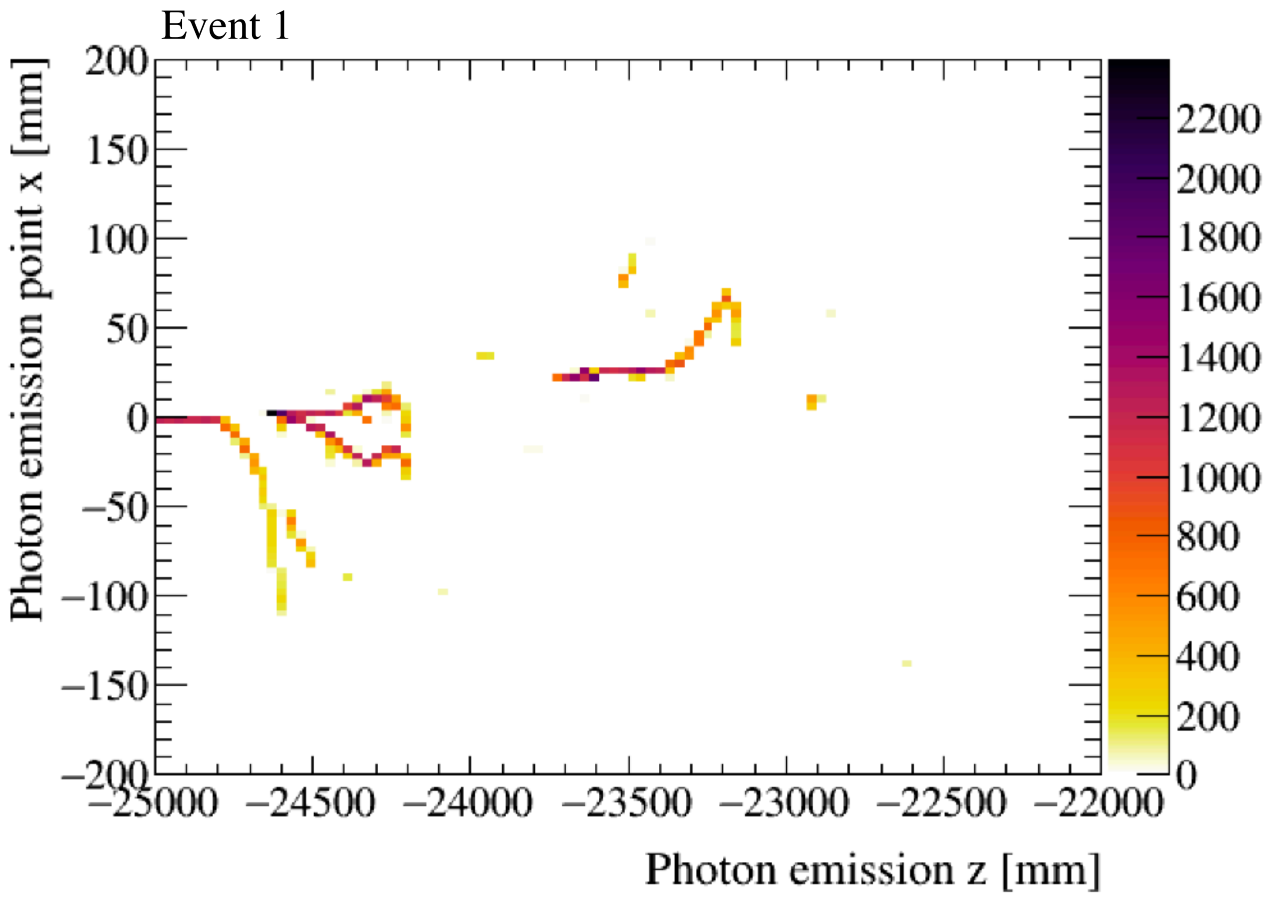
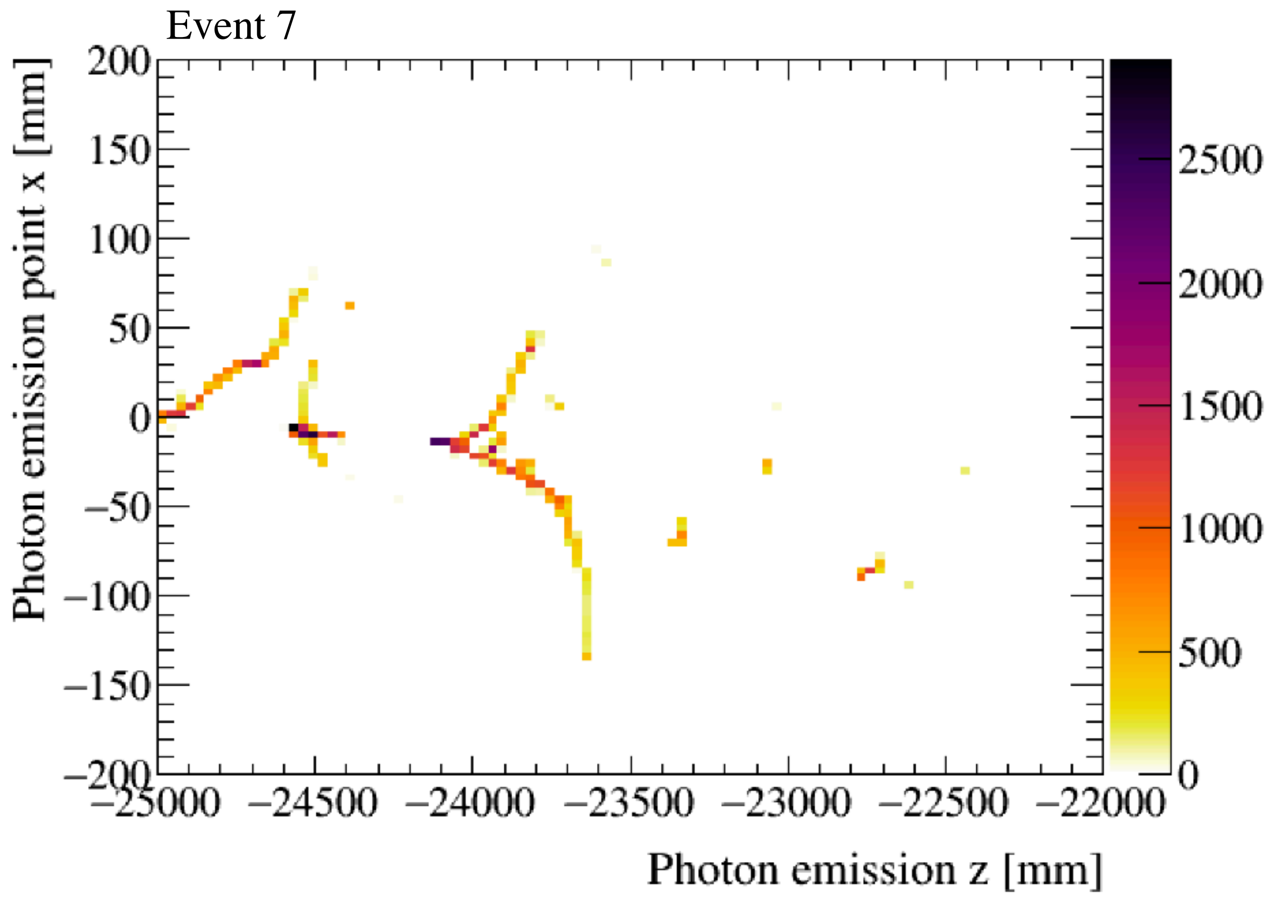
- For PMTs near Cherenkov angle, **about 30% correlation** → non-negligible?

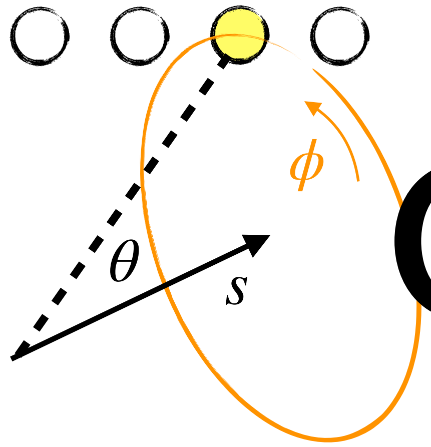


PMT correlations for on-ring PMT

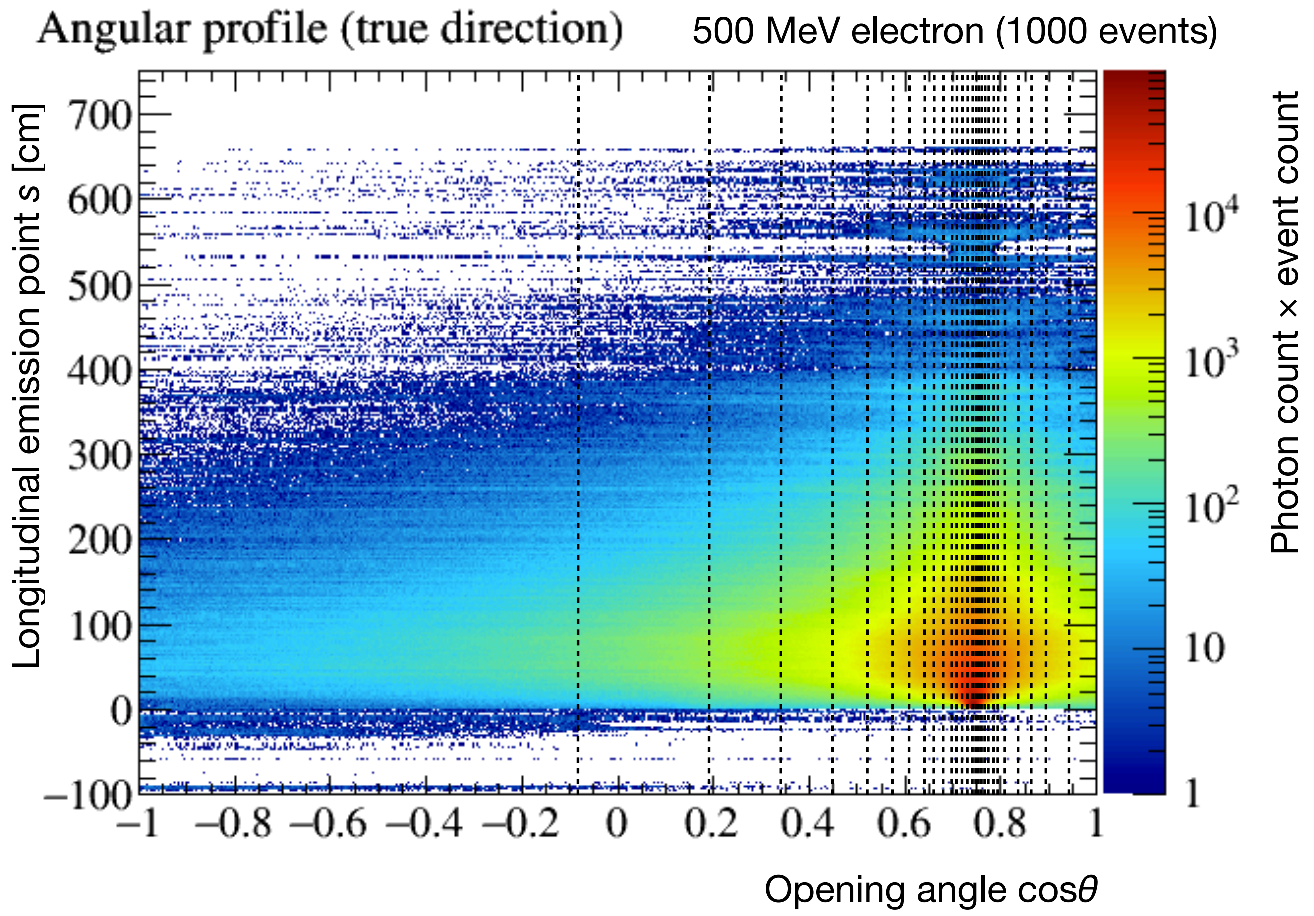


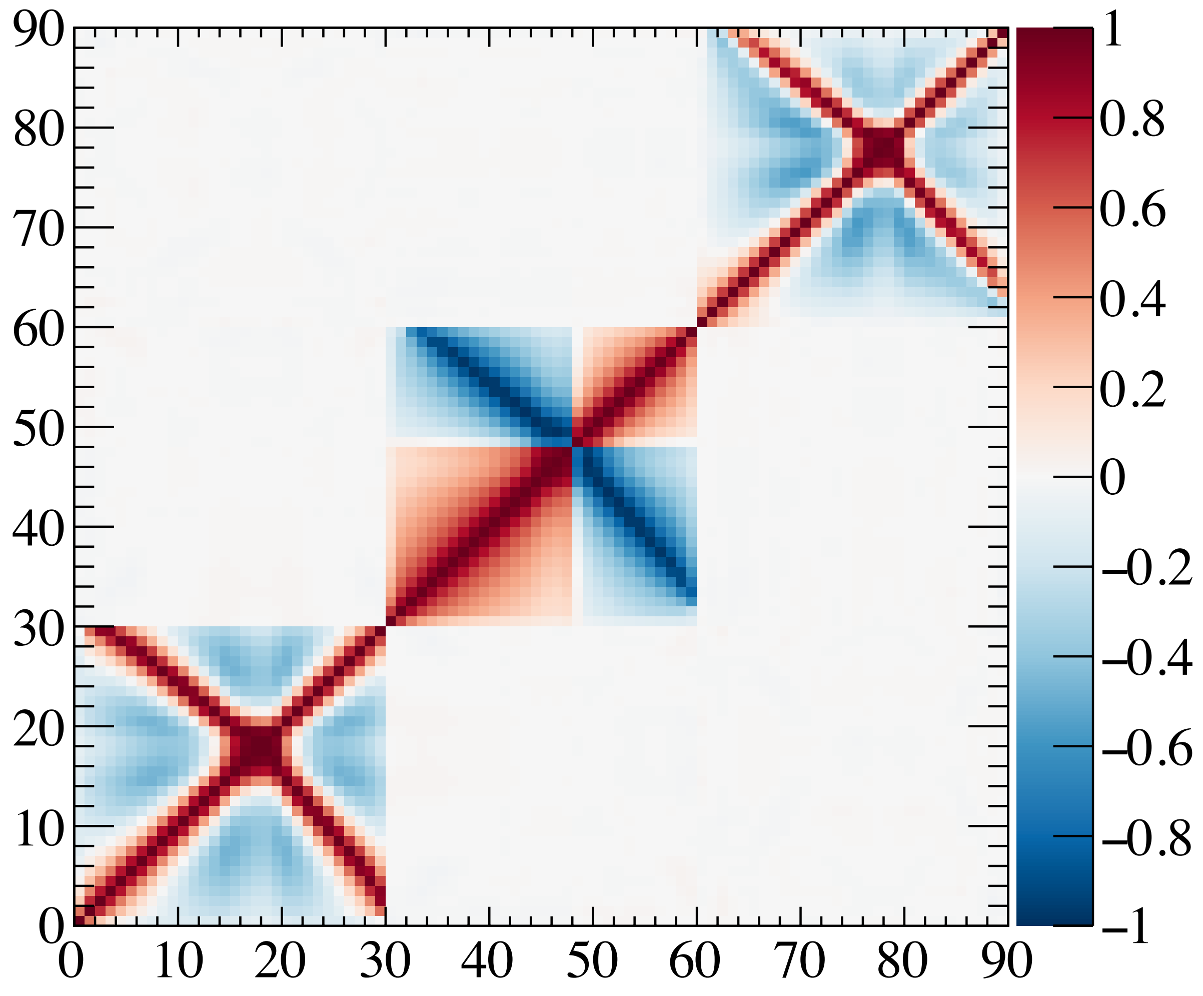
←
For electrons correlations are mostly due to showering. Essentially we have not a single ring, but an **ensemble of rings**.
For μ multiple-scattering also contributes, but these are already modeled in fiTQun.


Event 1, photon emission at -25.00 < z < -24.10 m
Photon direction tanθ sinφ
Photon direction tanθ cosφ
Number of photons

Event 7, photon emission at -25.00 < z < -23.68 m
Photon direction tanθ sinφ
Photon direction tanθ cosφ
Number of photons

Event 1
Photon emission point x [mm]
Photon emission z [mm]

Event 7
Photon emission point x [mm]
Photon emission z [mm]



Correlation matrix





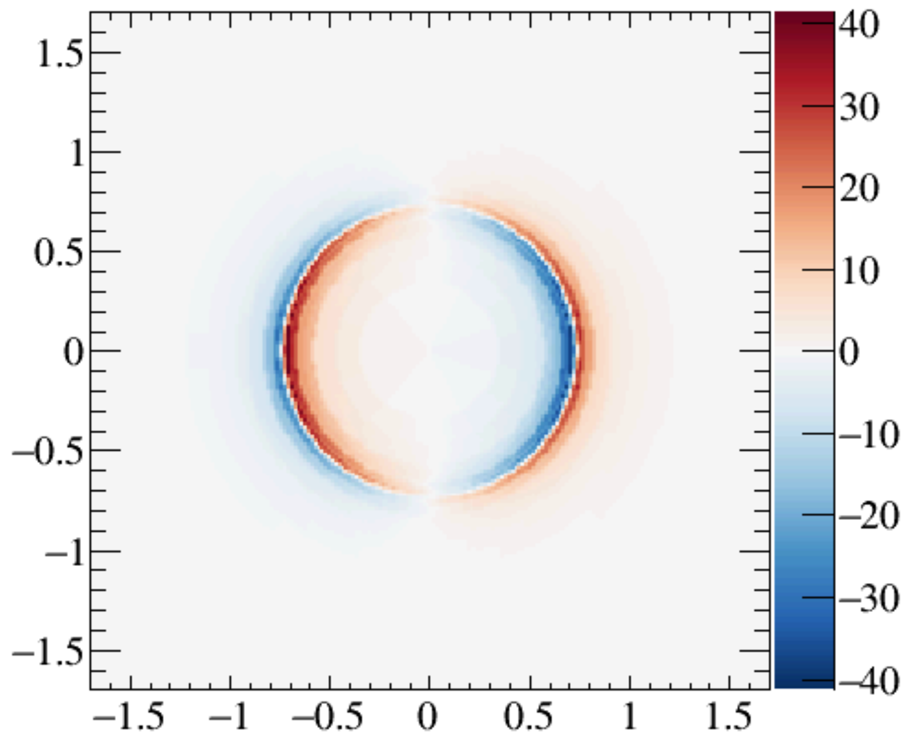
- For simplicity study variance of #photons integrated over longitudinal emission direction
- Bins are chosen to capture most important features with small number of bins (30) … see backup.

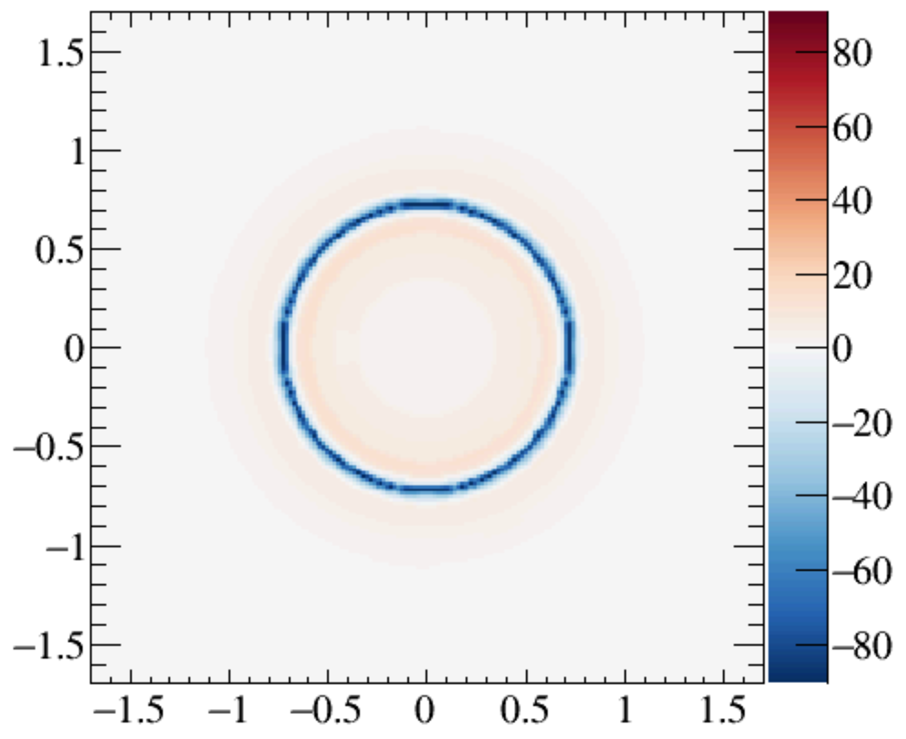
Taylor expansion in ϕ

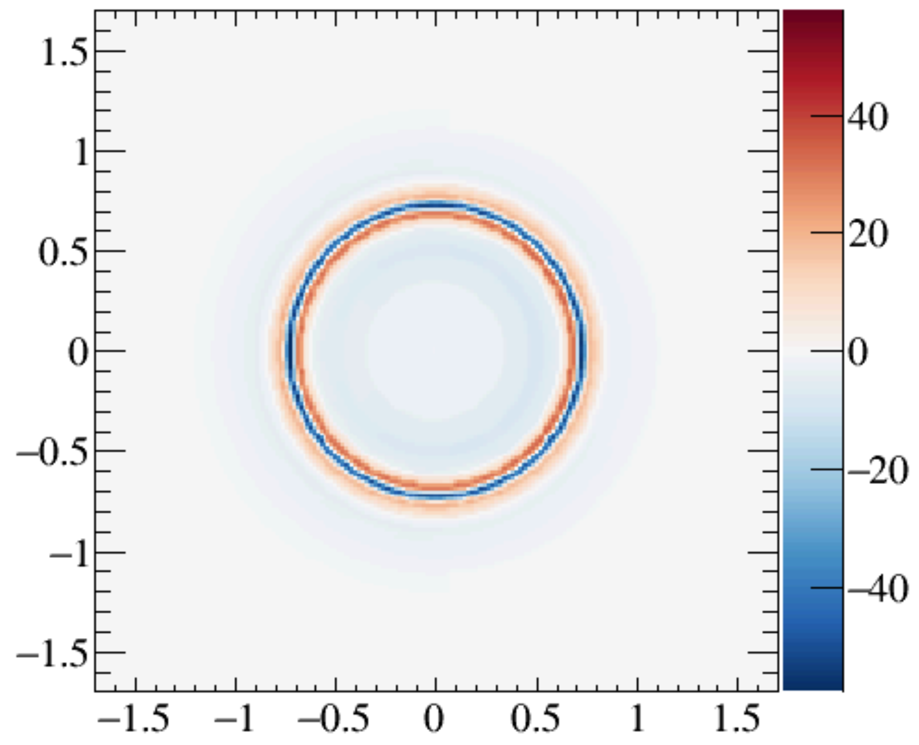
$$f(\theta,\phi) = \Sigma_m f_m(\theta)\, e^{im\phi}$$

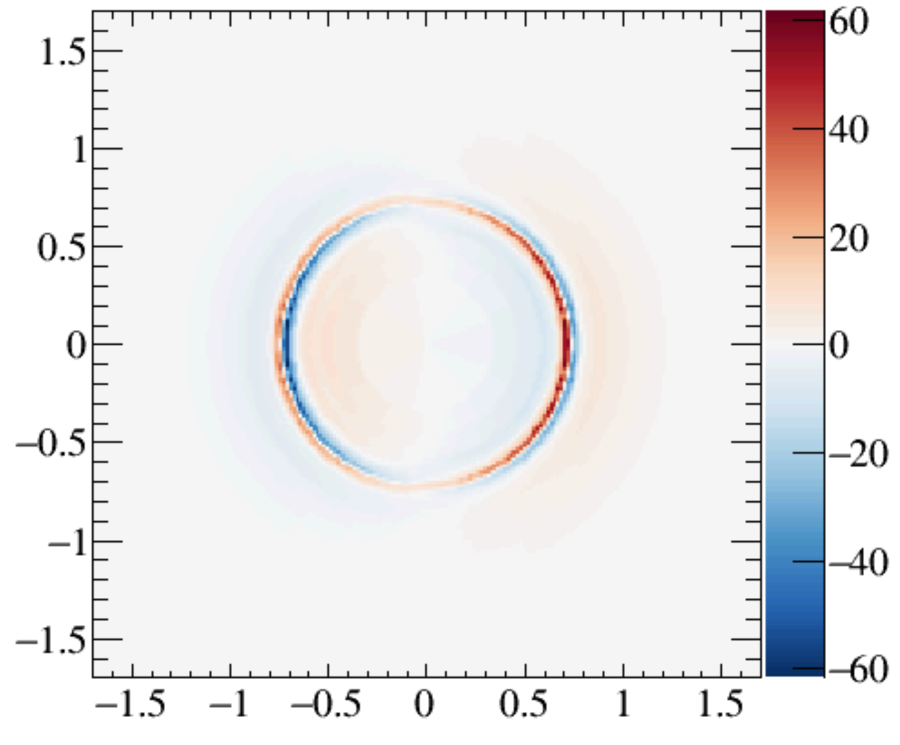
Eigenvectors (principal modes)

Principal modes are mostly *dipole* and *monopole*, some *quadrupole*(#7)


Eigenvector 0, std. deviation = 758.023


Eigenvector 1, std. deviation = 645.028


Eigenvector 3, std. deviation = 455.422


Eigenvector 4, std. deviation = 421.681

Basic symmetry structure

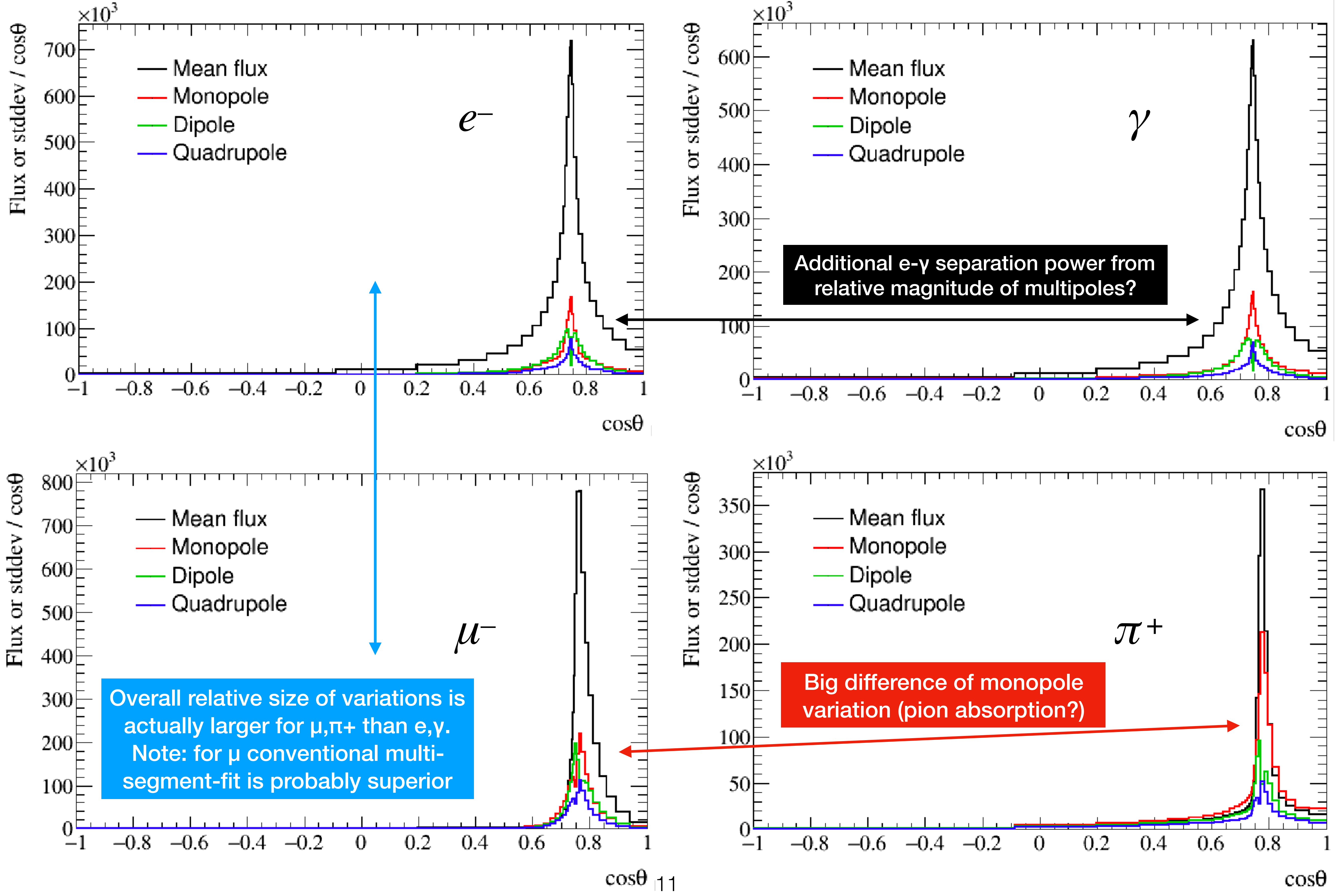
- Expand in Fourier modes $f(\theta, \phi) = \Sigma_m f_m(\theta)\, e^{im\phi}$
- Azimuthal symmetry
  $$\frac{\partial}{\partial\phi}\big\langle f(\theta,\phi)\big\rangle = \frac{\partial}{\partial\phi}\big\langle f(\theta,\phi) f(\theta',\phi+\Delta\phi)\big\rangle = 0$$
  implies
  - $\langle f_m \rangle = 0$ unless $m = 0$
  - $\langle f^*_m f'_{m'} \rangle = 0$ unless $|m| = |m'|$
  - Real and imaginary parts of f_m can be seen as $\cos$ and $\sin$ terms (f^c, f^s) with $m \geq 0$.
    Then for same m: $\langle f^c f^{c'} \rangle = \langle f^s f^{s'} \rangle, \langle f^c f^{s'} \rangle = 0$

With conventional method (mean profile) no azimuthal dependence

Multipoles have non-zero variations.
Correlations between different multipoles cannot be captured with this method.

Actually it is possible to study the correlations, see backup. In fact probably necessary.

Magnitude of variations



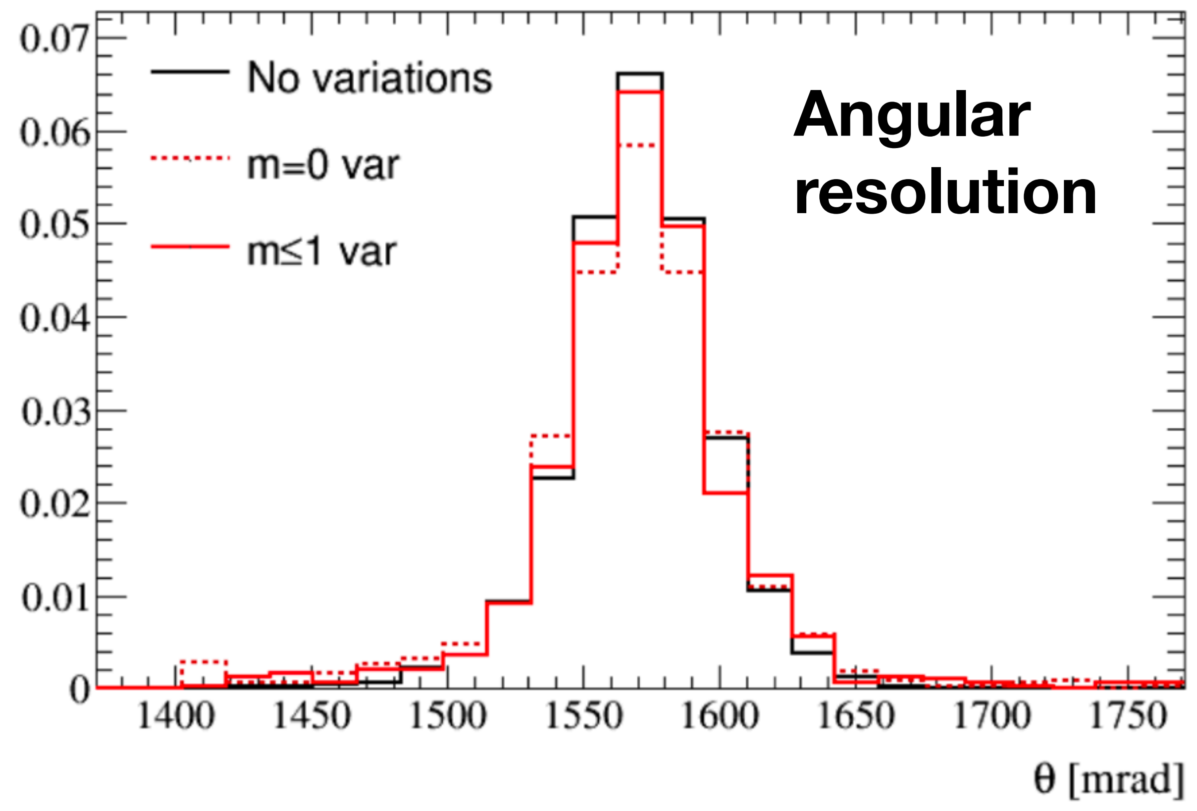
Additional e-γ separation power from relative magnitude of multipoles?

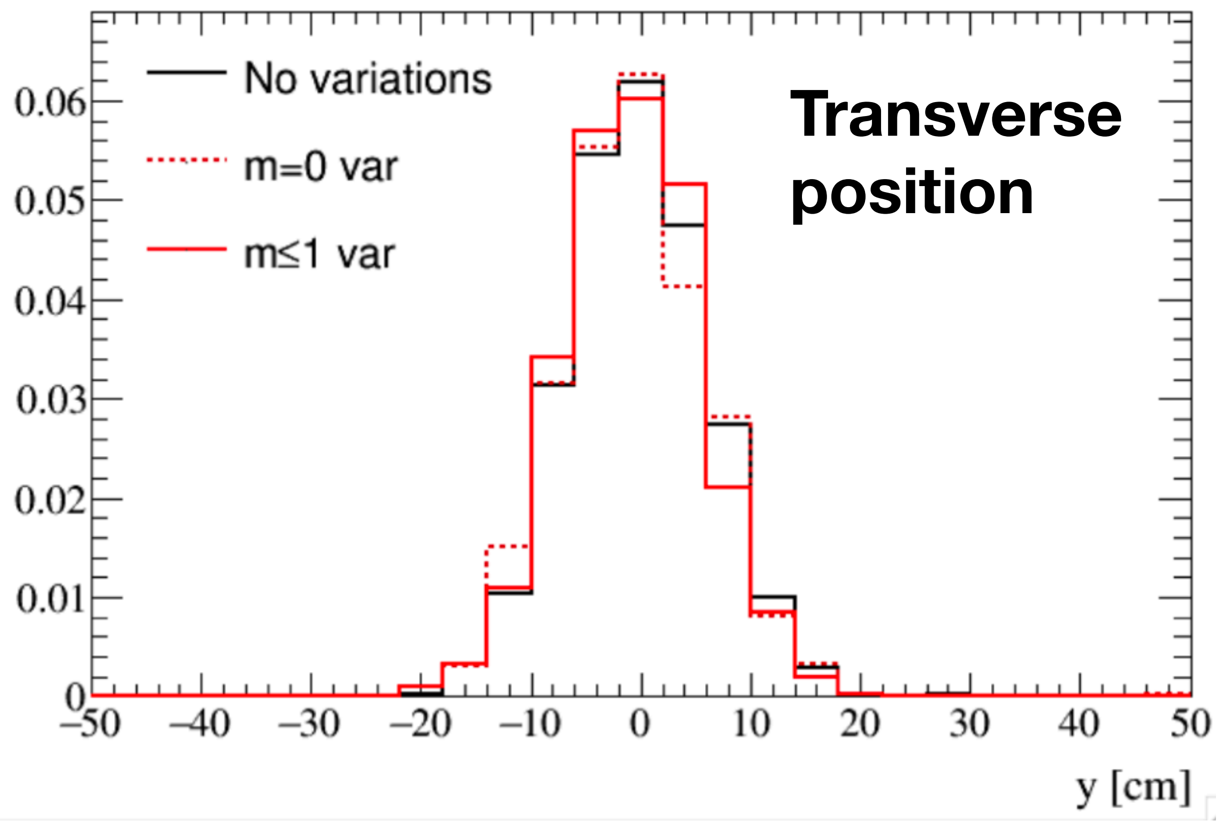
Overall relative size of variations is actually larger for μ,π+ than e,γ. Note: for μ conventional multi-segment-fit is probably superior

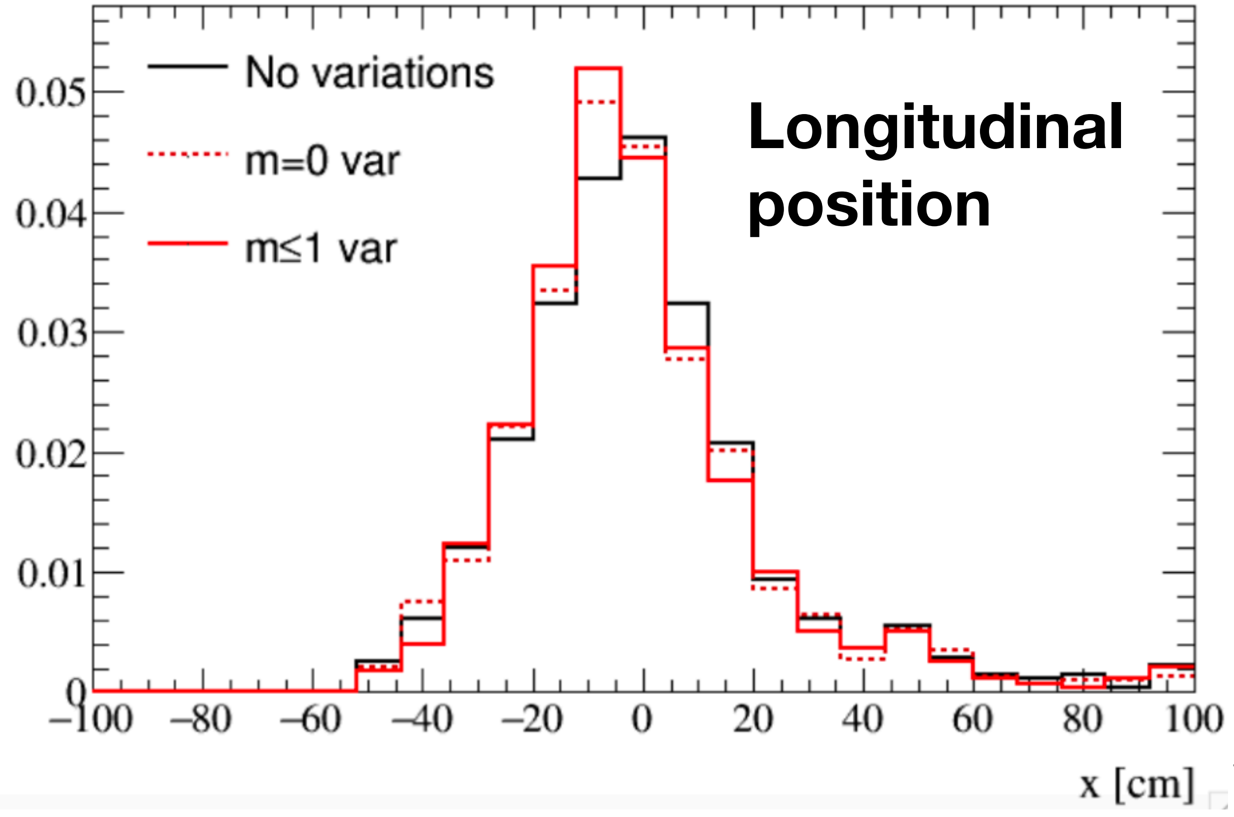
Big difference of monopole variation (pion absorption?)

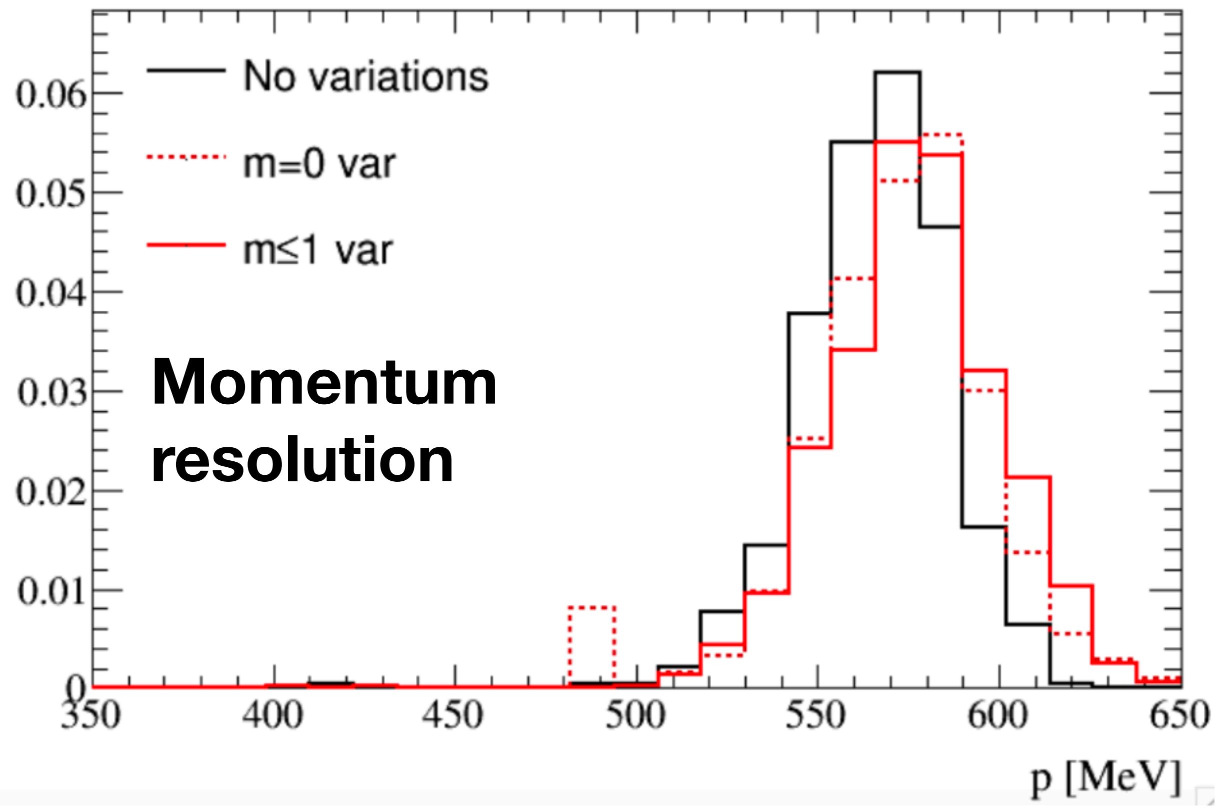
Electron track reconstruction performance

500 MeV electron emitted at center of tank moving in horizontal direction.









Electron track reconstruction performance

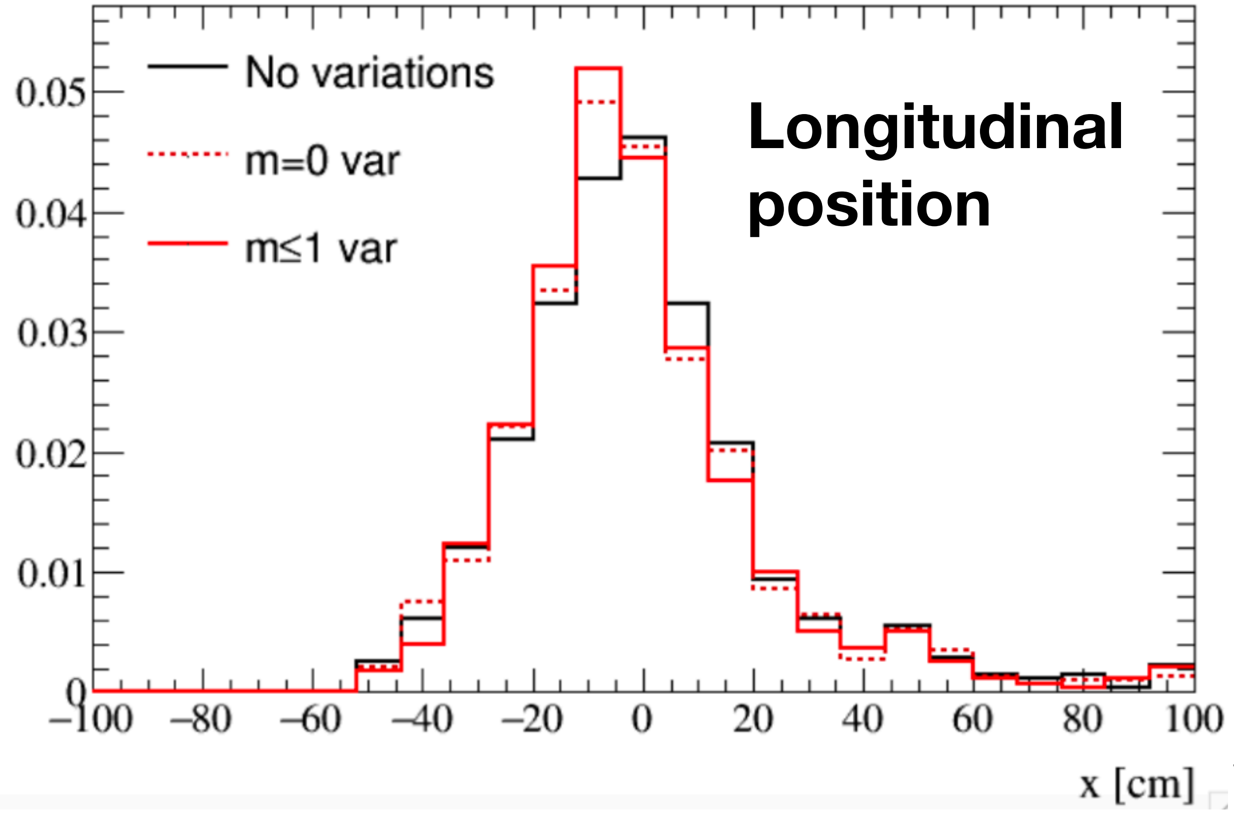
500 MeV electron emitted at center of tank moving in horizontal direction.

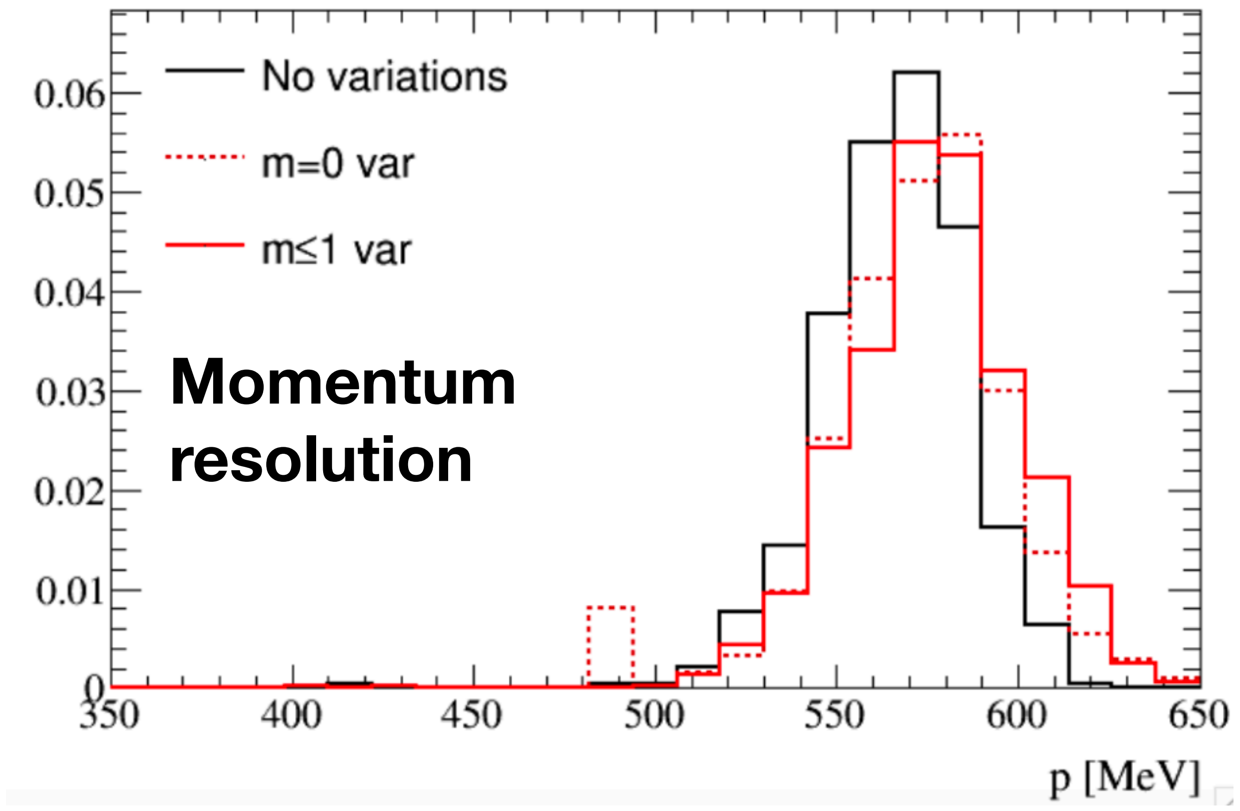
Some are slightly better, some slightly worse.

Differences seen here could also be due to fitting artifacts.

note: #fit-params increases from 7 (no var) → 37 (m=0) → 67 (m≤1) …

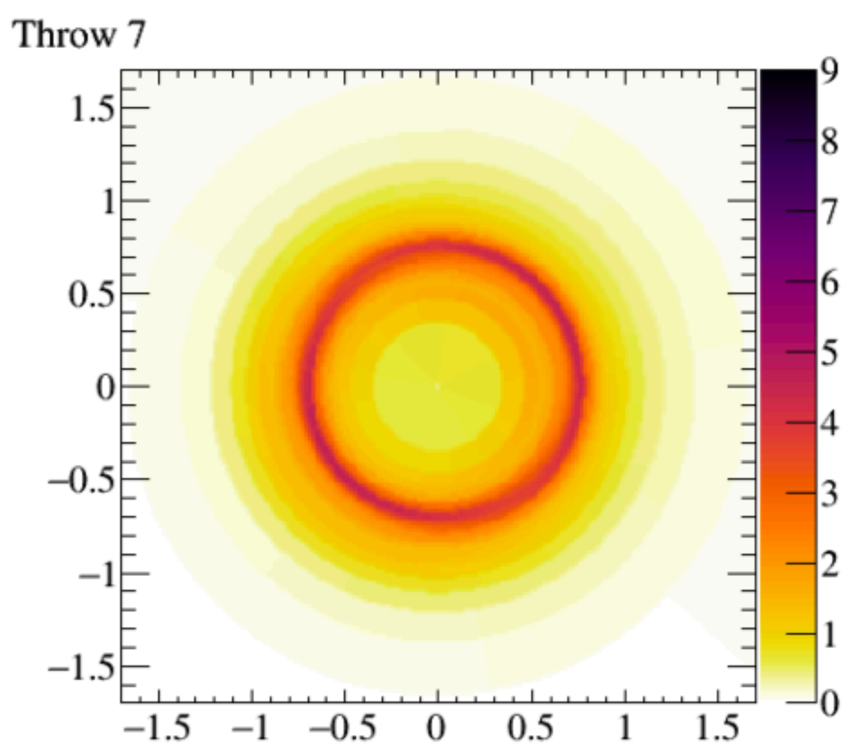
In any case no significant improvements for reconstruction…
possibly the model is unable to capture essential features

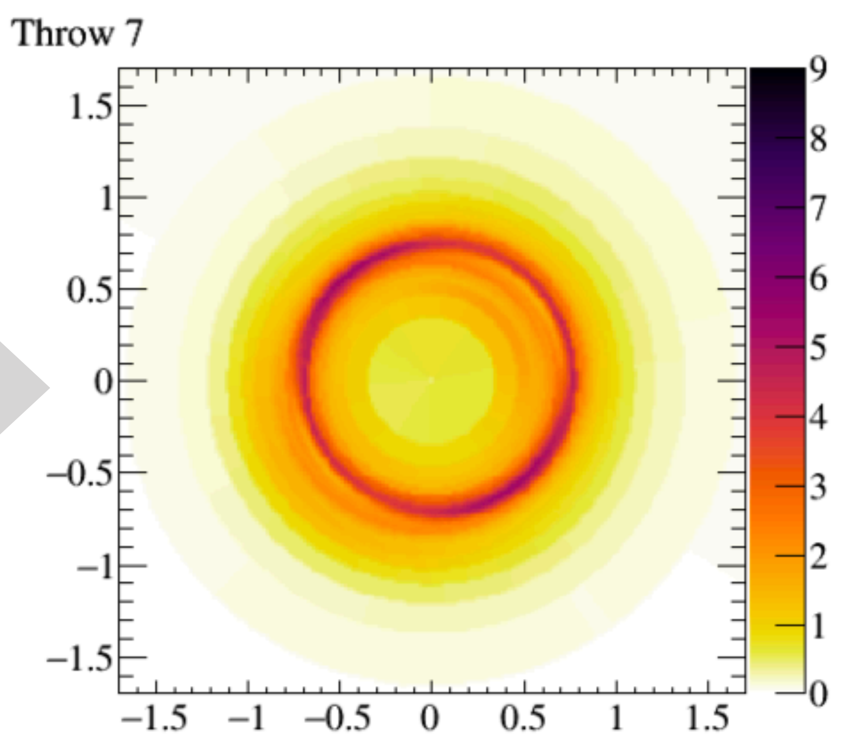


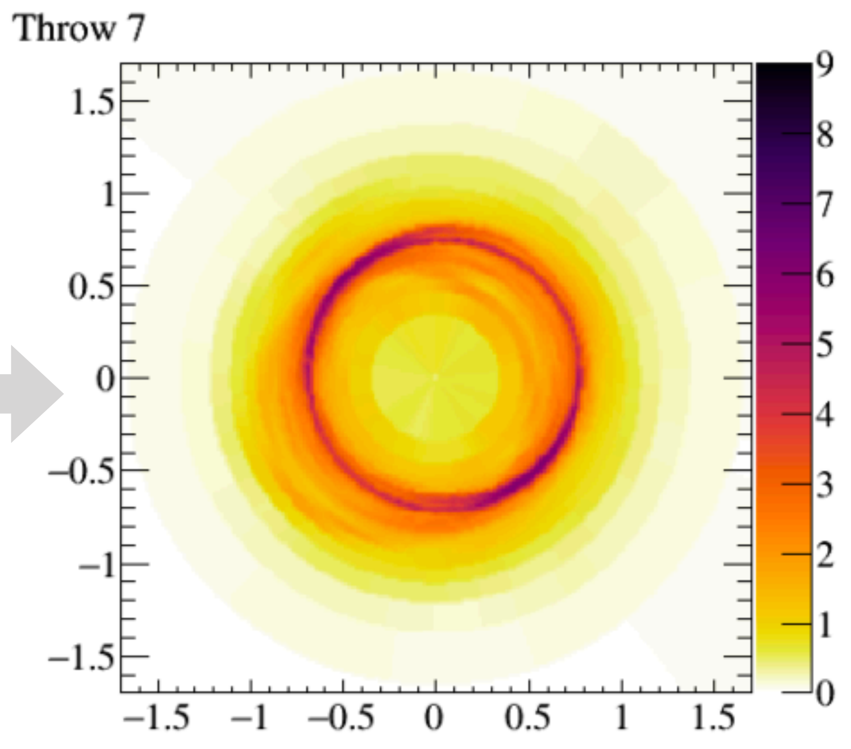


How to improve the model + next steps

$m \leq 1$ → $m \leq 2$ → $m \leq 10$







- Dipole variations correspond to vertex shift and should thus be impossible to fit. Features such as double-rings demand **quadrupole** or higher modes. **Correlations between multipoles** are essential, otherwise just adding "noise".
- Also electrons have variation in **track length**, which is not seen for muons.
- Tried several approaches but **not easy** to realize efficiently. (ideas welcome) PID performance also to be studied.

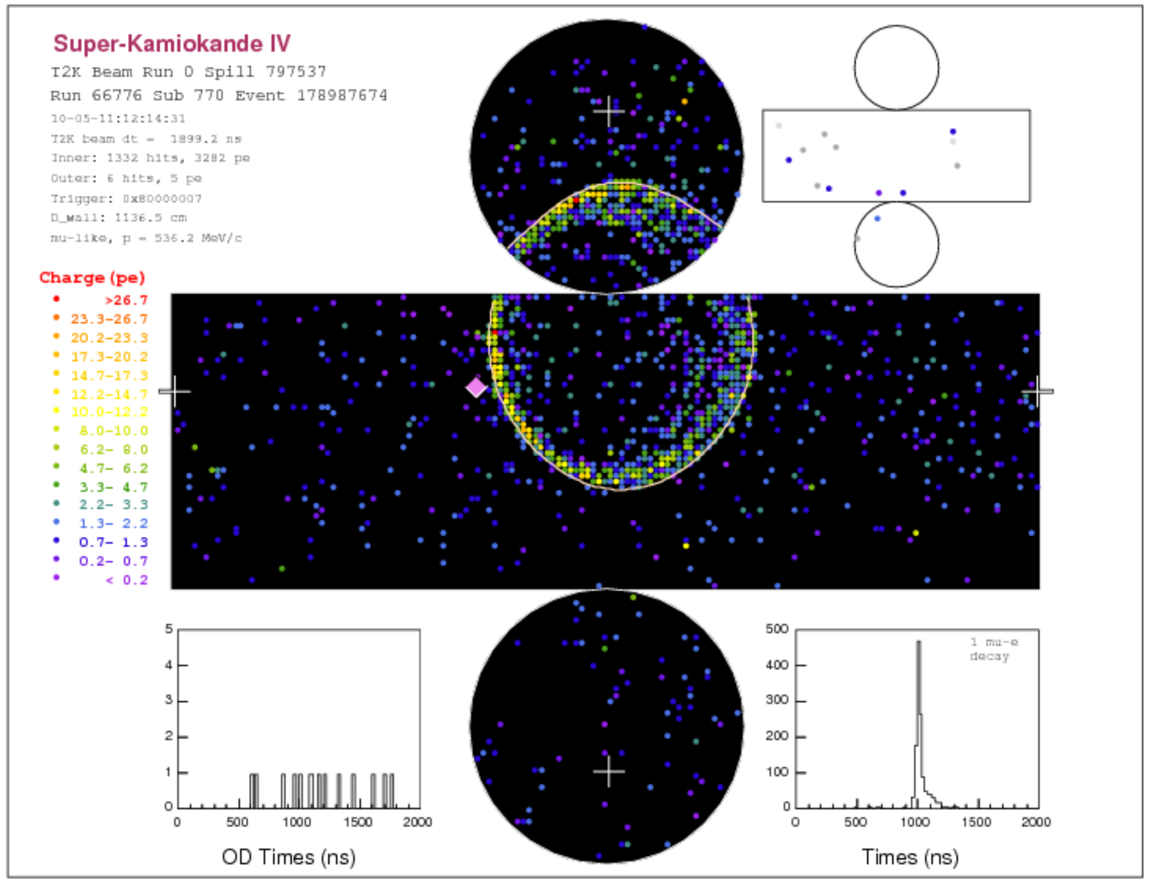
Some properties of CNN that I think are key to its success:

- Captures geometrical information of pixels
- Small number of parameters (esp. compared to a fully-connected layer)
  → trainable
- Translation invariance

When considering doing CNN for water-Cherenkov, we have two issues to solve:

1. Correct boundary condition that captures detector **topology** of 2-sphere.
2. How to encode **geometrical information**. Being in the pixel-plane next to each other has completely different meaning for barrel and endcaps.

CNN-based reconstruction



On top of this we need to make sure we can do every computation efficiently on a GPU.

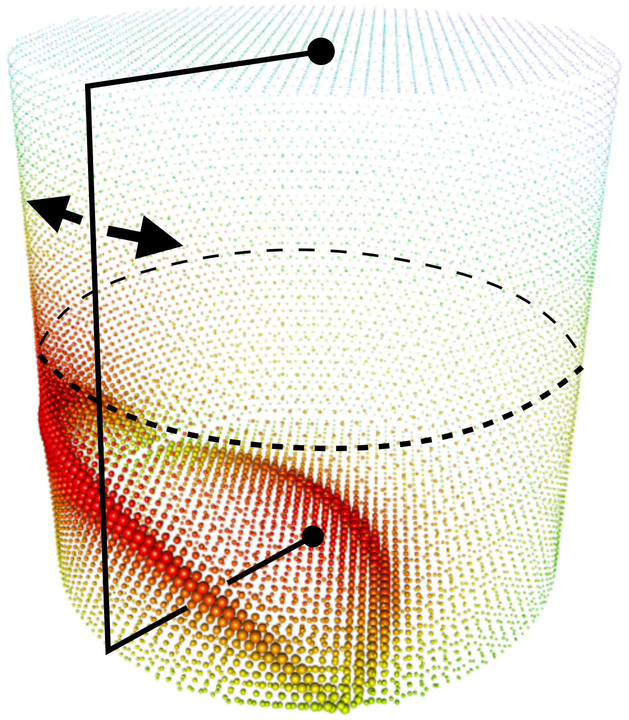
Topological map

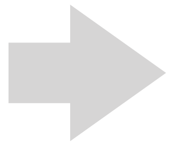
Cut open along solid line and map to square, with $W(\rho, z)$ chosen to preserve area:

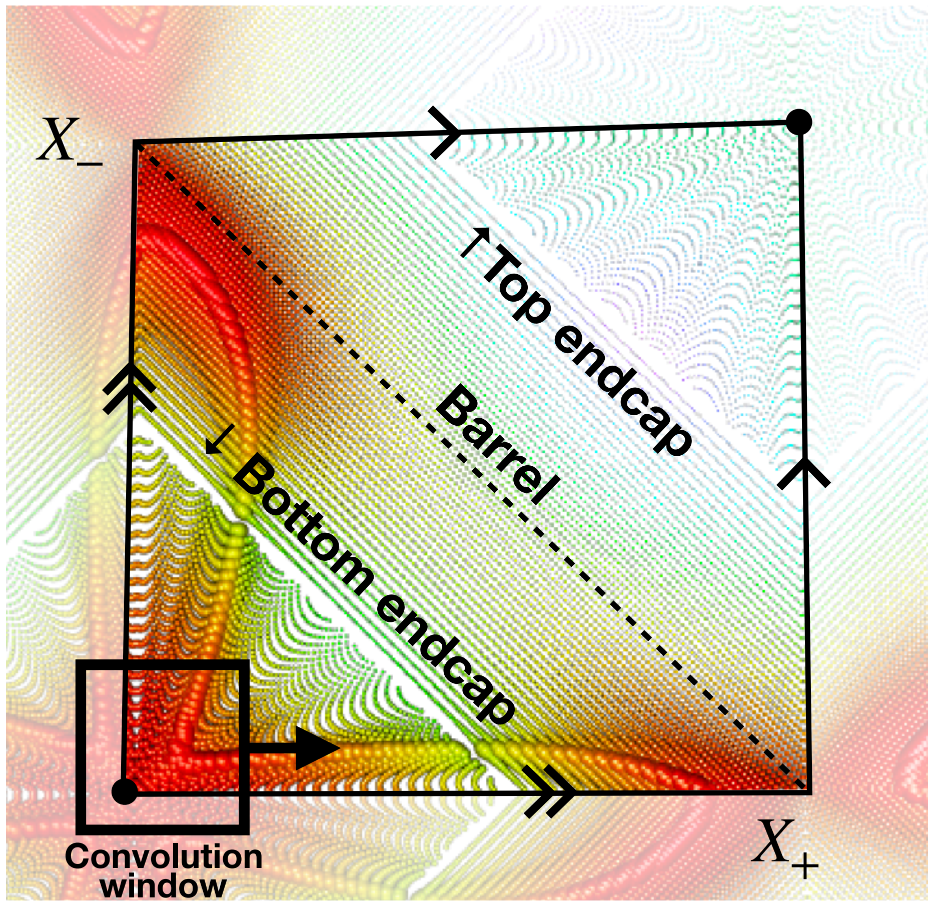
$$\mathrm{d}X_+\,\mathrm{d}X_- = \left|\frac{\partial(X_+, X_-)}{\partial(\rho, \phi)}\right| \mathrm{d}\rho\,\mathrm{d}\phi$$

$$X_\pm = W(\rho, z)\frac{\pi \pm \phi}{2\pi} \qquad W(\rho, z) = \sqrt{\frac{\rho^2 + 2Rz + RH}{R^2 + RH}}$$

Solve differential eq. for constant Jacobian







The PMTs are then put on a square grid which can be fed to machine learning libraries. Prior to convolution, pad sides according to **identification given by arrows** to embody topology of 2-sphere.

Passing geometry to convolution

inspired by gauge freedom of worldsheet coordinates in string theory

Normally convolution is performed over site indices

$$\text{output layer} \rightarrow N_i^{(n+1)} = \sum_j K_{j-i} N_j^{(n)} \leftarrow \text{input layer}$$

where we have translation invariance because the integration kernel K only depends on the relative position of the input and output layer sites (i.e. j–i)

For typical 3x3 convolution, this operation essentially is a weighted sum of moving average, 1st, and 2nd order discrete derivatives.

Analogously propose a map invariant under ***spatial* translations**:
(I, J: spatial indices)

$$N^{(n+1)}(x) = \left(K + K^I \partial_I + \frac{1}{2} K^{IJ} \partial_I \partial_J \right) N^{(n)}(x)$$ where the **Taylor expansion coefficients** are learned: K, K^I, and K^{IJ} (scalar, vector, …)

Passing geometry to convolution

The Taylor expansion can be implemented as convolution on the 2D grid by inserting precomputed weights w:

$$N_i^{(n+1)} = \sum_j \left(K w_{ij} + K^I w_{I,ij} + \frac{1}{2} K^{IJ} w_{IJ,ij} \right) N_j^{(n)}$$

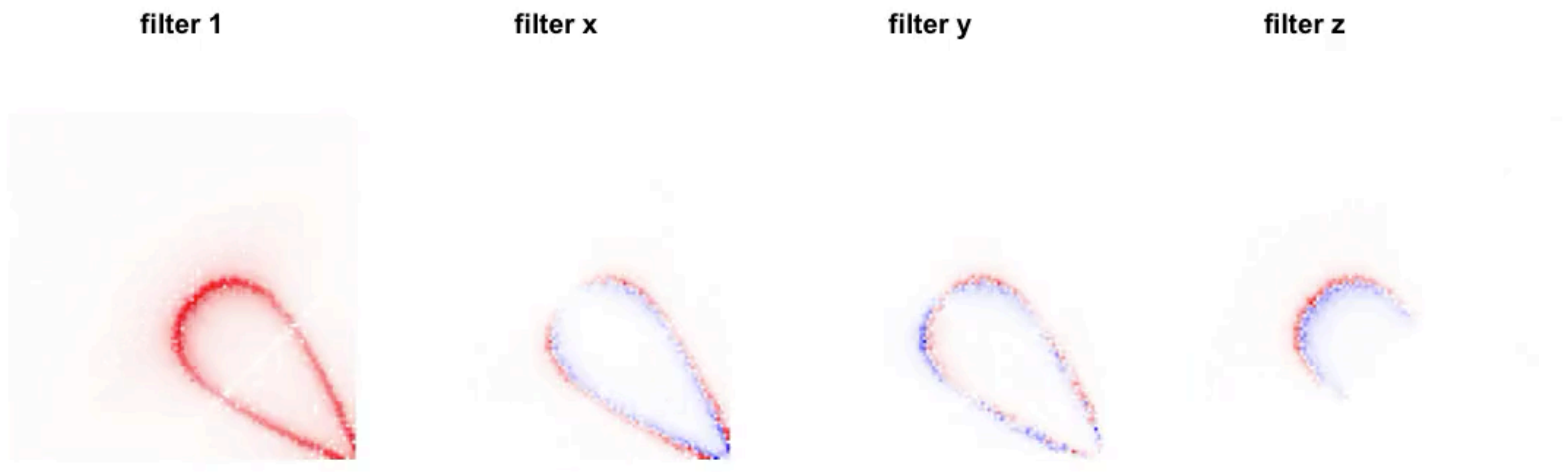
For each site i weights w, w_I, w_{IJ} defined to obtain spatial derivatives by summing over neighboring grid sites j:

$$f_i = \Sigma_j w_{ij} f_j$$
$$\partial_I f_i = \Sigma_j w_{I,ij} f_j$$
$$\partial_I \partial_J f_i = \Sigma_j w_{IJ,ij} f_j$$

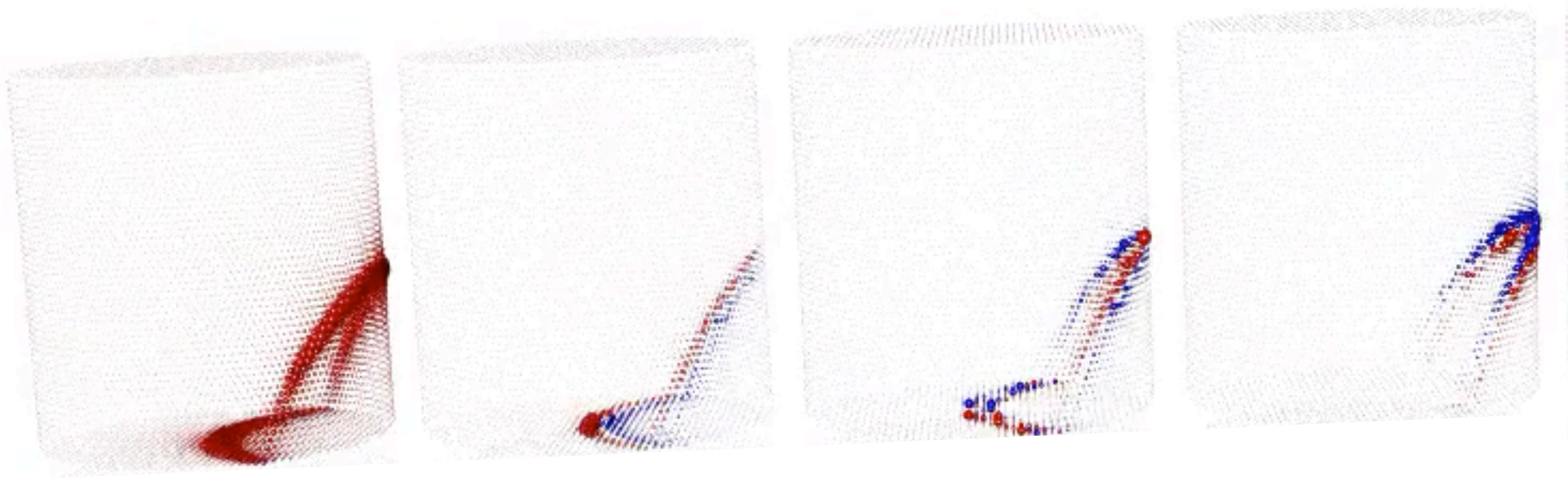
where $f_i := f(x_i)$

Analogously propose a map invariant under ***spatial* translations**:
(I, J: spatial indices)

$$N^{(n+1)}(x) = \left(K + K^I \partial_I + \frac{1}{2} K^{IJ} \partial_I \partial_J \right) N^{(n)}(x)$$ where the **Taylor expansion coefficients** are learned: K, K^I, and K^{IJ} (scalar, vector, …)



θ-motion



Dataset

- Super-K simulated in WCSim [2,3]
- 50k events for {e⁻, μ⁻, π⁺} each
- Uniform vertex distribution
  (inside inner detector)
- Isotropic direction
  (particles might leave the detector = no FC cut)
- Flat energy from 1 MeV to 1 GeV
- Only using charge information for now

Optimization target:
classification of three particle types (PIDs)
(cross-entropy)

Implemented in PyTorch on GPU. The Taylorizing and random rotations introduce ~10x computing overhead.

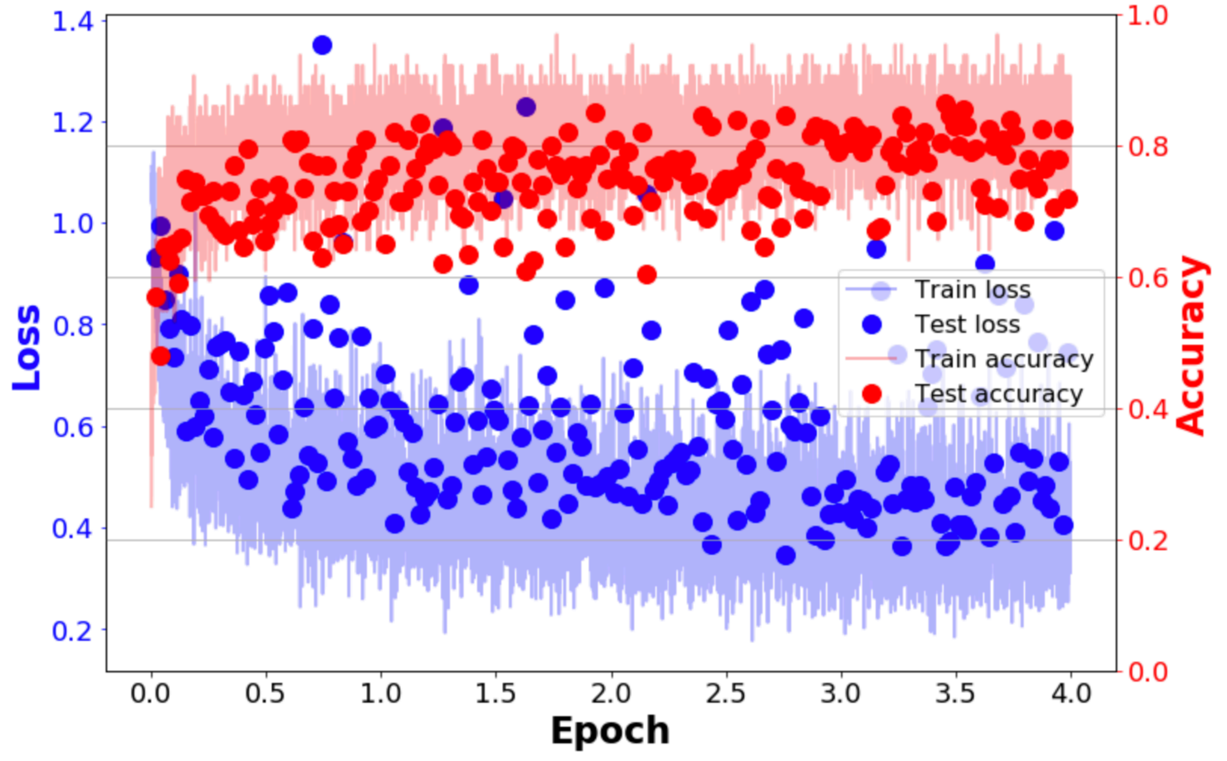
PID performance study

- About 80% PID accuracy (e,μ,π⁺).

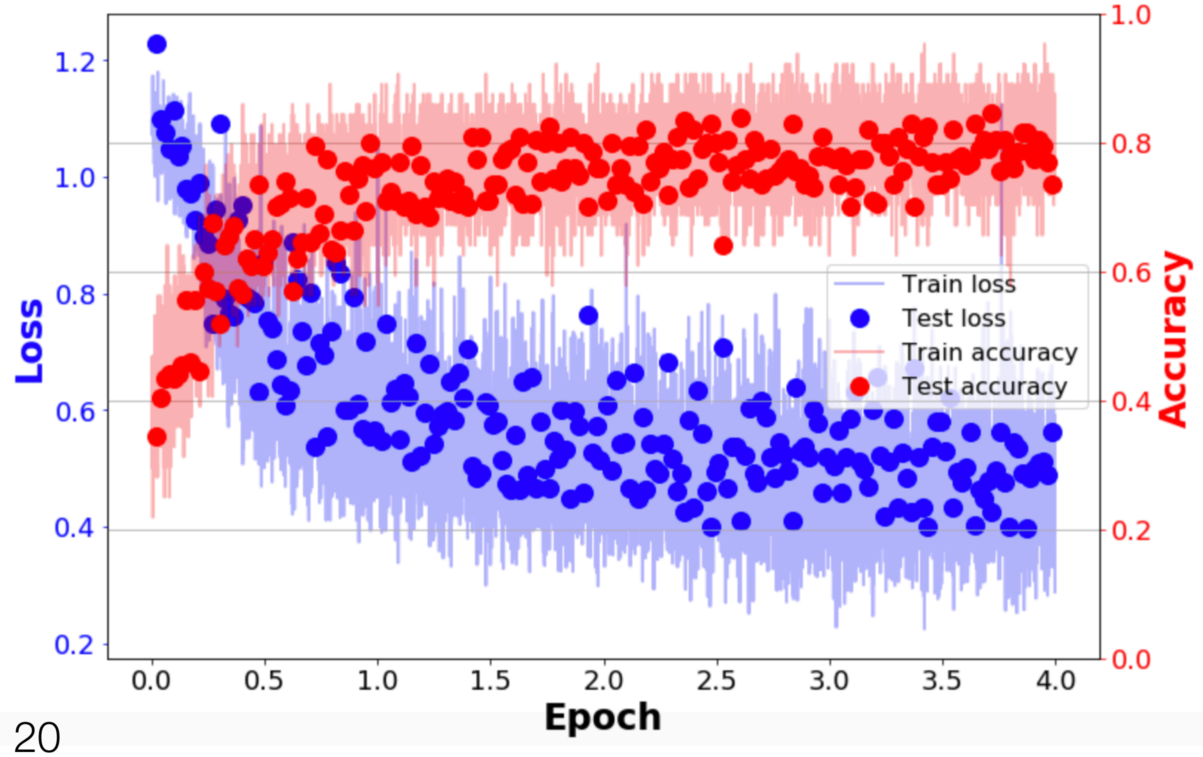
- At this stage, this Taylor-decomposition technique is **not *that* much better**
  (it *does* seem to somewhat **generalize better?**)

- I wonder how this looks like with position/direction reconstruction which are inherently geometric

Ordinary convolution on topological map



TaylorConv



Effective rotation

For example for a scalar input layer $N^{(n)}(x) \to N^{(n)}(Rx)$, demanding the output layer $N^{(n+1)}(x)$ to be scalar as well, the Taylor expansion coefficients have a clear **geometrical meaning**:

$K \to K$	scalar
$K^I \to R^I{}_M K^M$	vector
$K^{IJ} \to R^I{}_M R^J{}_N K^{MN}$	tensor

this can be exploited to impose e.g. the **ϕ-rotation symmetry** of the tank upon the network by inserting a random rotation R about the z-axis during the training, effectively making the training set larger:

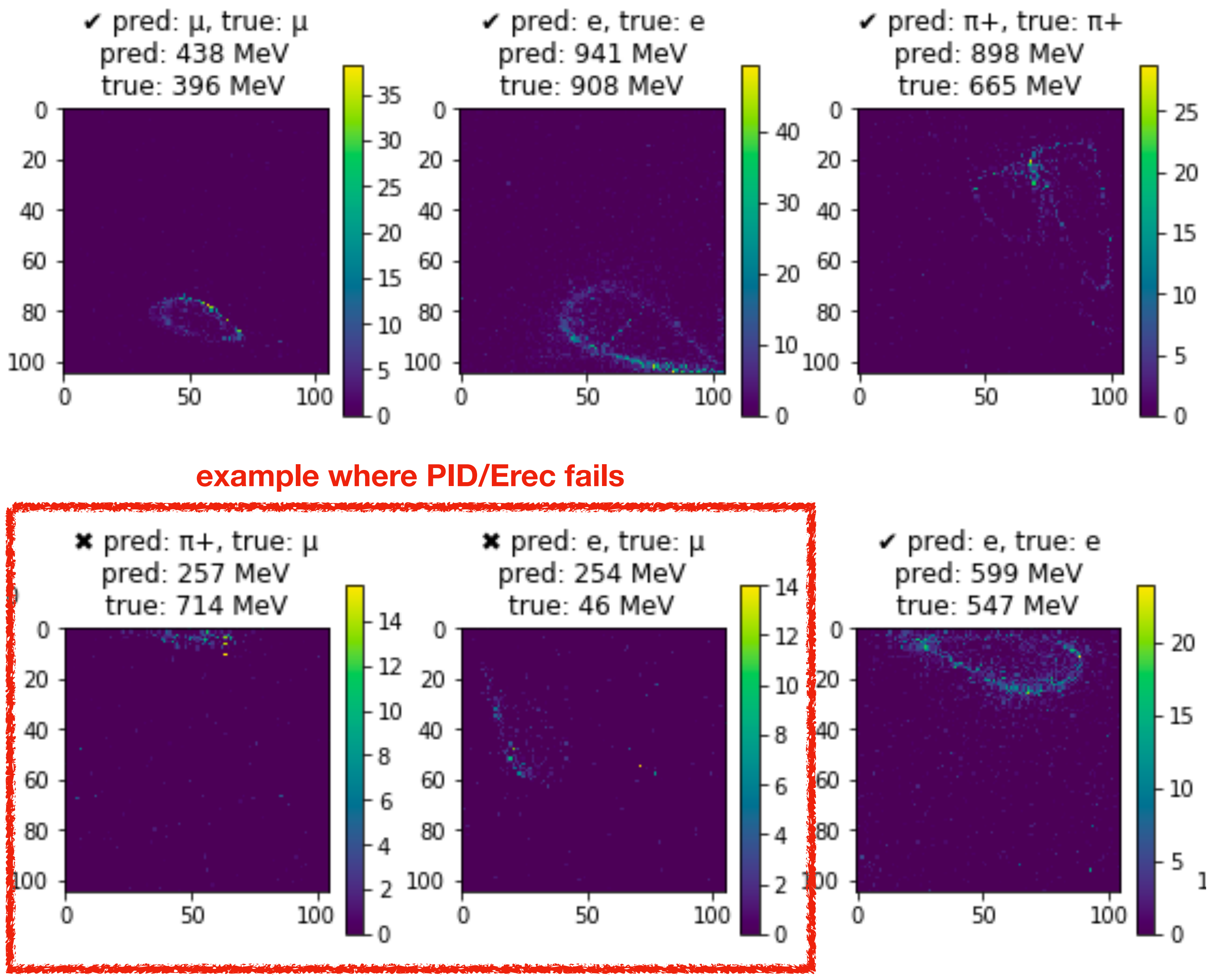
$$N_i^{(n+1)} = \sum_j \left(K w_{ij} + R^I{}_M K^M w_{I,ij} + \frac{1}{2} R^I{}_M R^J{}_N K^{MN} w_{IJ,ij} \right) N_j^{(n)}$$

Any vectorial variables (vertex, direction) to compute the loss against would need to be rotated too. It might be interesting to allow hidden channels to have vector indices too.

Tried training using **1000 events only.** With rotations it seems **overtraining is suppressed** to some extent.

→ given small stats of neutrino samples, such techniques might be valuable for **data-driven** training

Toward full reconstruction



- Started experimenting toward full reconstruction, here PID + Energy reco., how to combine loss functions etc.
- Still far from maximum-likelihood fitters but fun
- Can we beat maximum-likelihood fitters in the future? I think yes, but need to find data-driven methods in order not to rely on MC

Summary

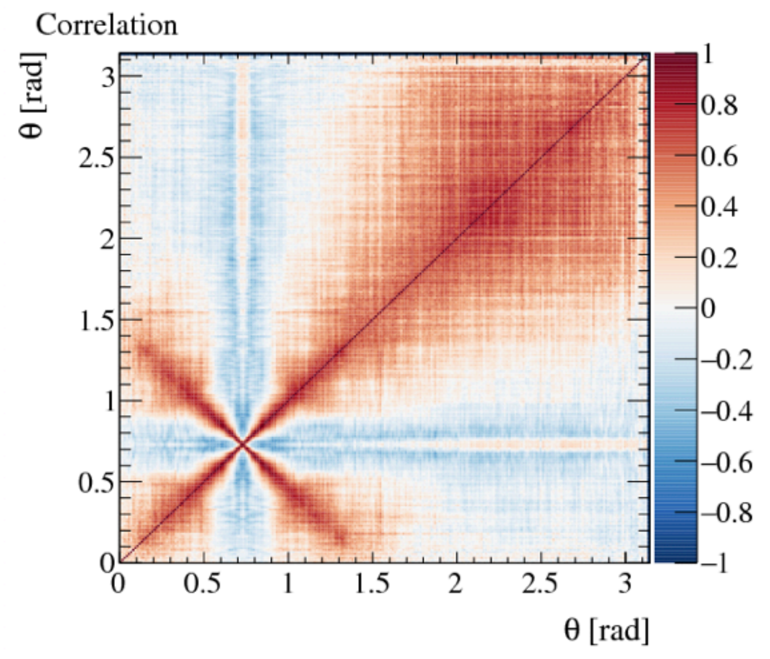
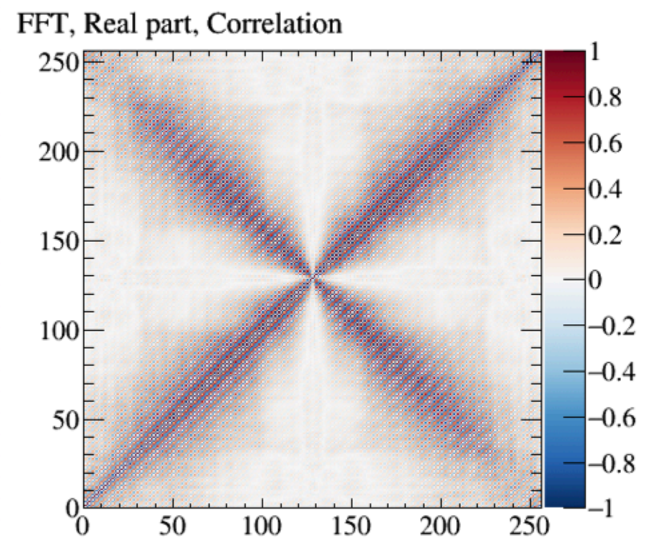
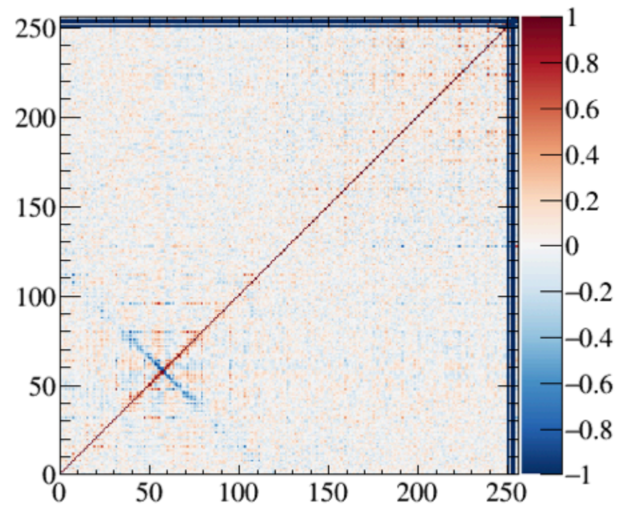
- Ring reconstruction in water-Cherenkov detectors might benefit from modeling the PMT-correlations due to Cherenkov profile variations
- Working on two approaches (maximum-likelihood and CNN), so far no significant improvement to conventional methods but development is ongoing.
- Proposed CNN architecture for topology and geometry of cylinder, might be generalizable to other applications (mostly embedded sub-manifolds?)
- In general, should be careful whether correlations in MC can be trusted. **Data-driven** approaches need to be considered.

Ideas welcome!

backup

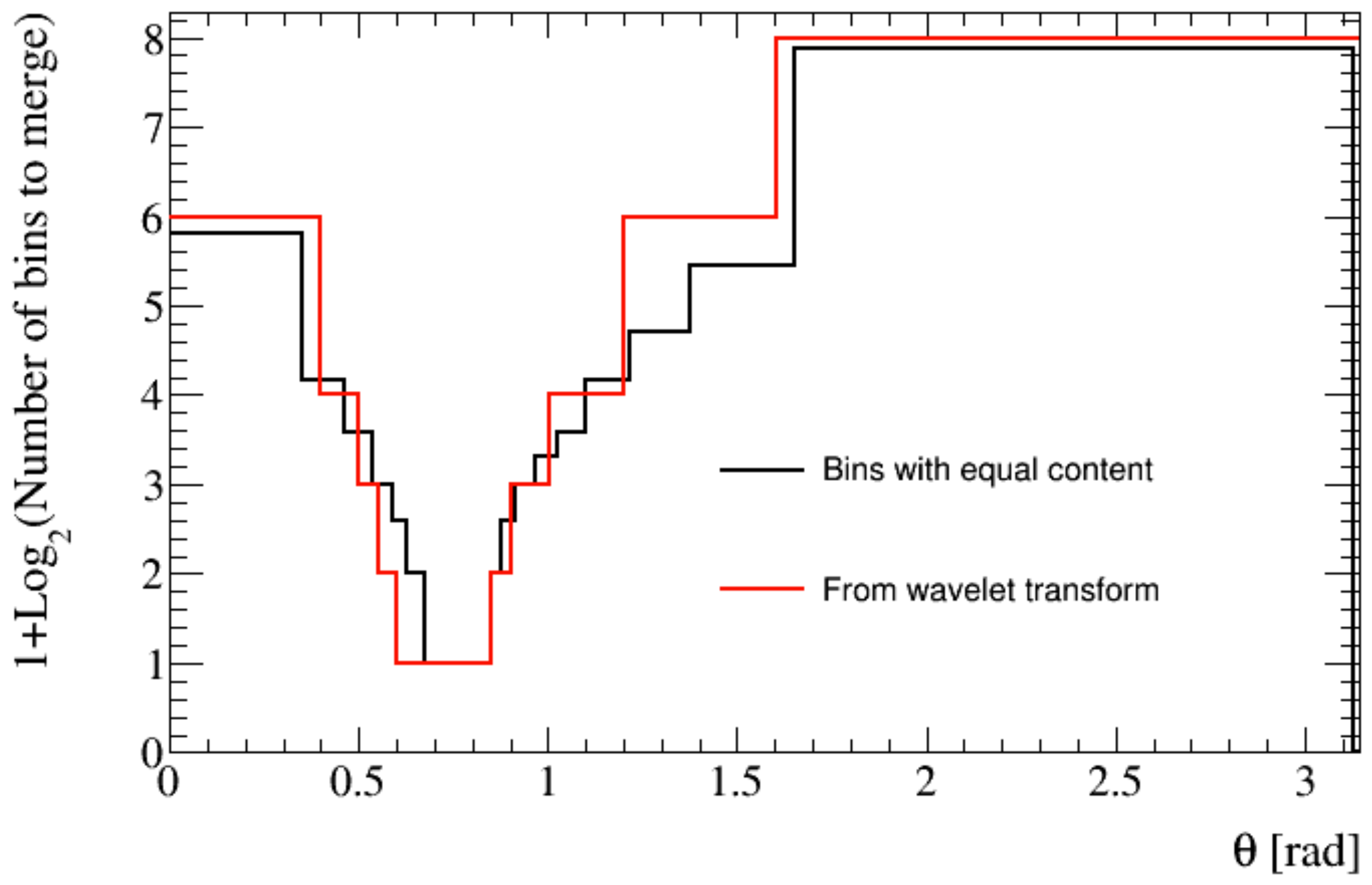
Find good basis

- For practical reasons it would be nice to have a basis with sparse correlation matrix (to ignore off-diagonals)
- Wavelet transform seems best (blue lines at >250 are artifact because original bin count was not power of 2)

Original	FFT (Real part)	Wavelet
		

Find good bin width

- Ignoring off-diagonals in wavelet transform = combining neighboring bins → suggest binning scheme by combining pairs with most negligible corr.
- Turns out simply choosing binning by requiring equal number of photons in each bin gives almost the same result → much easier, so let's use this



Advanced symmetry structure

- Further expand each Fourier coefficient into magnitude and complex phase angle: $f_m(\theta) = |f_m(\theta)|\, e^{-im\alpha_m(\theta)}$

- Redefinition of azimuthal origin is gauge freedom, f should not change:

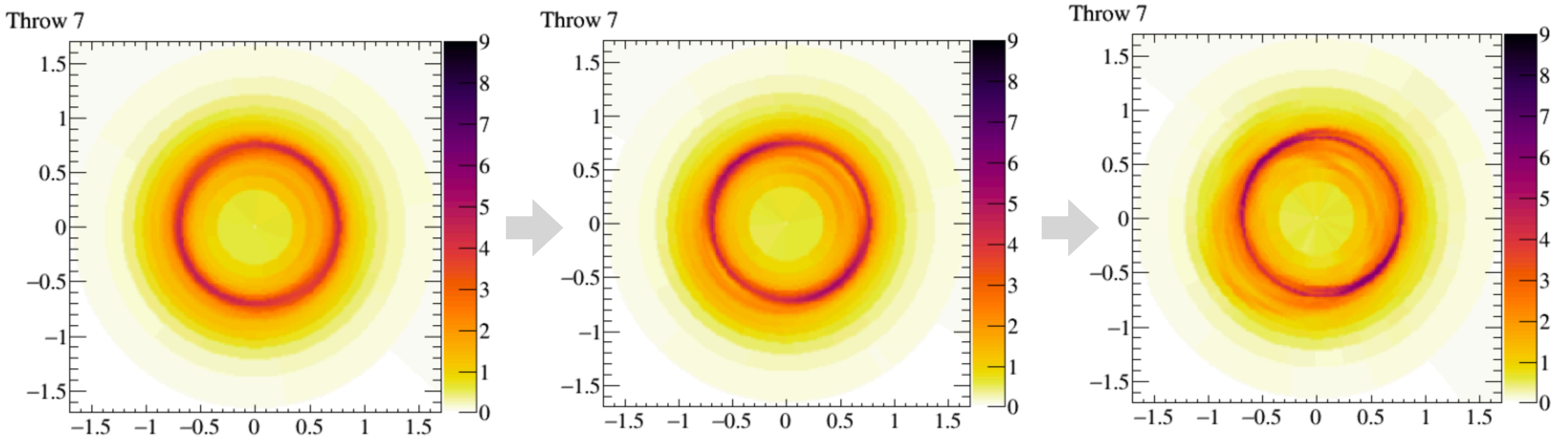
  $\phi \to \phi' = \phi + \delta$

  $f \to f' = f$ iff $\alpha_m(\theta) \to \alpha_m(\theta) + \delta$

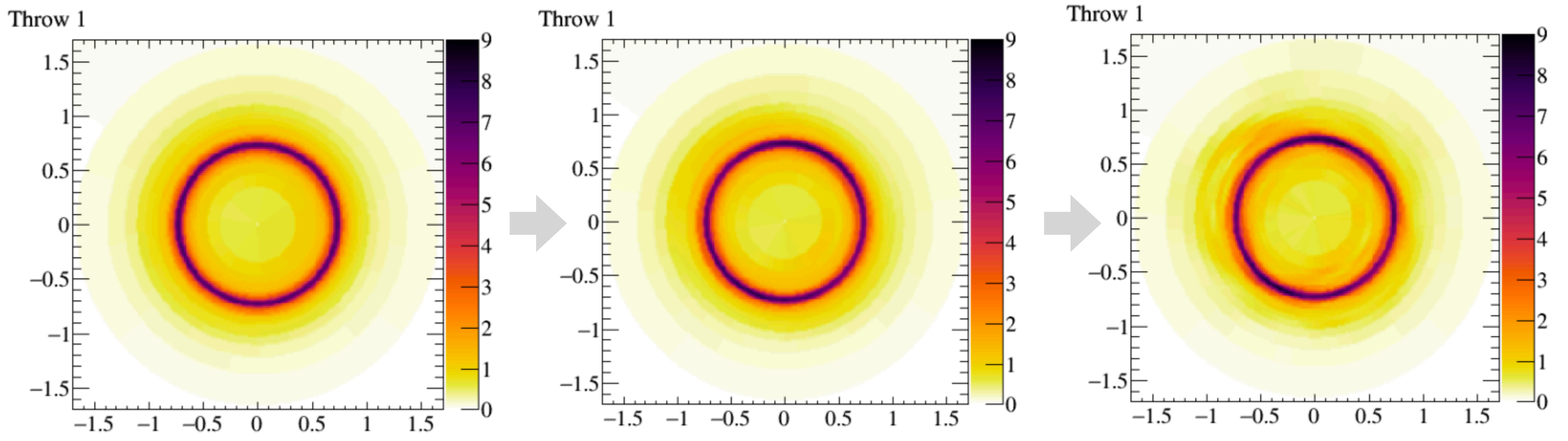
  - $\mathbb{E}[\,\cdot\,] = 0$ unless gauge-singlet. For example:
  - $\mathbb{E}[\,|f_m|\,|f_{m'}|\,]$ is singlet, can be non-zero for any m, m' and θ, θ'
  - $\mathbb{E}[e^{inm\alpha_m} e^{-in'm'\alpha_{m'}}]$ is singlet if $nm = n'm'$, in which case it can be non-zero for any θ, θ'. e.g. take $nm = \mathrm{LCM}(m, m')$

- Assuming parity invariance (no magnetic field etc.), we further have
  - $\mathbb{E}[\sin(nm\alpha_m - n'm'\alpha_{m'})] = 0$, i.e. only cosine correlations

$m \leq 1$ $m \leq 2$ $m \leq 10$

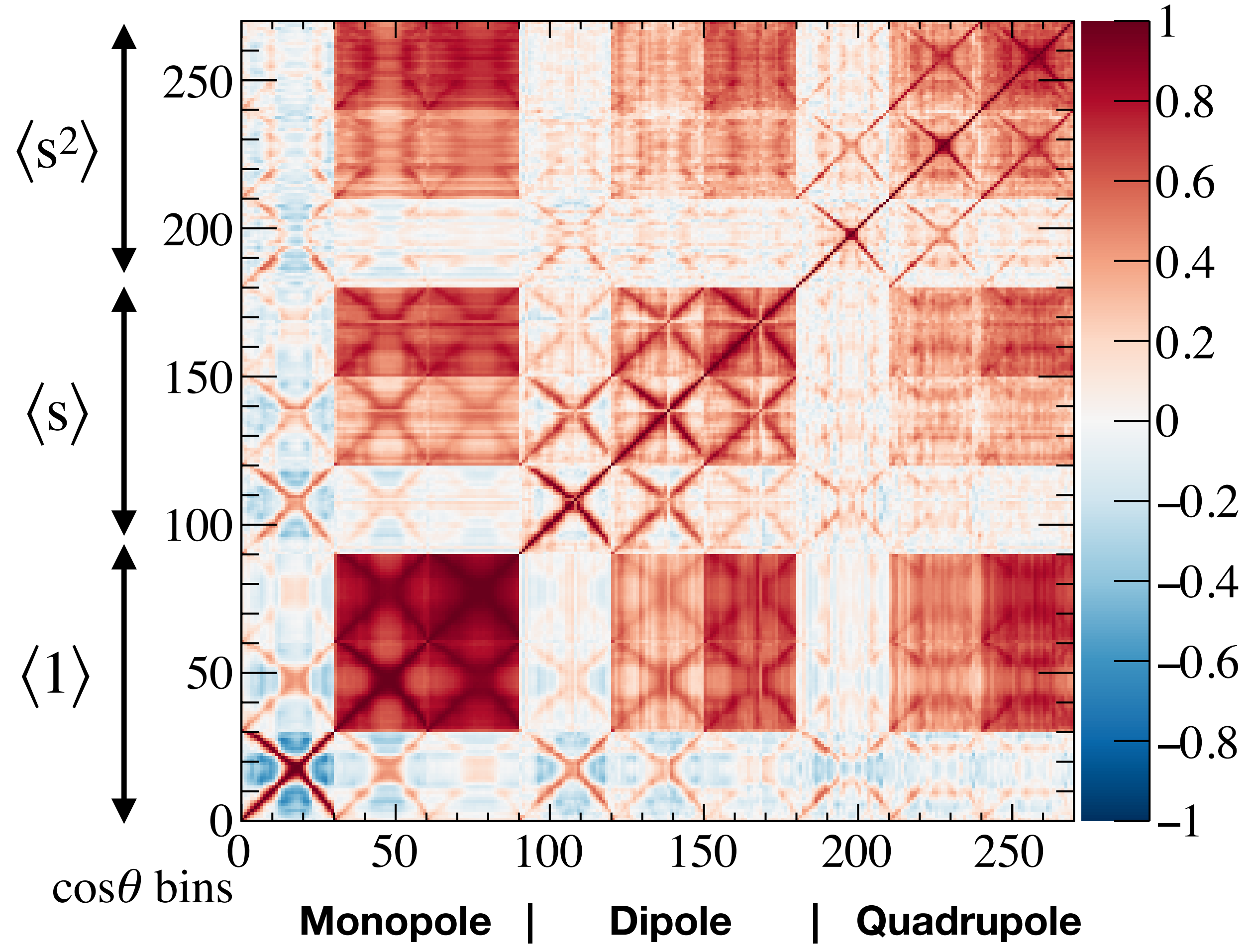


Need at least quadrupole variations to see ring-like structures of variations.
Correlations between multipoles is essential.



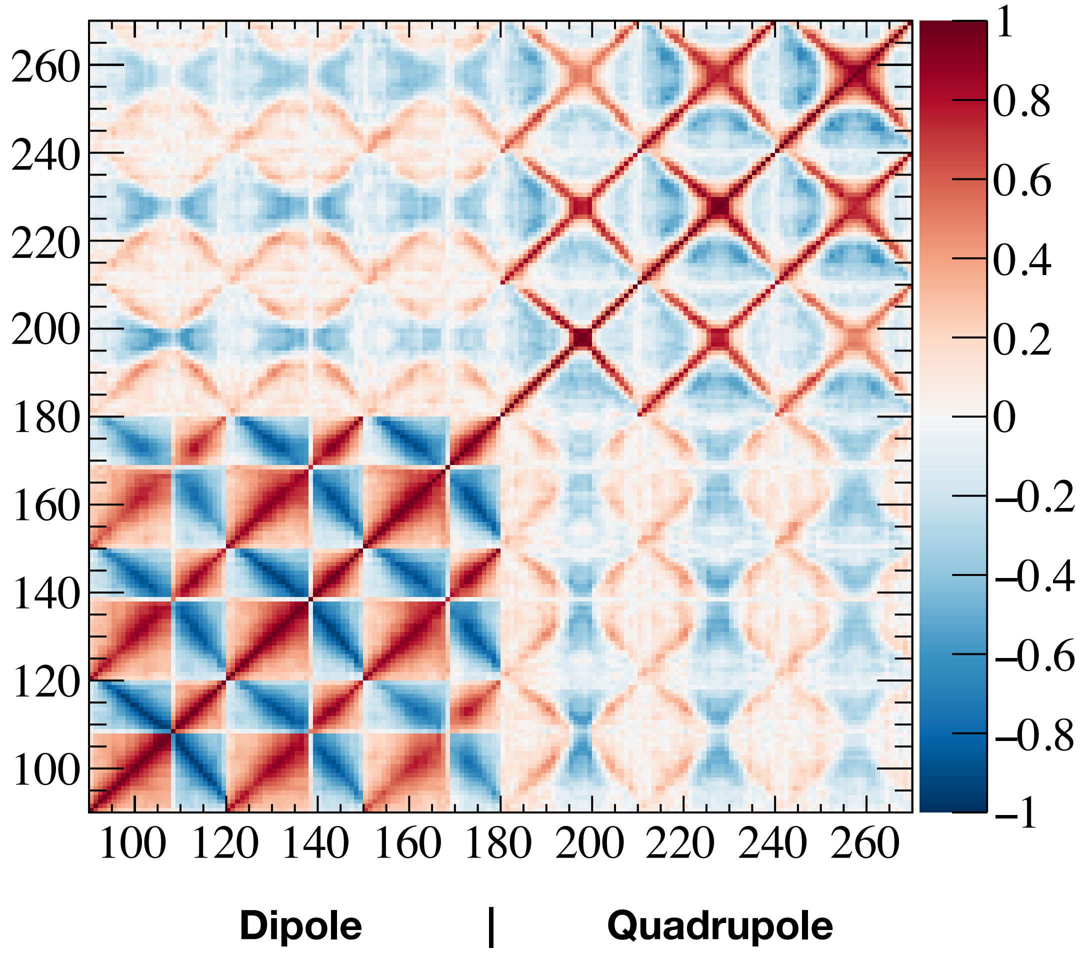
Correlations between multipoles

$$\mathrm{Cor}\left[\log|f_m|,\ \log|f'_{m'}|\right]$$

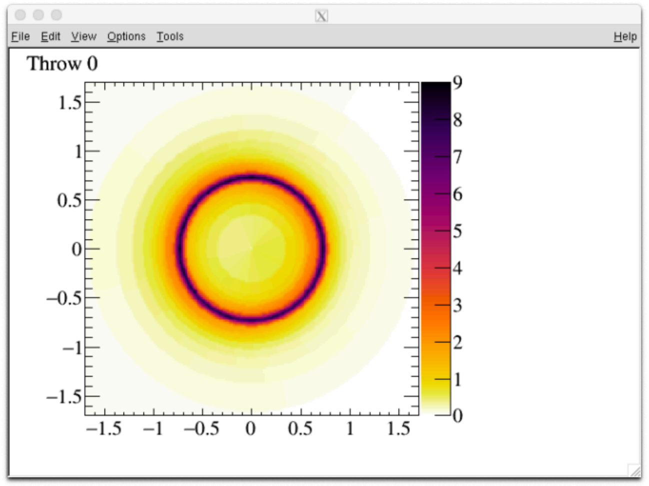


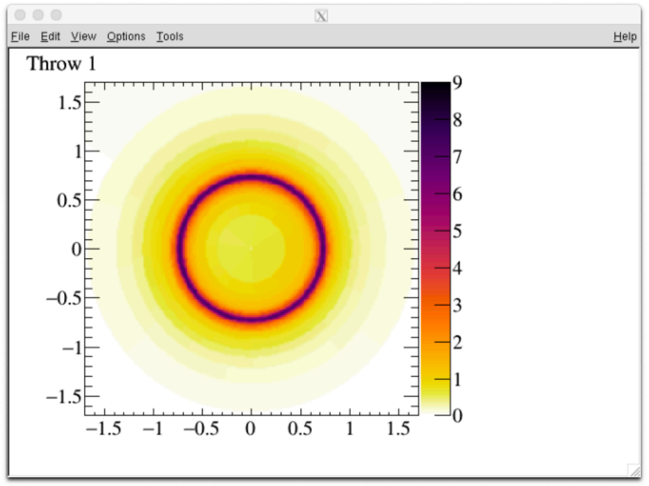
$\mu := \mathrm{LCM}(m, m')$

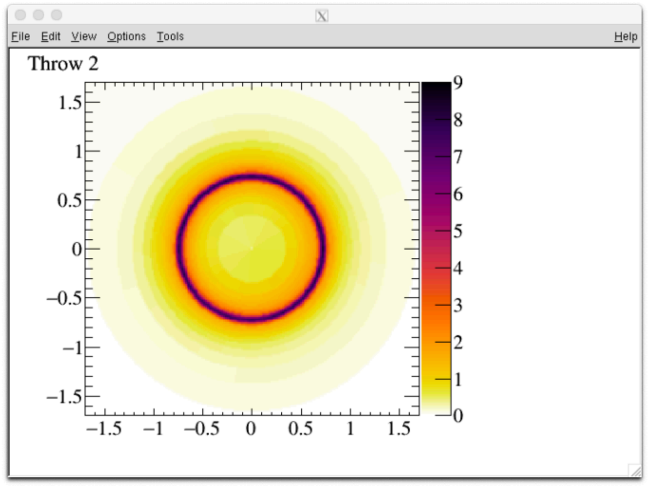
$$\mathbb{E}\left[\cos\mu(\alpha_m - \alpha'_{m'})\right]$$

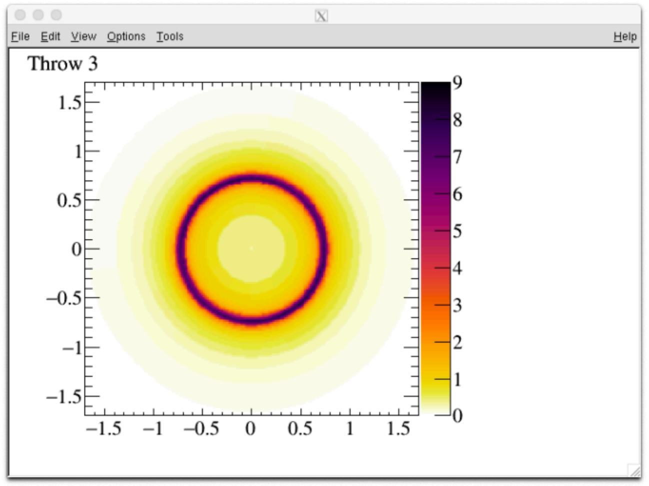


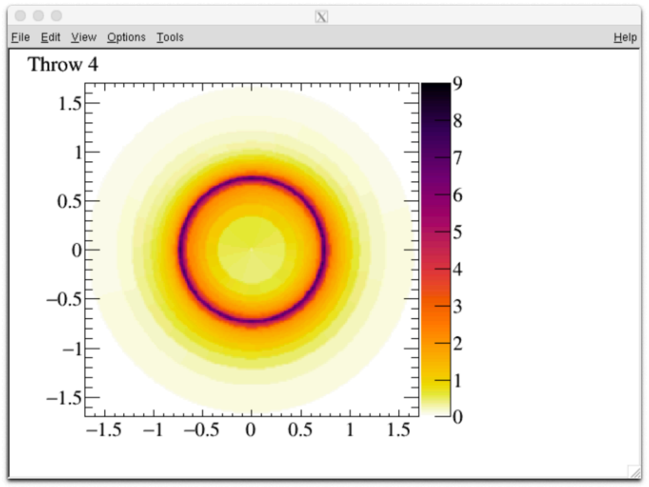
m ≤ 1 throws

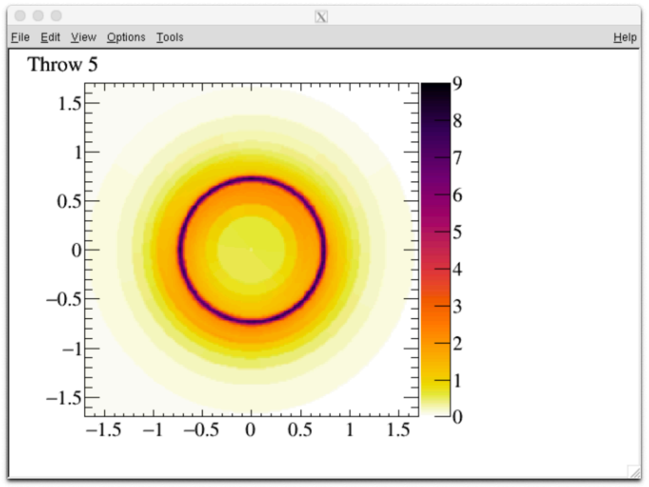

File Edit View Options Tools Help
Throw 0

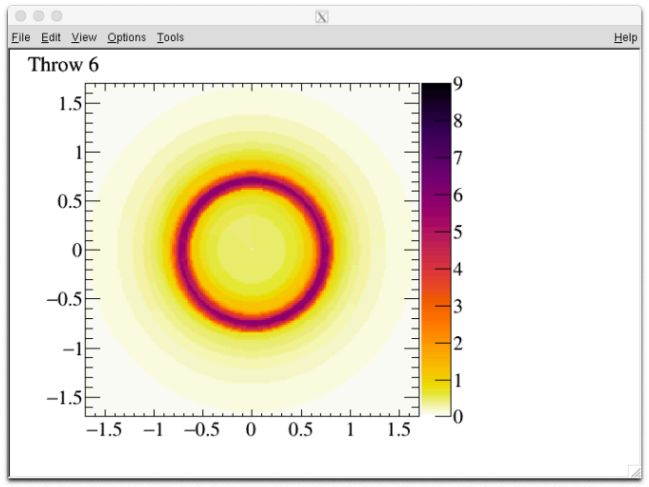

File Edit View Options Tools Help
Throw 1

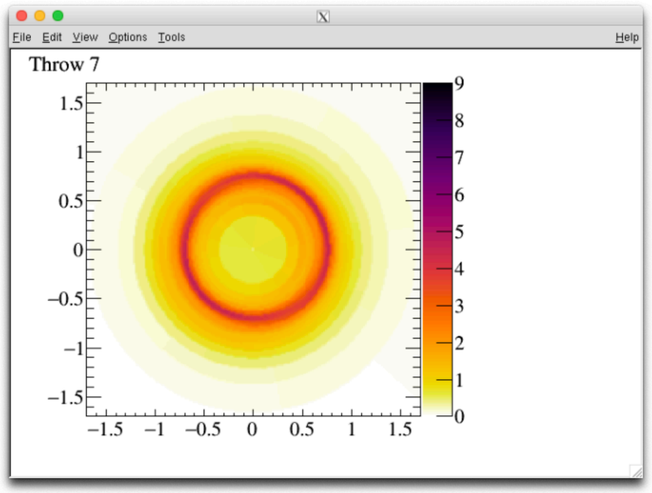

File Edit View Options Tools Help
Throw 2


File Edit View Options Tools Help
Throw 3

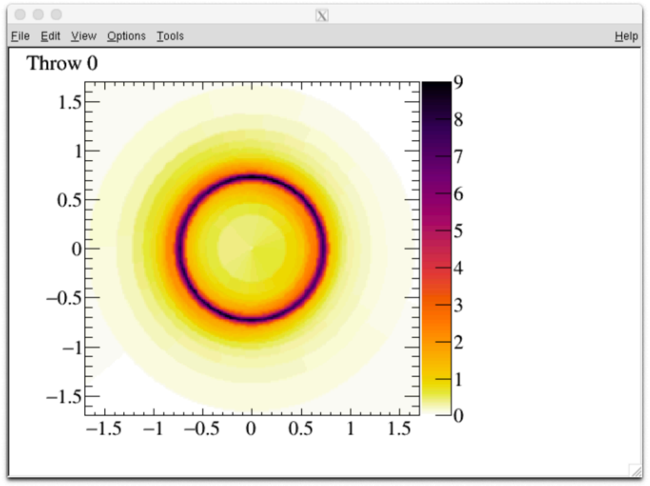

File Edit View Options Tools Help
Throw 4

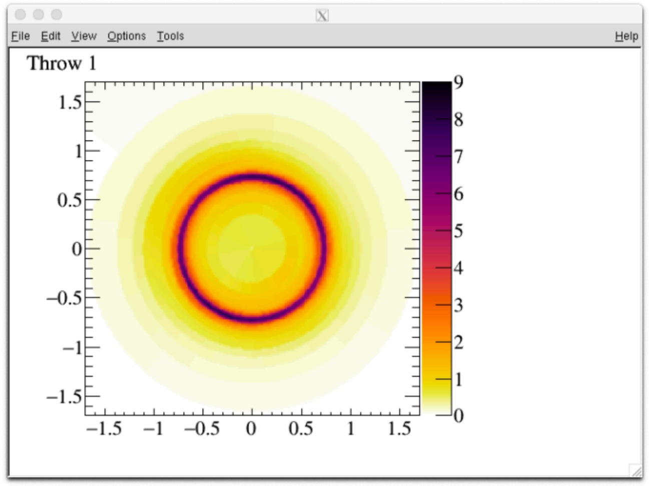

File Edit View Options Tools Help
Throw 5

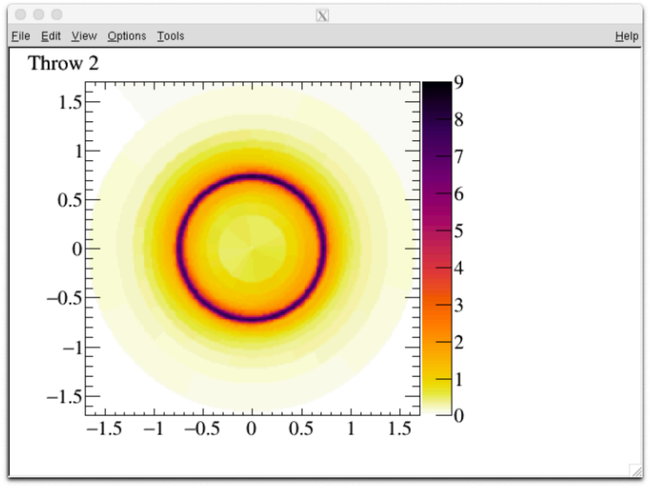

File Edit View Options Tools Help
Throw 6

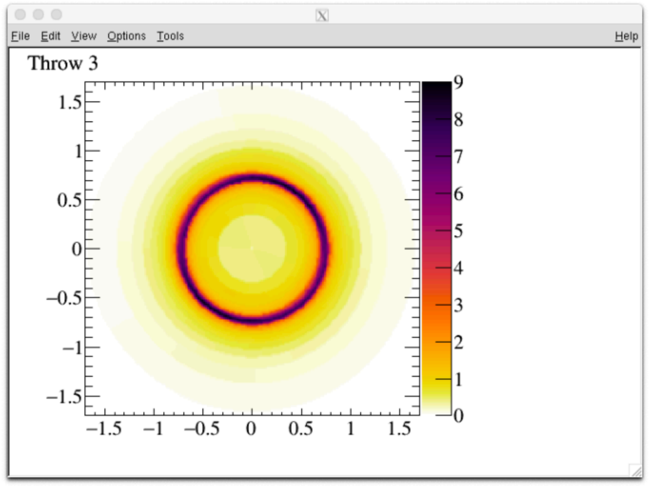

File Edit View Options Tools Help
Throw 7

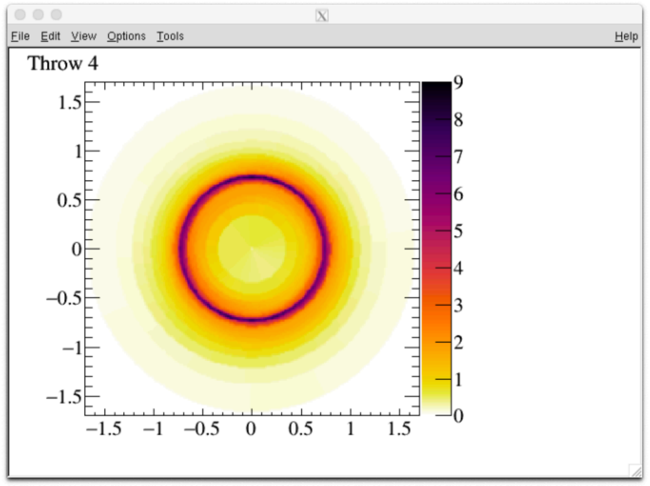
m ≤ 2 throws

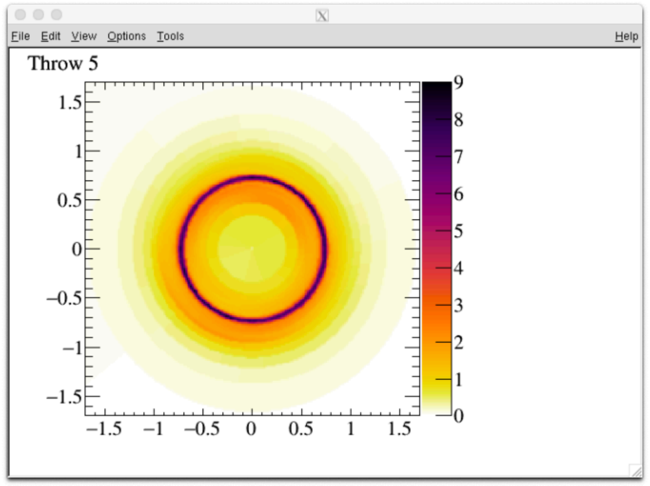


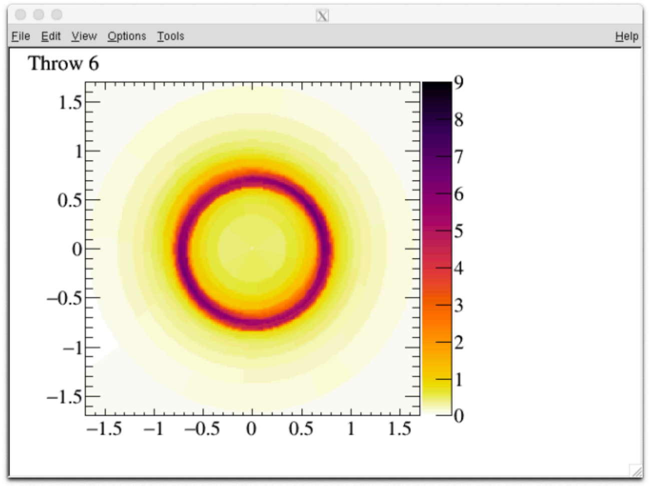


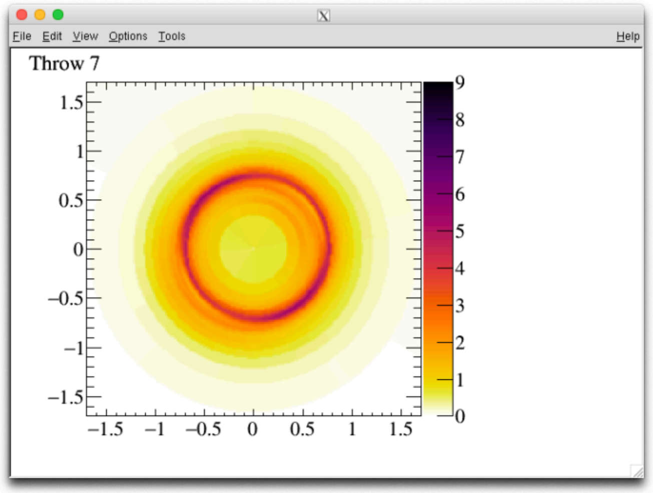




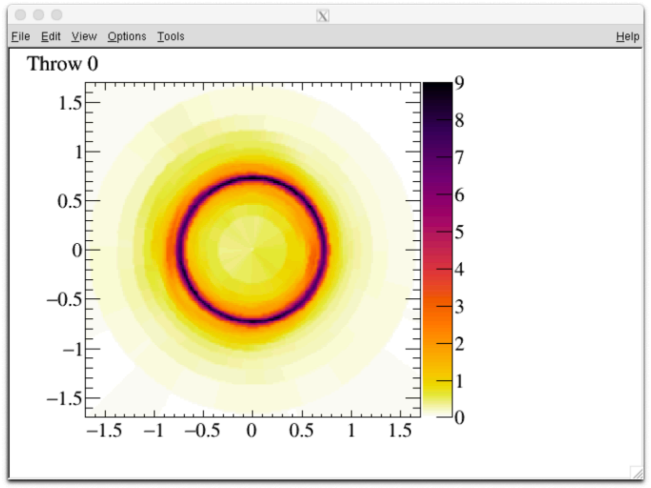


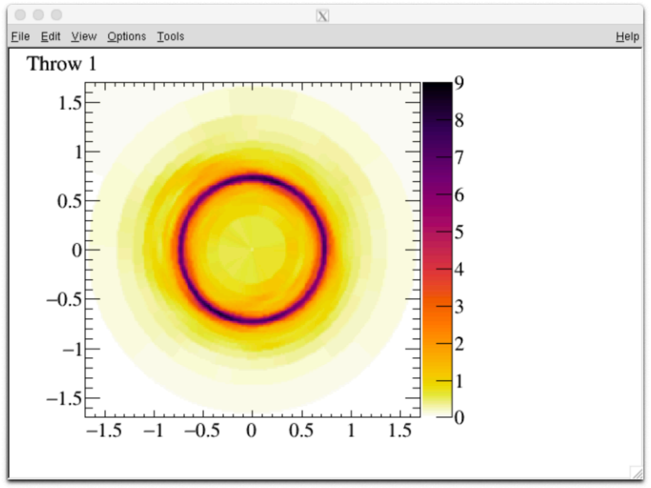


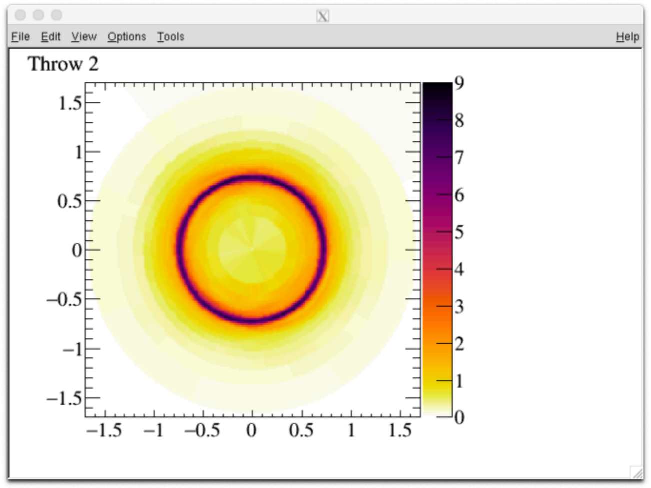


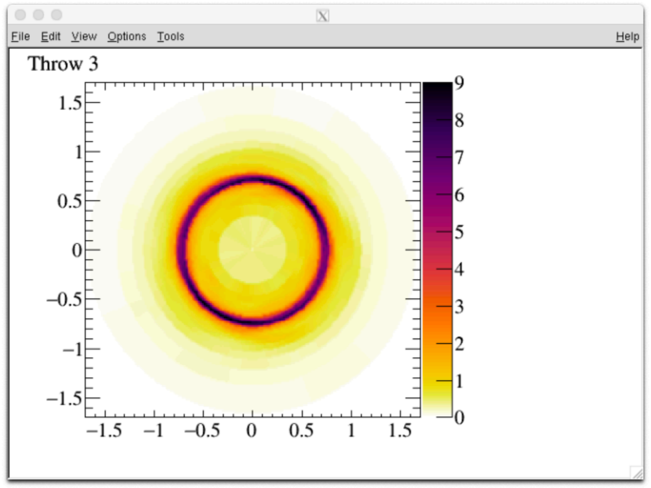


m ≤ 10 throws

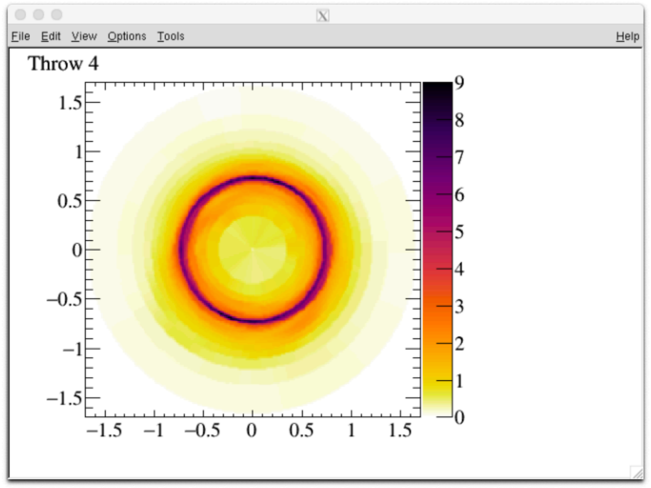


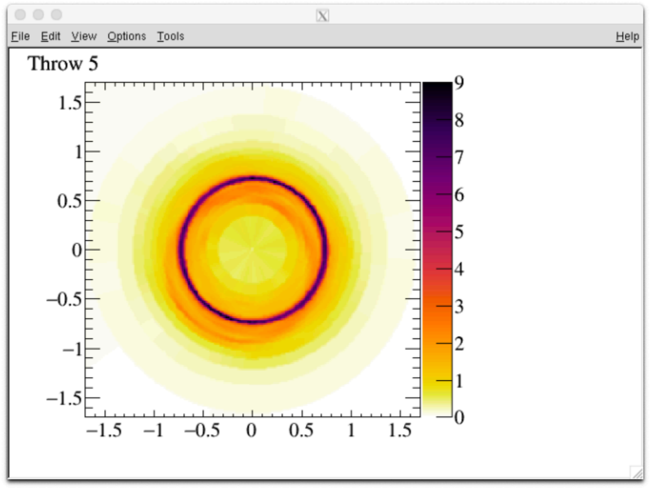


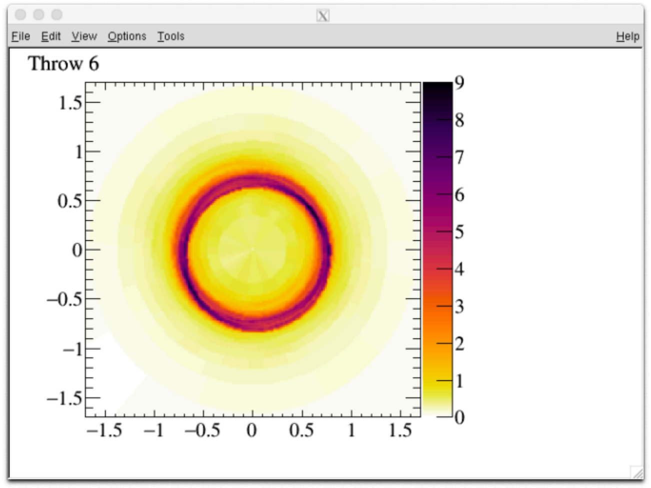


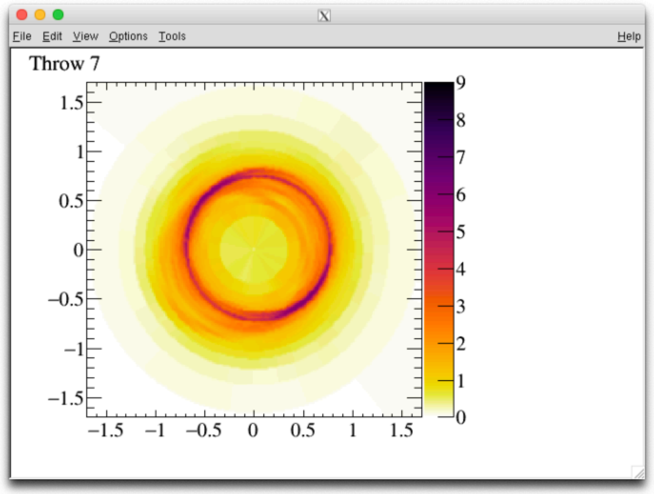


the nice thing about marginalization is we can easily implement these without much change



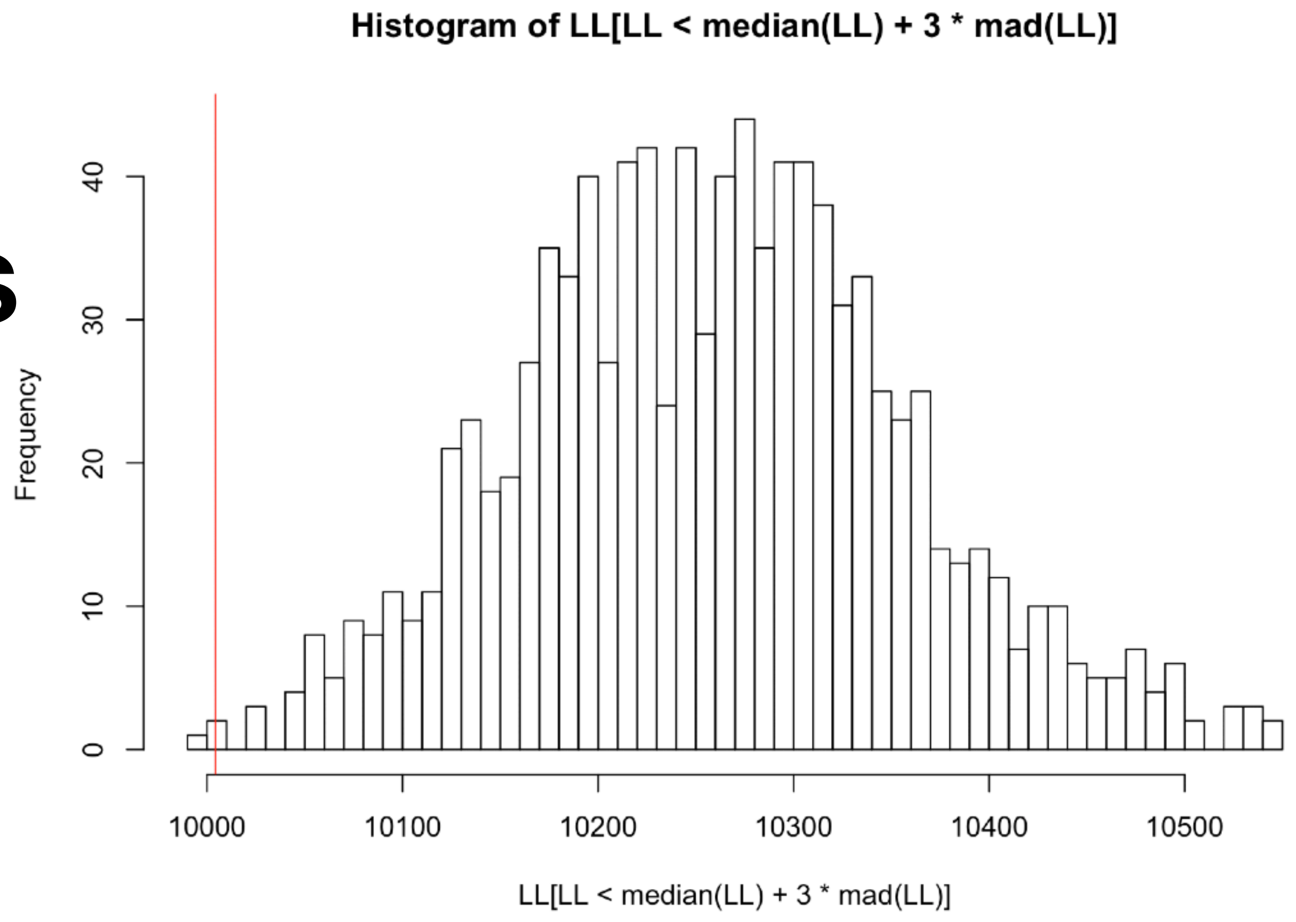


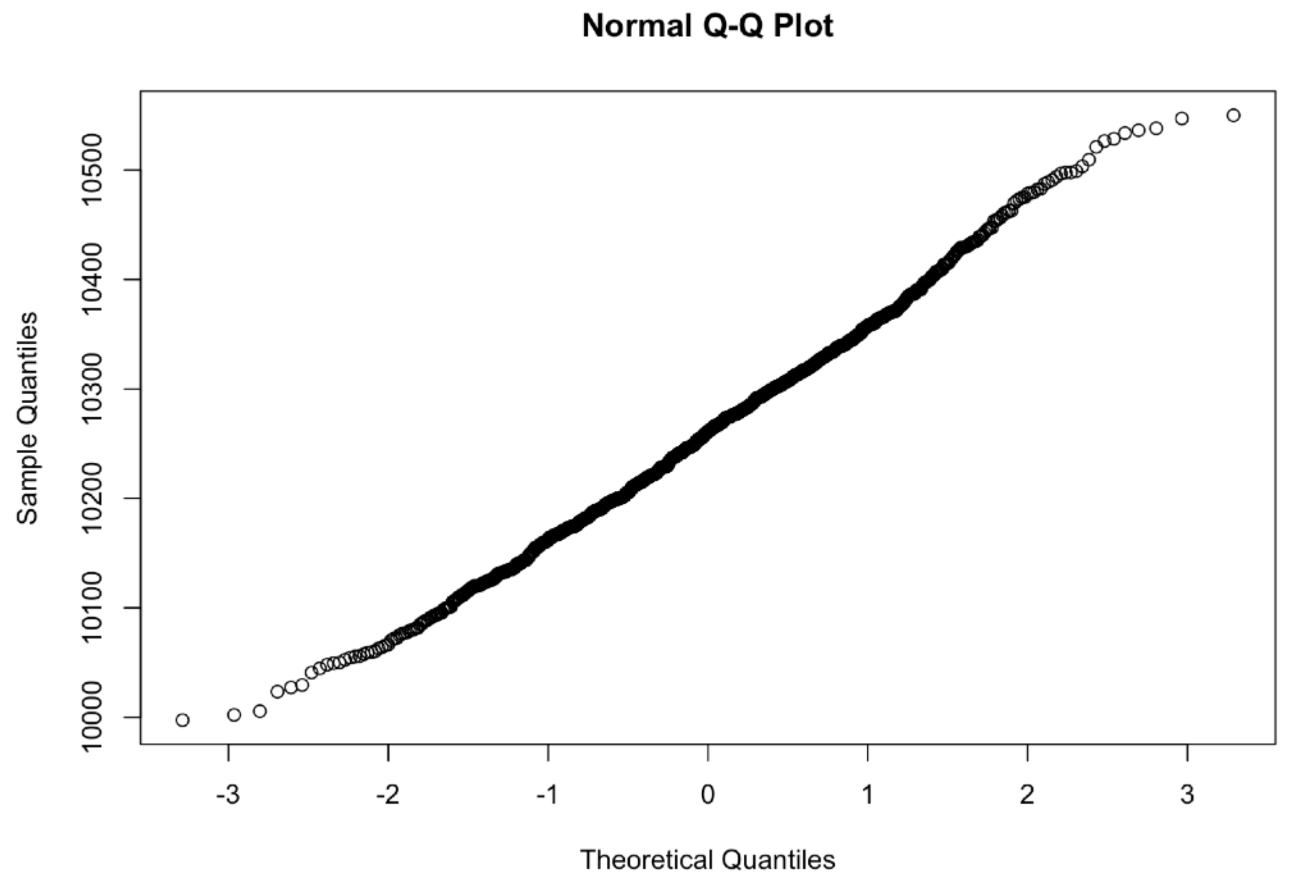




Distribution of log-likelihoods for throws

- 1000 throws with m≤1, no s-dependence
- Distribution is quite normal with somewhat shorter tails (not sure if this is expected)
- problem: –logL varies by orders of ~100!
  → marginalized –logL (red line) is dominated by very few throws with small –logL
- three interpretations:
  1. I made a mistake with calculation
  2. need very large number of throws
  3. this is really a problem where we should be fitting (profiling), the data contains "too much" information about these variations
- However I should note this is the distribution at the initial point for the likelihood fit found with grid search. Maybe at true values or best fit the spread is a bit smaller.





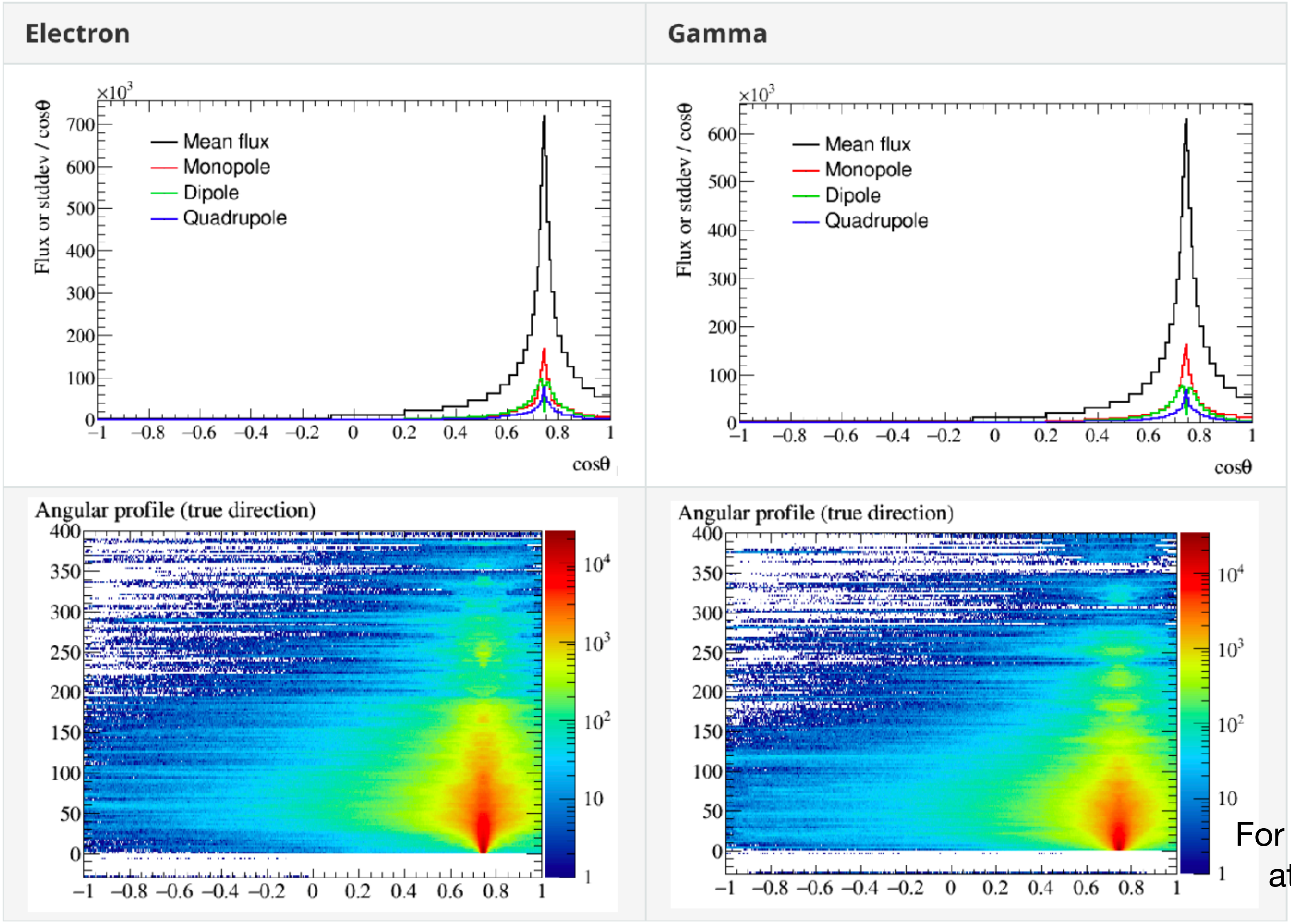
Analytic marginalization

- If –logL distribution is normal, we could simply measure mean and variance (ideally with robust methods) and compute analytically
- Could be interesting for a SGD like method where we compute a running mean of mean and variance (normal marginalization has the nonlinear exp,log)
- Another idea: introduce notion of inverse temperature and perform annealing?

$$L_m(x\,|\,\beta) := \frac{1}{\beta}\,\mathbb{E}\left[e^{-\beta\Lambda}\right]$$

$$\begin{aligned}
\mathcal{L}_m(x) &:= \int \mathrm{d}\eta\, \mathcal{L}(x|\eta)\, \mathcal{P}(\eta) \\
&= \int \mathrm{d}\eta\, e^{\log \mathcal{L}(x|\eta)}\, \mathcal{P}(\eta) \\
&= \frac{1}{N_{\mathrm{throws}}} \sum_{i\in \mathrm{throws}} e^{\log \mathcal{L}(x|\eta_i)} \\
&= \frac{1}{N_{\mathrm{throws}}} \sum_{i\in \mathrm{throws}} e^{-\Lambda_i} \\
&\sim \int \mathrm{d}\Lambda\, e^{-\Lambda}\, \mathcal{P}(\Lambda) \\
&\sim \int \frac{\mathrm{d}\Lambda}{\sqrt{2\pi\sigma_\Lambda^2}}\, e^{-\Lambda}\; \exp\left[-\frac{(\Lambda-\mu_\Lambda)^2}{2\sigma_\Lambda^2}\right] \\
&= \int \frac{\mathrm{d}\Lambda}{\sqrt{2\pi\sigma_\Lambda^2}}\, \exp\left[-\frac{1}{2\sigma_\Lambda^2}\Lambda^2 + \frac{1}{\sigma_\Lambda^2}\left(\mu_\Lambda - \sigma_\Lambda^2\right)\Lambda - \frac{\mu_\Lambda^2}{2\sigma_\Lambda^2}\right] \\
&= \exp\left[\frac{\left(\mu_\Lambda - \sigma_\Lambda^2\right)^2}{2\sigma_\Lambda^2} - \frac{\mu_\Lambda^2}{2\sigma_\Lambda^2}\right] \int \mathrm{d}\Lambda\, \mathcal{N}(\Lambda;\mu_\Lambda,\sigma_\Lambda^2) \\
&= \exp\left[-\mu_\Lambda + \frac{\sigma_\Lambda^2}{2}\right]
\end{aligned}$$

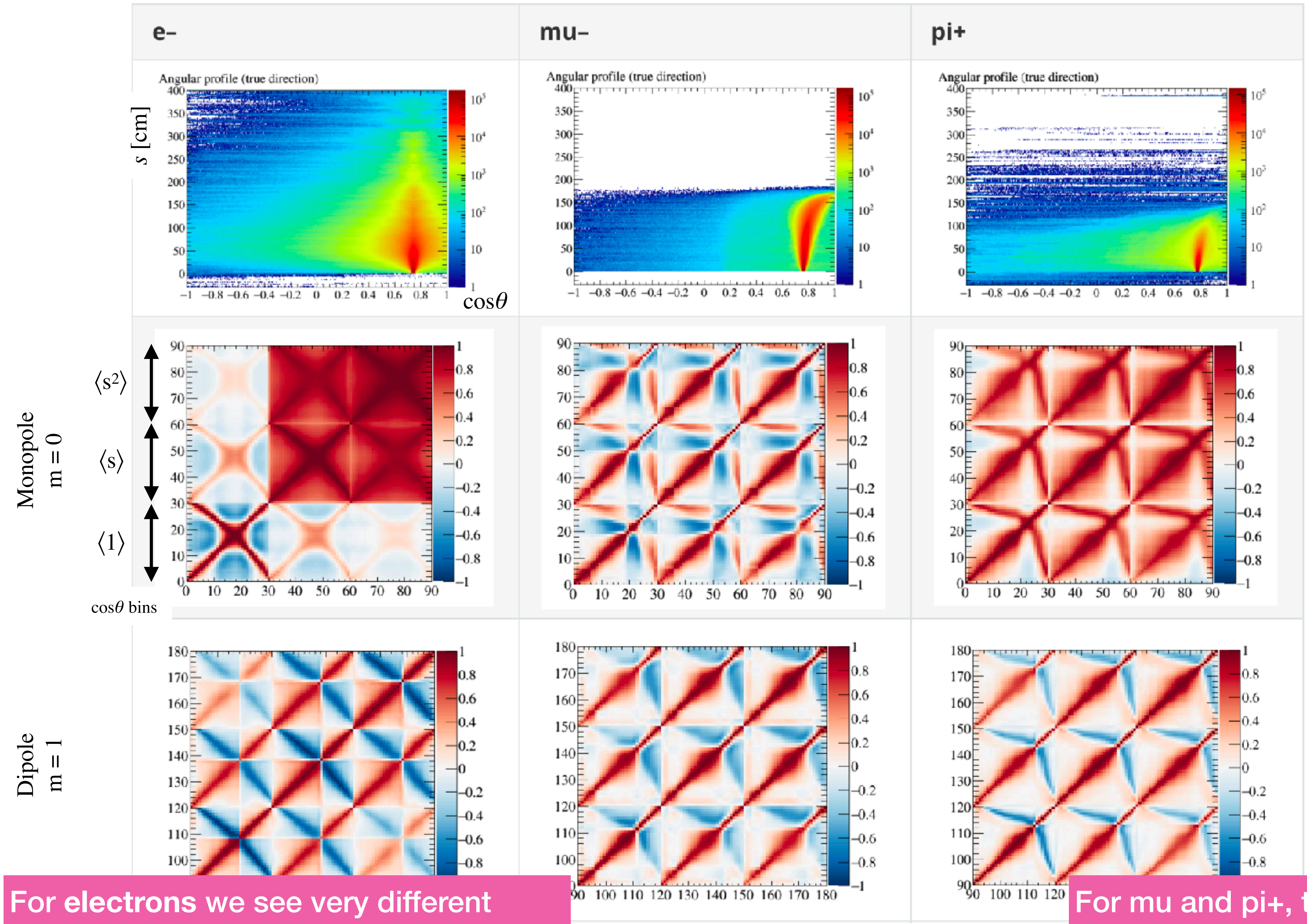
e-gamma separation?



For gamma I define s=0 at conversion point

- Integrated flux looks similar, but relative size of multipole variations look *slightly* different → could be due to somewhat low stats here, but could also give some handle on e-gamma separation

s-dependent correlations



Multiple scattering is intentionally *not* turned off for muons and pions

For **electrons** we see very different correlation of s-weighted projections → changes in track length?

For mu and pi+, the s-dependence essentially has no additional variations

Putting PMTs on a grid

Prepare square grid with size √NPMT + (0~2)

1. Sort by

additional space helps better arrangement ↑

$$\alpha\left(\frac{|z|}{H/2}-\frac{\rho}{1\,\mathrm{m}}\right)+\mathrm{rand}()$$

where $\alpha = 10$ and H is tank height.

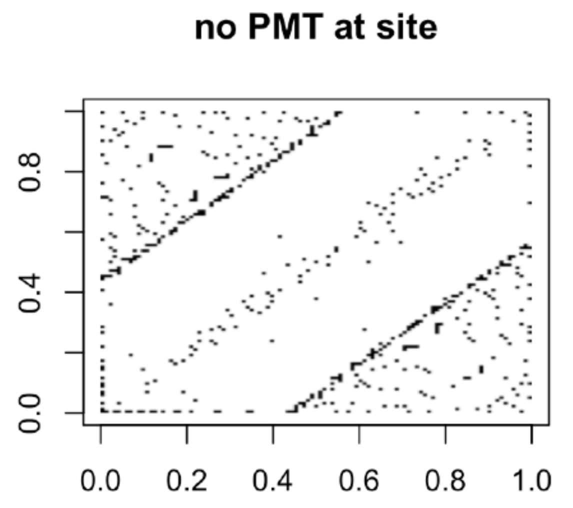
2. Assign to closest empty grid site with distance given by formula:

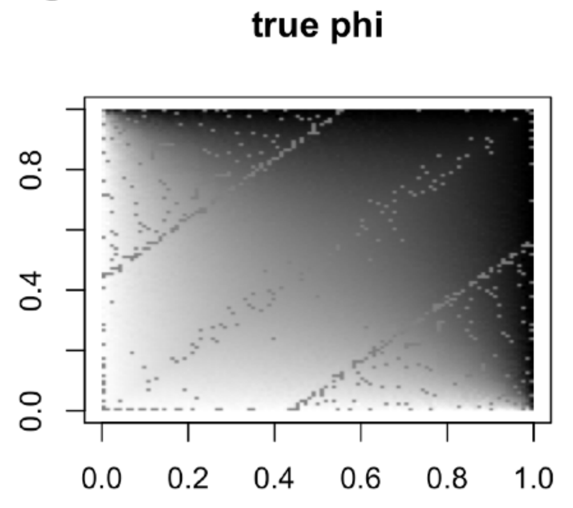
$$(\Delta W)^2+\beta(\Delta P)^2$$

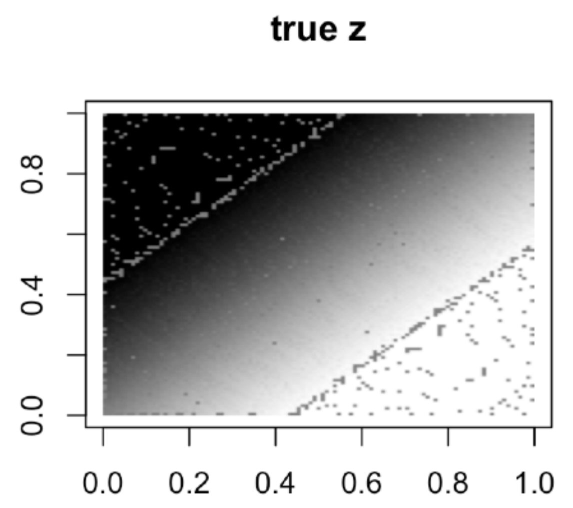
where $\beta = 3$ and

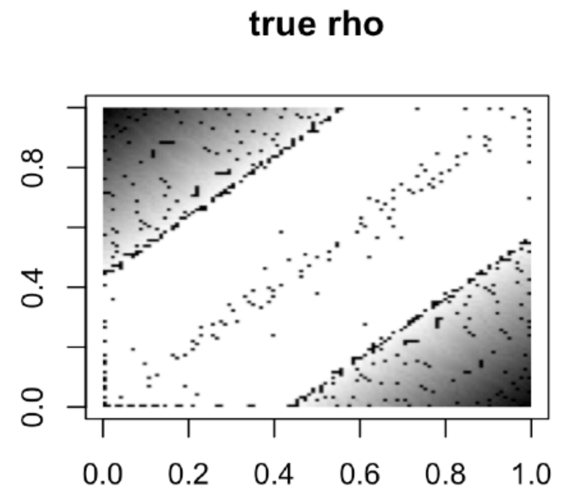
$$W := X_+ + X_-$$
$$P := X_+ - X_-$$

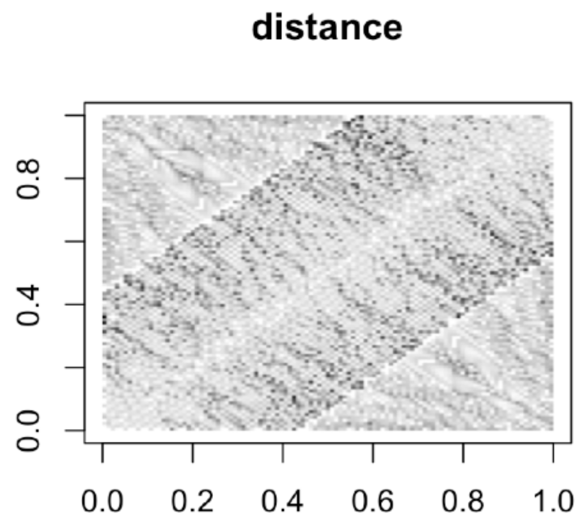
Very likely there are better algorithms. Also α,β can be optimized if necessary.

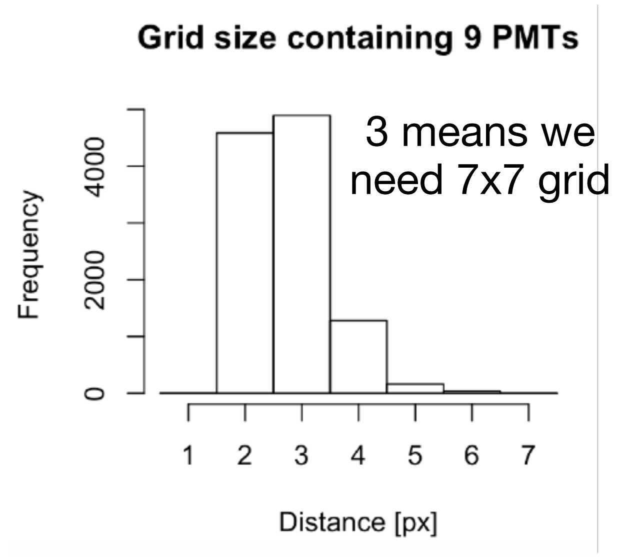




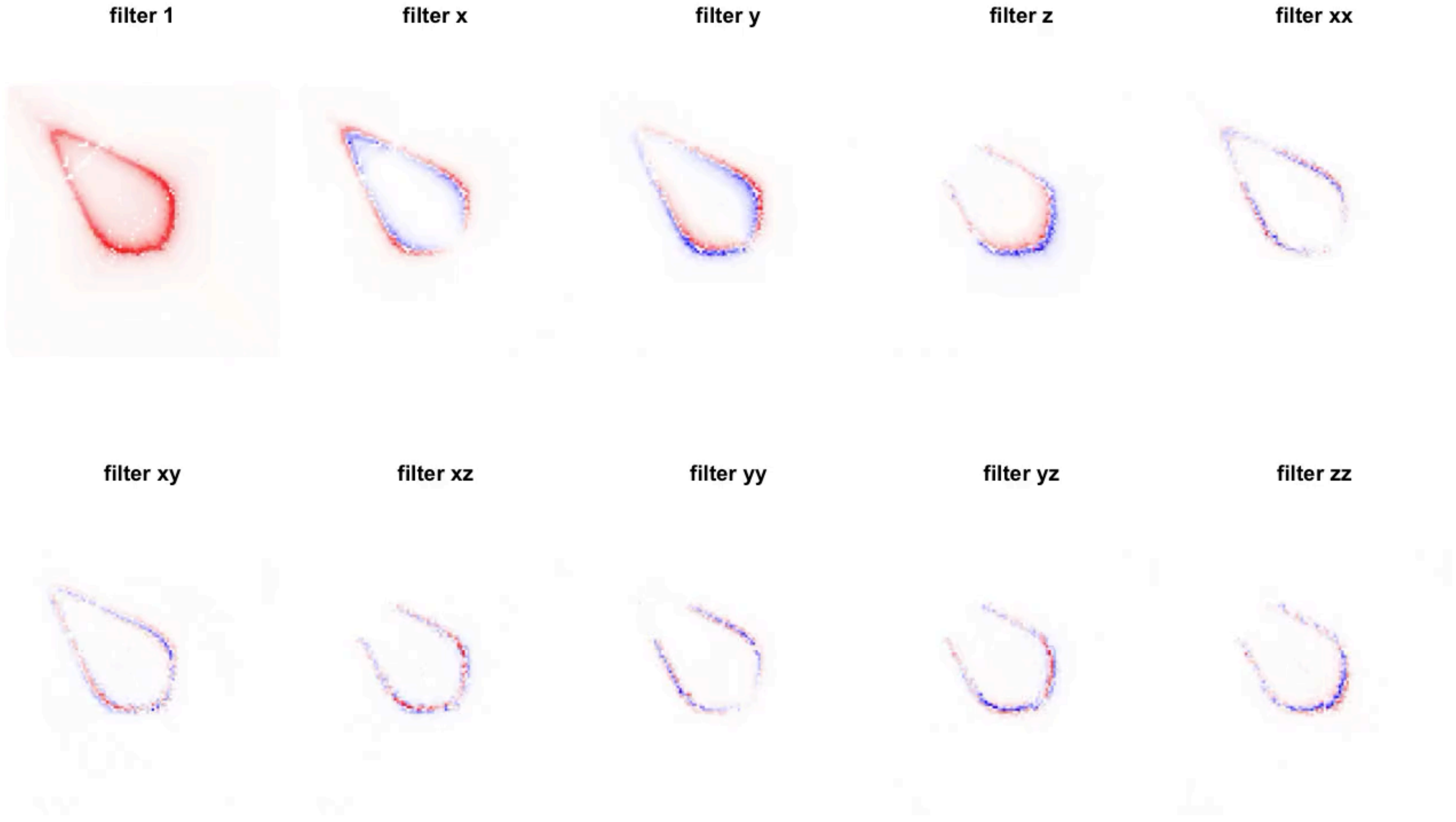








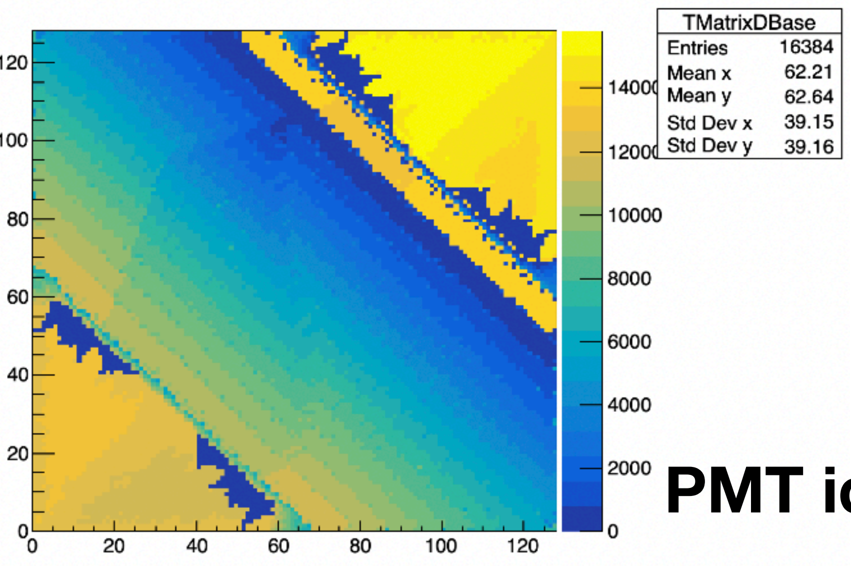
ϕ-motion



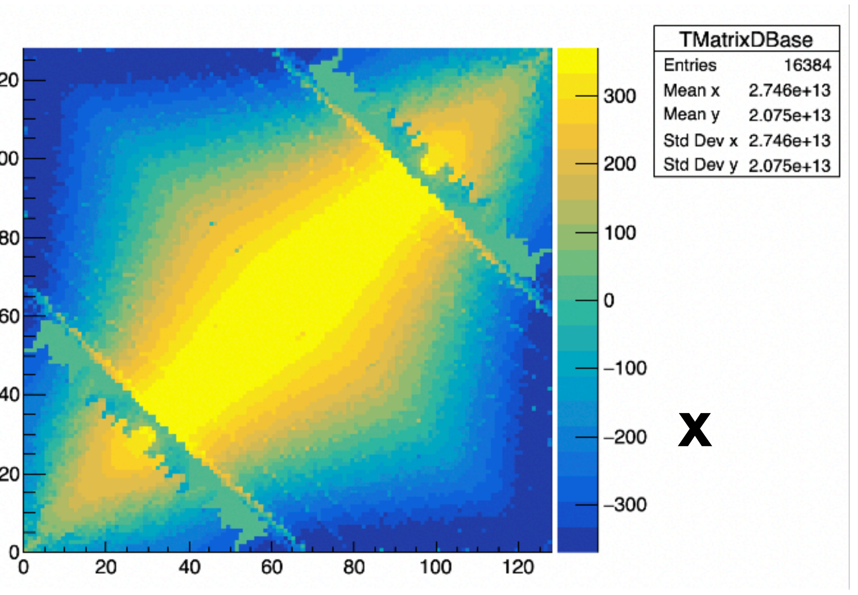
filter 1
filter x
filter y
filter z
filter xx
filter xy
filter xz
filter yy
filter yz
filter zz

IWCD mPMTs mapped onto grid

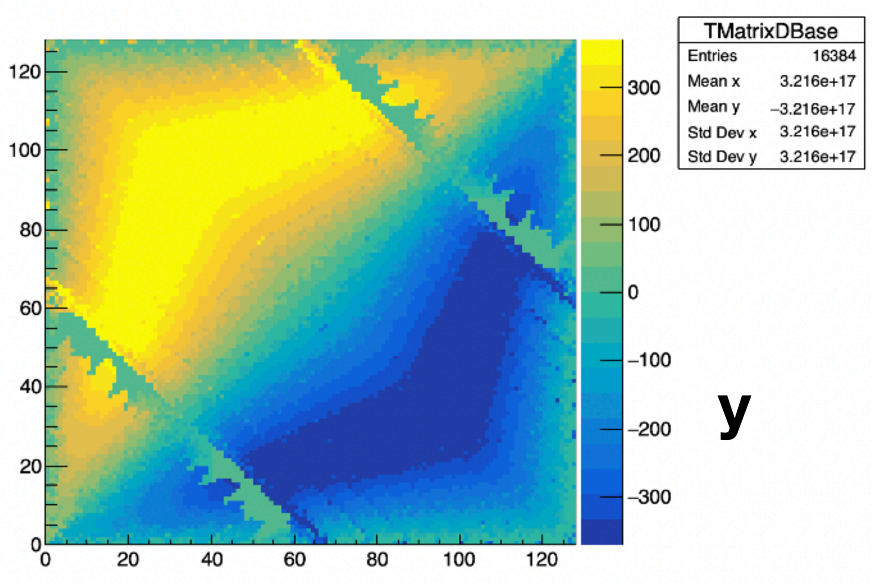
Same algorithm as before, see backup slide



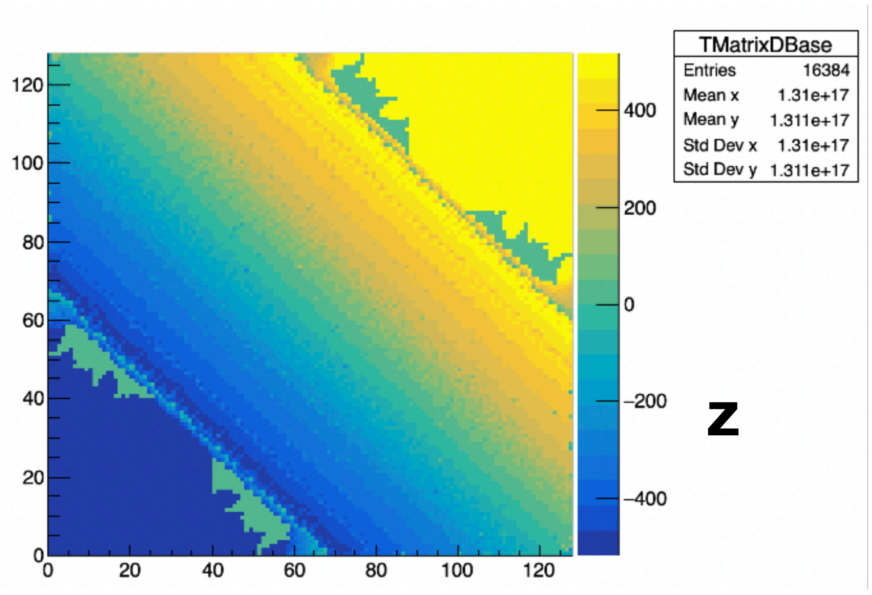
PMT id



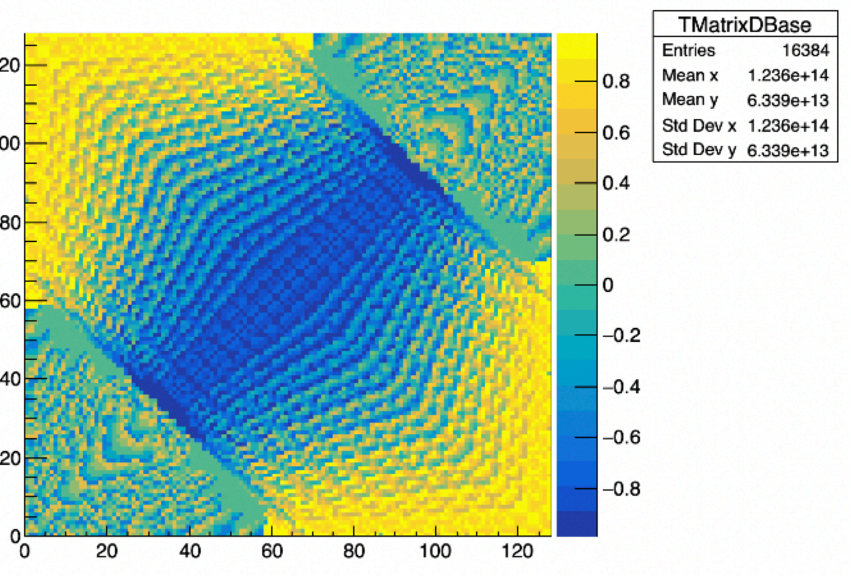
x



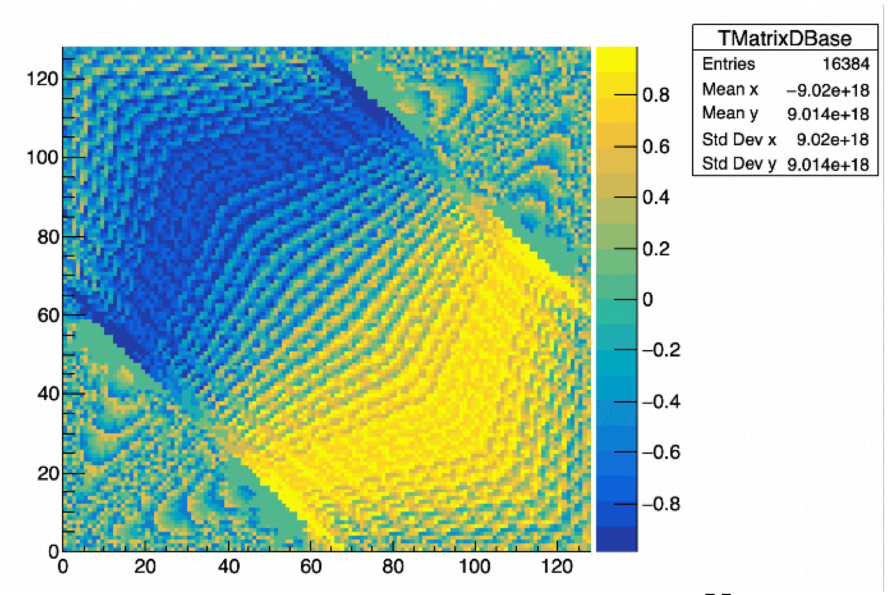
y



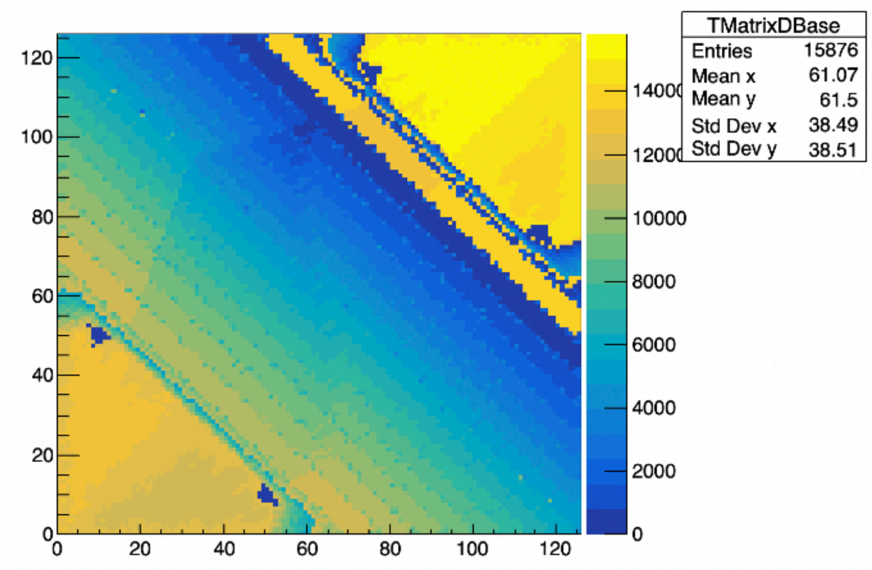
z



dir_x

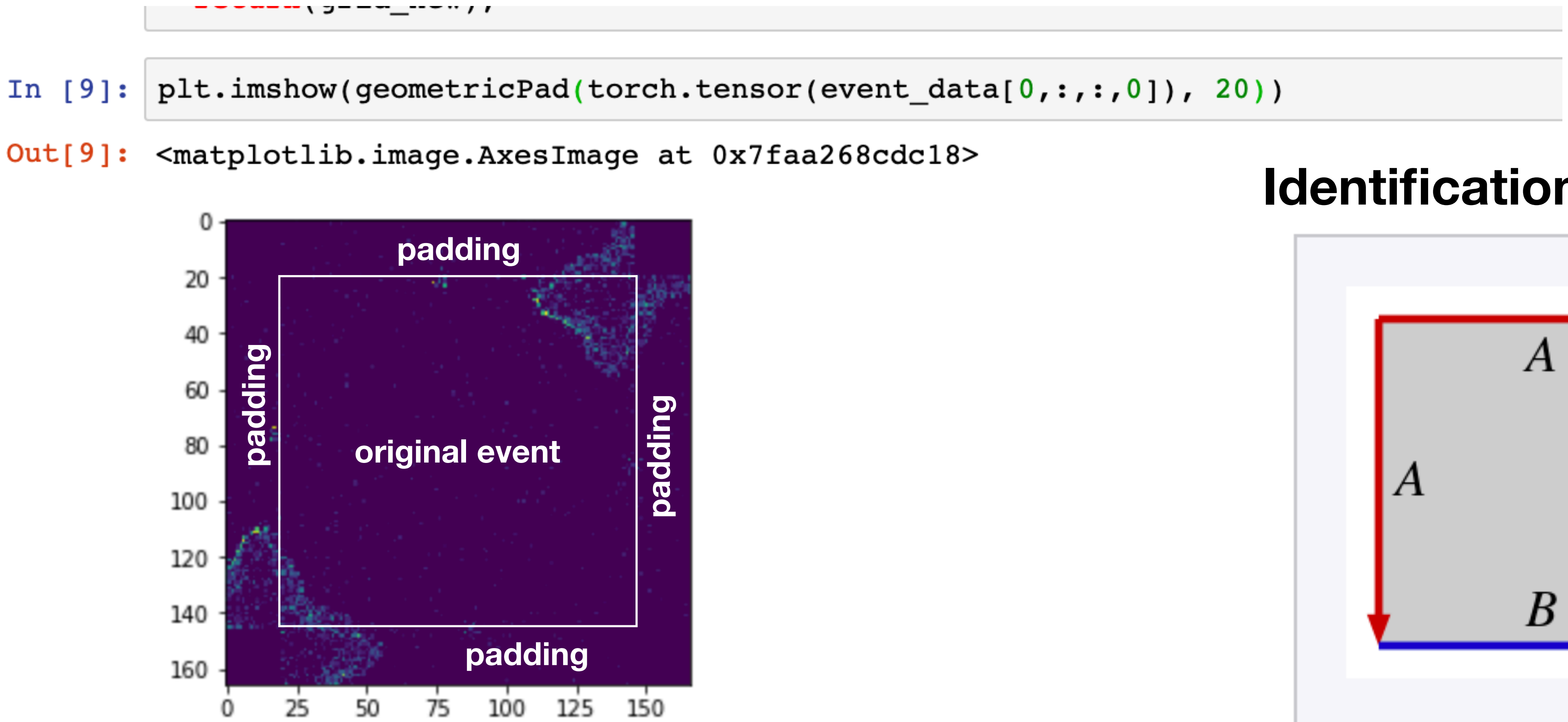


dir_y



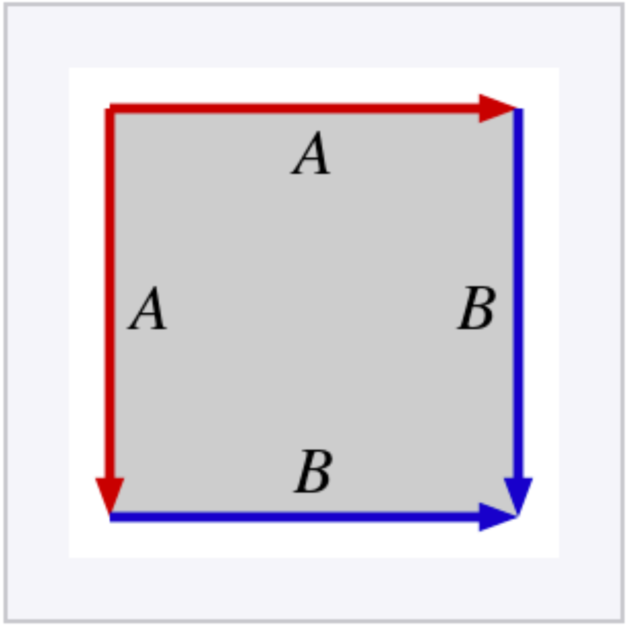
dir_z

Event from WCsim mapped to grid



```
In [9]: plt.imshow(geometricPad(torch.tensor(event_data[0,:,:,0]), 20))
Out[9]: <matplotlib.image.AxesImage at 0x7faa268cdc18>
```

Identification of sides:



- Saved in format (HDF5) that can be read by pyTorch efficiently
- Padding that reflects geometry is applied inside pyTorch using tensor operations

Effective rotation

1. when requesting an event (get entry), randomly sample $\phi_0 \in [0, 2\pi]$.
2. when taylorizing (the w-sum) insert a rotation matrix:

$$w_I = (R^T w)_I = R^K{}_I w_K =: R_I{}^K w_K$$
$$w_{IJ} = \left(R^T w R\right)_{IJ} = R_I{}^K R_J{}^L w_{KL}$$

   in the case of ϕ_0 this is

$$R^I{}_J(\phi_0) = \begin{pmatrix} \cos\phi_0 & -\sin\phi_0 & 0 \\ \sin\phi_0 & \cos\phi_0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

   so the whole convolution operation becomes:

$$N_i^{(n+1)} = \sum_j \left(K w_{ij} + K^I R^K{}_I w_{K,ij} + \frac{1}{2} K^{IJ} R^K{}_I R^L{}_J w_{KL,ij} \right) N_j^{(n)}$$

3. when comparing the output, we need to also rotate any vectors like the direction or position:

$$x_I = (Rx)_I = R_{IJ} x_J$$

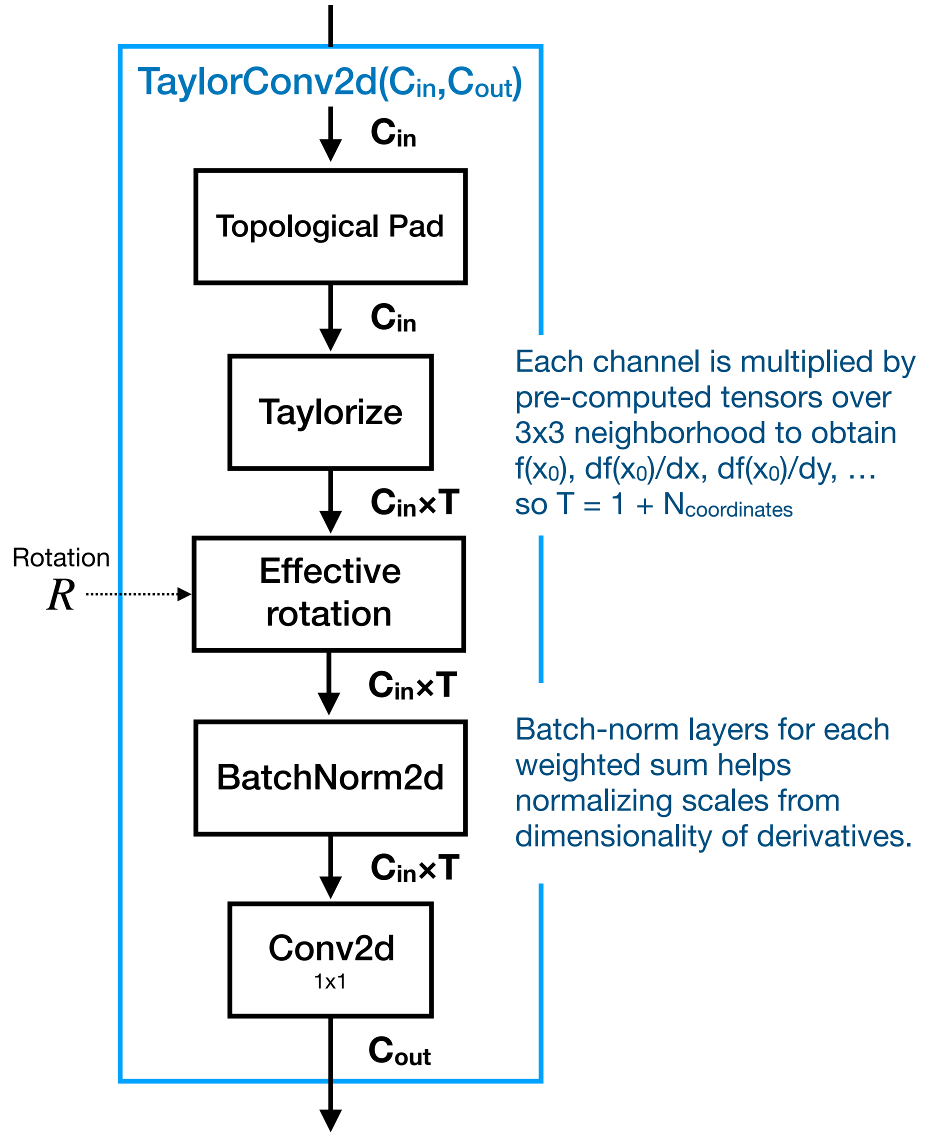
   notice that w and x rotate in opposite direction. That's because w is related to derivatives and is therefore covariant, whereas x is contravariant.

Maybe for efficiency reasons it's actually better to use the same ϕ_0 for one batch.

should prevent overtraining when training set is small

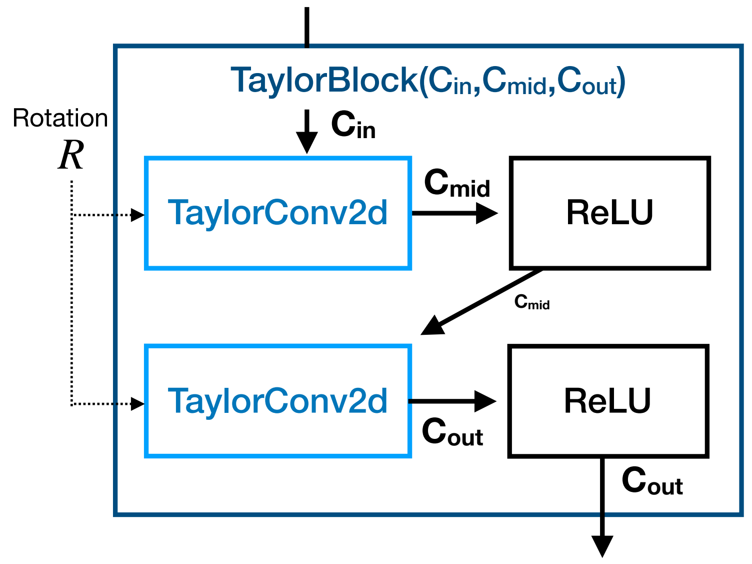
(of course concept itself is not new, but this way it's very simple and has no issues with grid-layout at endcaps)

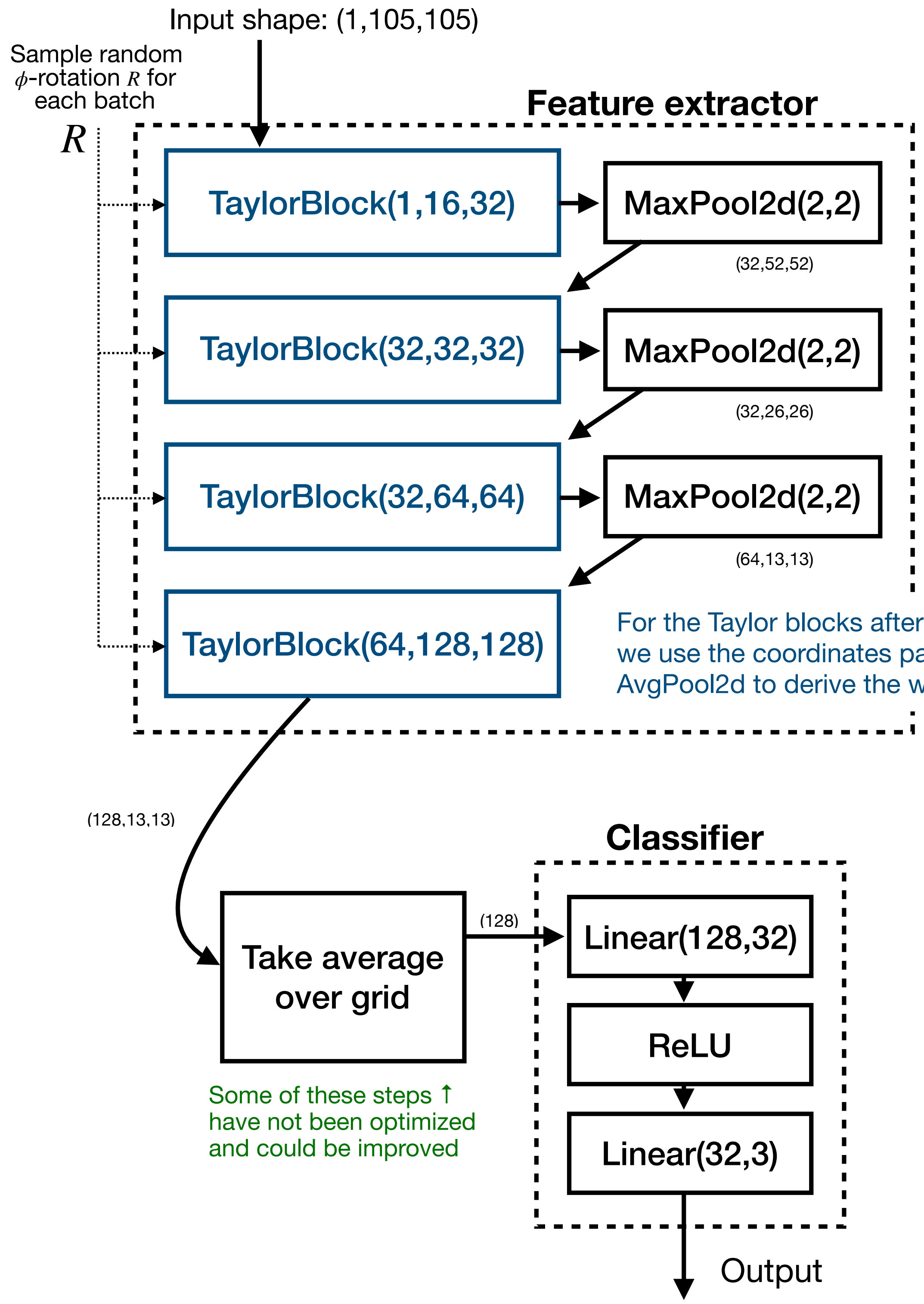
5 Network architecture



Each channel is multiplied by pre-computed tensors over 3x3 neighborhood to obtain f(x₀), df(x₀)/dx, df(x₀)/dy, … so $T = 1 + N_{coordinates}$

Batch-norm layers for each weighted sum helps normalizing scales from dimensionality of derivatives.





For the Taylor blocks after the max pooling we use the coordinates passed through AvgPool2d to derive the weights.

Some of these steps ↑ have not been optimized and could be improved

Dataset

- SK 40% coverage in WCSim
- Two wall-material sets: {blacksheet, retro-reflector}

  ↑ contact me if interested
- 50k events for {e^-,μ^-,π^+} each
- Uniform vertex inside ID
- Isotropic direction (no FC cut yet)
- Flat energy from 1 MeV to 1 GeV

```
/mygen/generator gps
/gps/particle ###
/gps/pos/type  Volume
/gps/pos/shape Cylinder
/gps/pos/centre 0 0 0
/gps/pos/radius 17. m
/gps/pos/halfz  18. m
/gps/ang/type iso
/gps/ene/type Pow
/gps/ene/min   0.001 GeV
/gps/ene/max   1.    GeV
/gps/ene/alpha 0.
```

Optimization target is classification of three PIDs (cross-entropy)

Performance study

Dataset

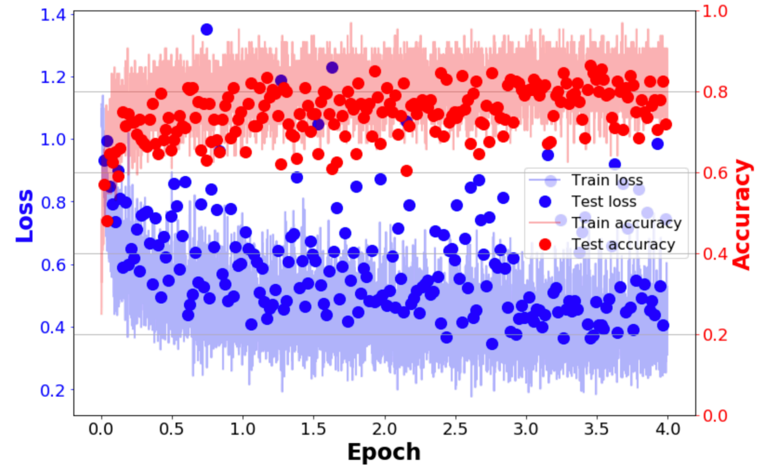
- Super-K simulated in WCSim [2,3]
- 50k events for {e⁻, μ⁻, π⁺} each
- Uniform vertex distribution
  (inside inner detector)
- Isotropic direction
  (particles might leave the detector = no FC cut)
- Flat energy from 1 MeV to 1 GeV
- Only using charge information for now

Optimization target:
classification of three particle types (PIDs)
(cross-entropy)

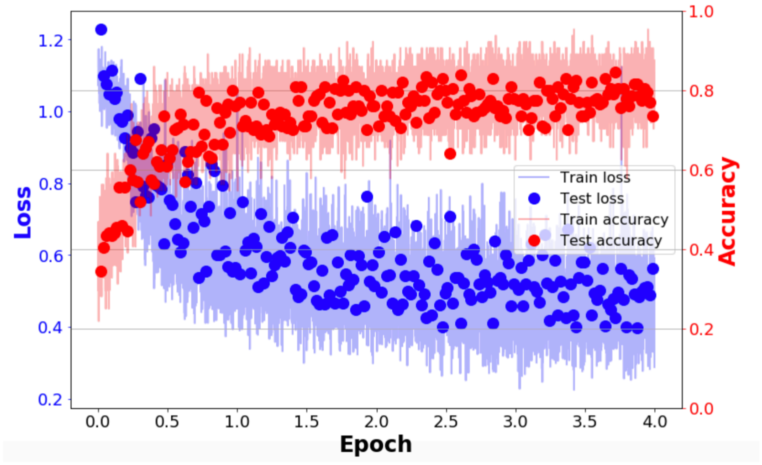
Implemented in PyTorch on GPU. The Taylorizing and random rotations introduce ~10x computing overhead.

Effect of Taylorizing itself

Ordinary convolution on topological map



TaylorConv



No random rotations performed at this stage.

TaylorConv might show slightly better generalization, but at this stage no significant difference compared to ordinary convolution (interesting). Should also compare with more naïve approaches that don't use the topological map.

Since particle ID depends on local features (blurriness) it would be interesting to see performance differences for full reconstruction of track parameters (vertex, energy, …).

Other aspects

$$N_i^{(n+1)} = \sum_j \left(K w_{ij} + K^I w_{I,ij} + \frac{1}{2} K^{IJ} w_{IJ,ij} \right) N_j^{(n)}$$

- Since K, K_I, K_{IJ} are geometric objects, rotating them effectively rotates the original event $\rightarrow$ can randomly rotate during training to impose ϕ-symmetry of tank

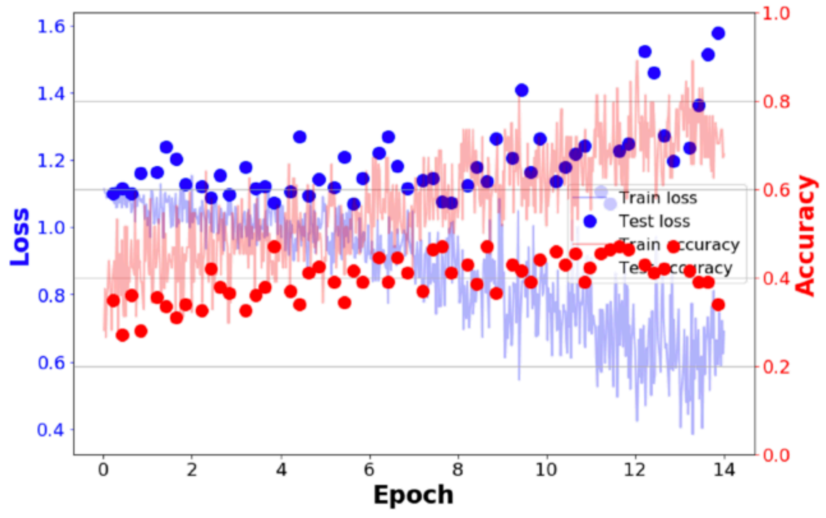
- Dead PMTs could be removed by simply recomputing weights w without much retraining, since training is in 3D-space, not in pixel-space.

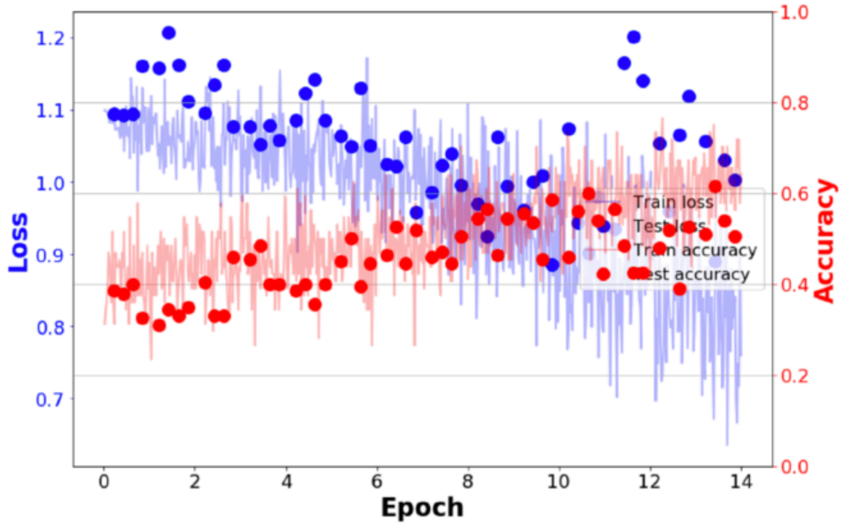
- PMT orientation could be passed as another "spatial" dimension. (for mPMT etc.)

PID training with just 1000 events

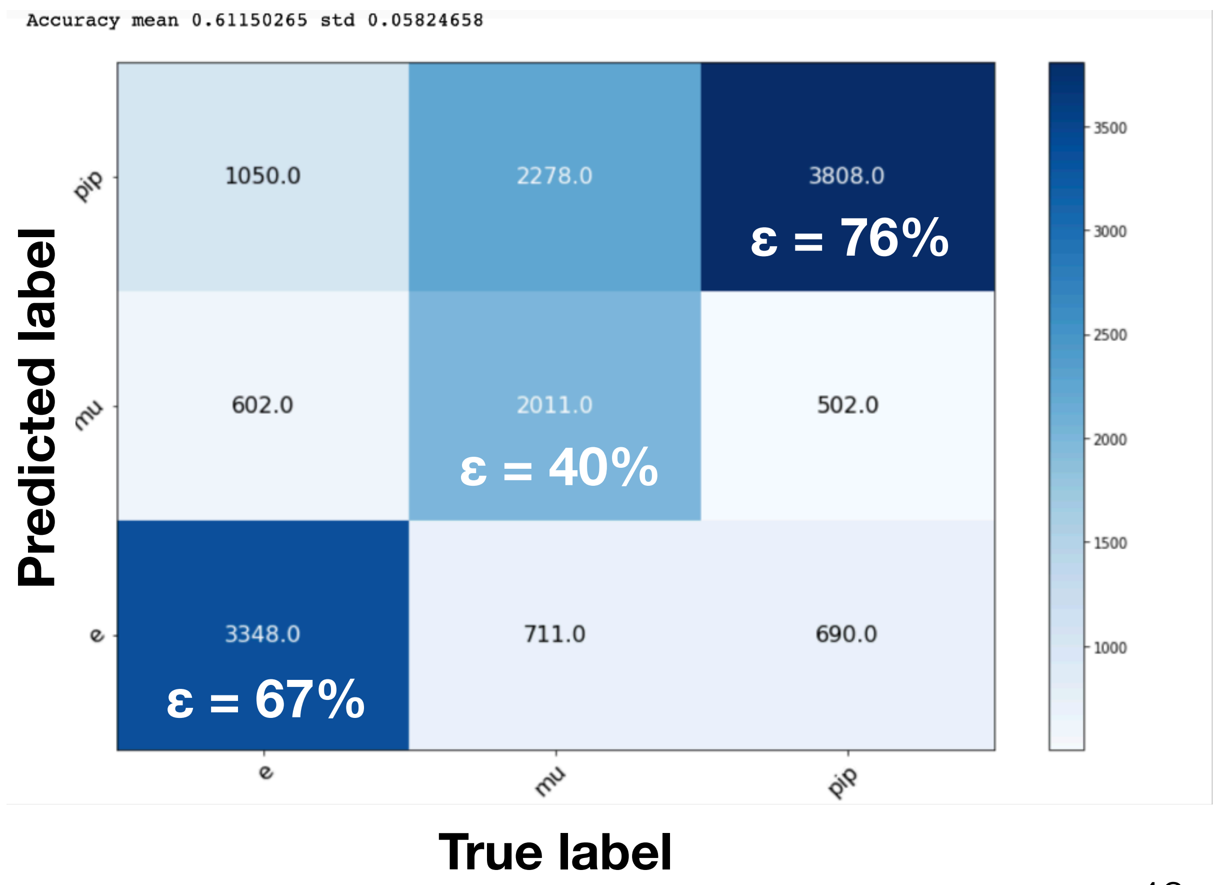
- e,μ,π+ classification in dataset with uniform, isotropic, flat energy $E \in [1\ \mathrm{MeV},\ 1\ \mathrm{GeV}]$
- Training set: 1000 events, test set: 5000 events

Without random rotations, quickly starts overtraining? →





With random rotations, learning at least for 10 epochs? ←



```
Accuracy mean 0.61150265 std 0.05824658
```

Predicted label \ True label	e	mu	pip
pip	1050.0	2278.0	3808.0 ε = 76%
mu	602.0	2011.0 ε = 40%	502.0
e	3348.0 ε = 67%	711.0	690.0

- Does surprisingly well for having seen only 1000 events
- If O(1000 events) can already give interesting results, maybe training feature extraction **with data** is possible?

Hybrid training?

$$\begin{tikzcd}\textbf{Feature extraction (CNN)} \arrow[d] \\ \textbf{Classification (NN)}\end{tikzcd}$$

- Normally: train everything at once **using MC**
- Feature extraction could be trained **using data** (auto-encoder) utilizing ϕ-rotation presented today.
- Take this **data-trained** feature extractor and train classifier **using MC** (where we have truth)
- Can prevent picking up subtle implementation issues of the MC in feature extraction?

Loss for energy reconstruction

- Using mean-squared error on $F := \sqrt{E}$. Thus assuming typical resolution of energy is proportional to $\sigma(E) \propto \sqrt{E}$, we get: $\sigma(F) = \frac{\mathrm{d}F}{\mathrm{d}E}\sigma(E) = \frac{\sigma(E)}{\sqrt{E}} = \mathrm{const}$

- I have batch-norm layers throughout the network. Maybe this causes issues with energy reconstruction which is mostly related to the total charge, I pass the raw total charge as an additional input to the fully-connected layer.

- Might need to look at using Huber loss. In this case I think I need to normalize the scale for the linear-quadratic transition, maybe can assume typical error scale of 2% around 500 MeV

Combining loss functions

- For CNN-based reconstruction, was stuck at how to combine loss functions (e.g. PID classification and momentum reconstruction).

- Depending on relative weights only one or the other gets optimized, hard to get both optimized. For example I get for cross entropy values like 0.5, whereas mean-squared error loss of energy gives me values like 1000 (obviously this depends on the units!)

- Found simple method [1] by *learning* this weight ($1/V_i$), regularized by logarithm.

$$L = \sum_i \frac{1}{V_i} L_i + \log V_i$$

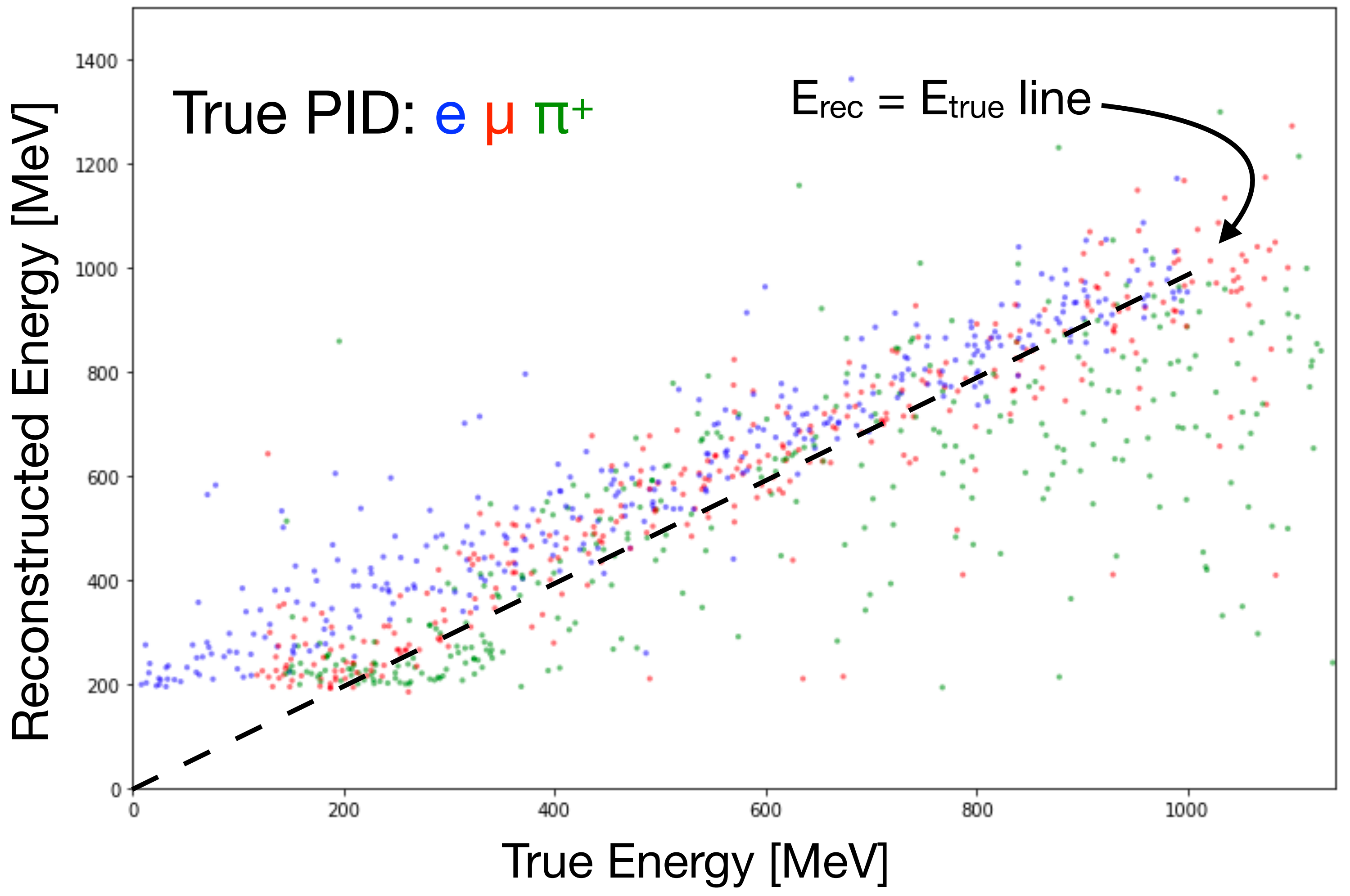
the idea is that by minimizing V_i, they become equal in magnitude to the individual losses L_i

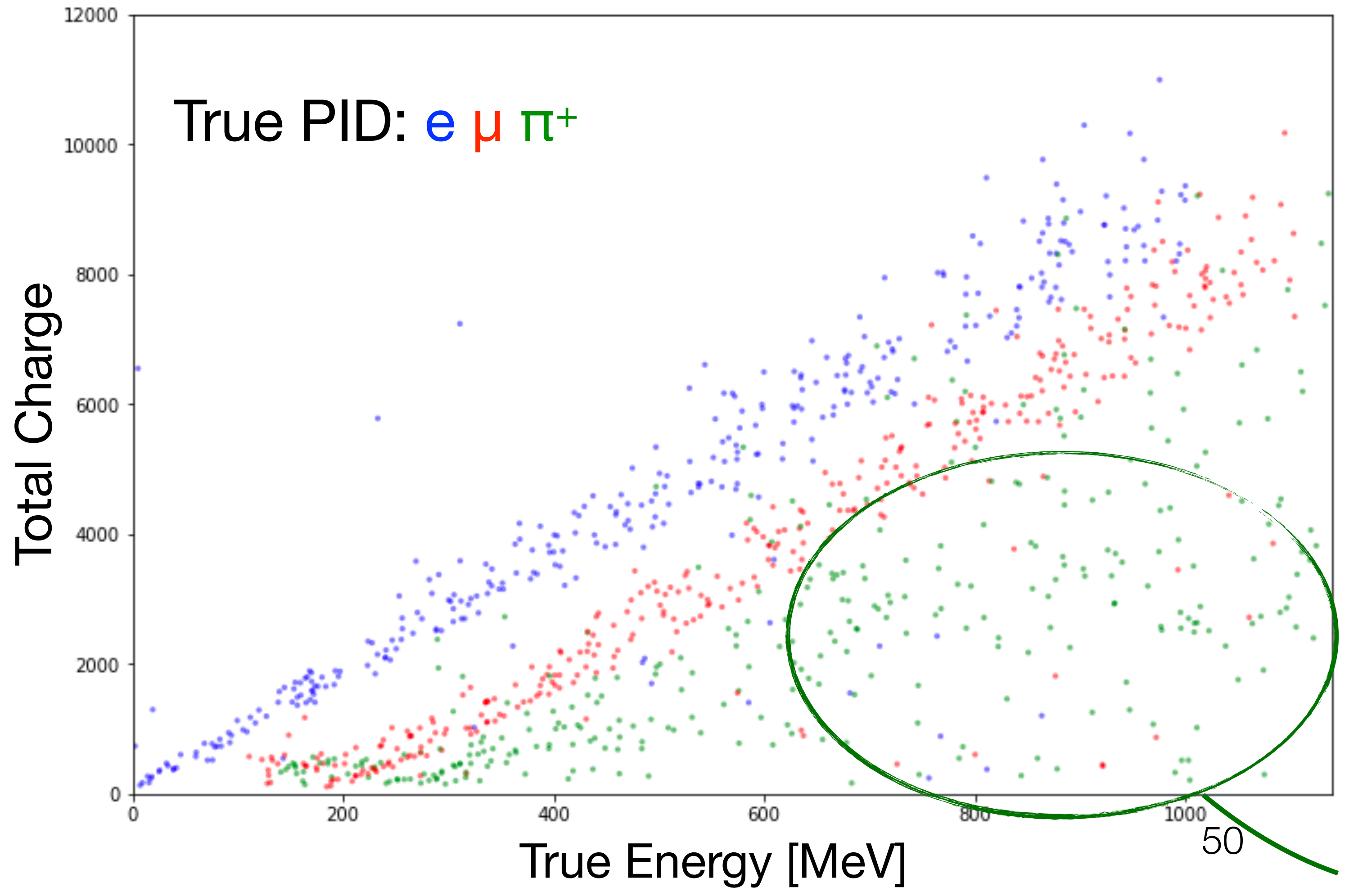
$$0 = \frac{\partial \langle L \rangle}{\partial V_i} = -\frac{1}{V_i^2}\left(\langle L_i \rangle - V_i\right) \quad \Rightarrow \quad V_i = \langle L_i \rangle$$

such that each individual loss is normalized by it's average magnitude:

$$L = \sum_i \frac{L_i}{\langle L_i \rangle} + \log \langle L_i \rangle$$

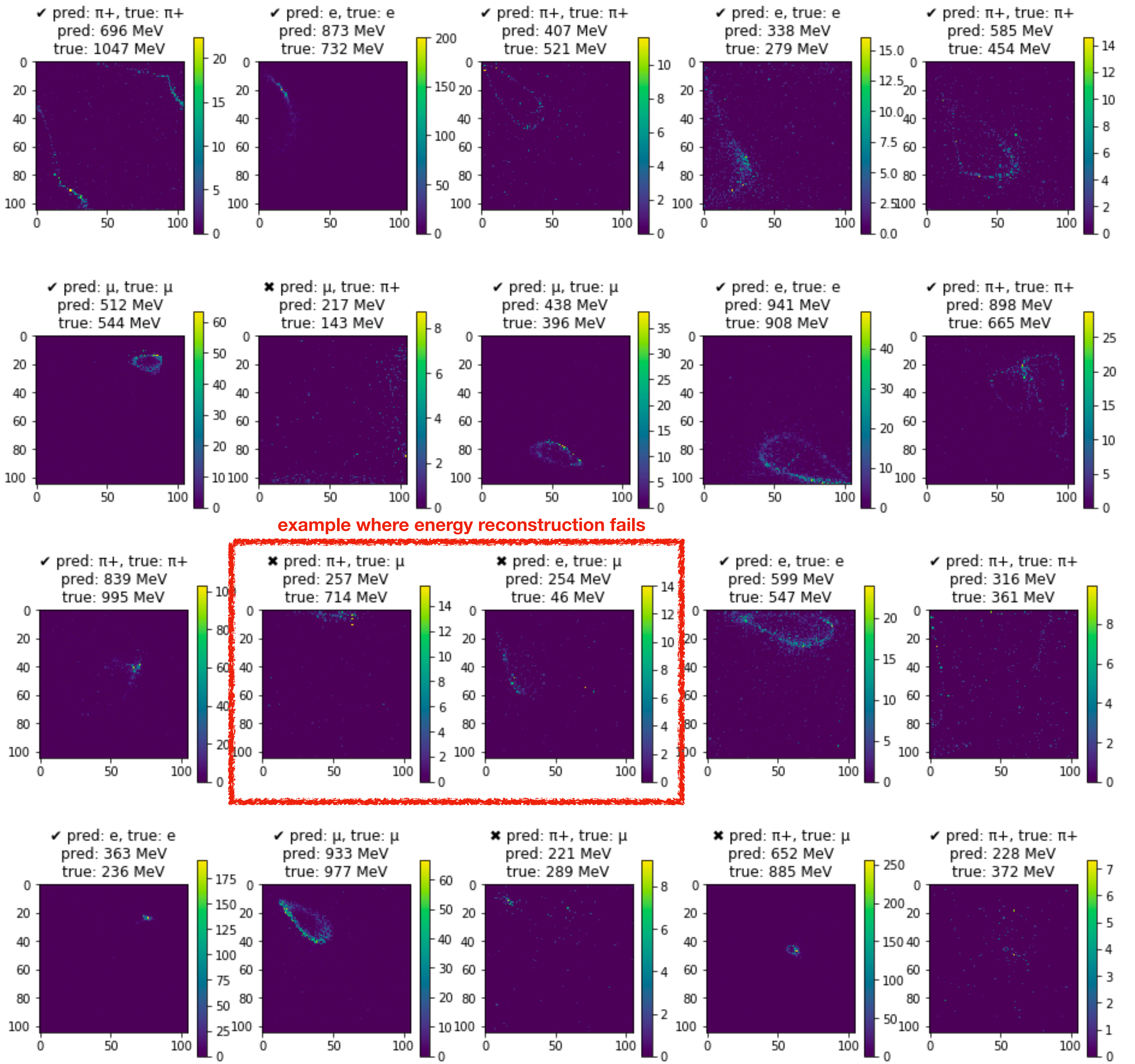
1. A. Kendall et al., “Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics” (2018)



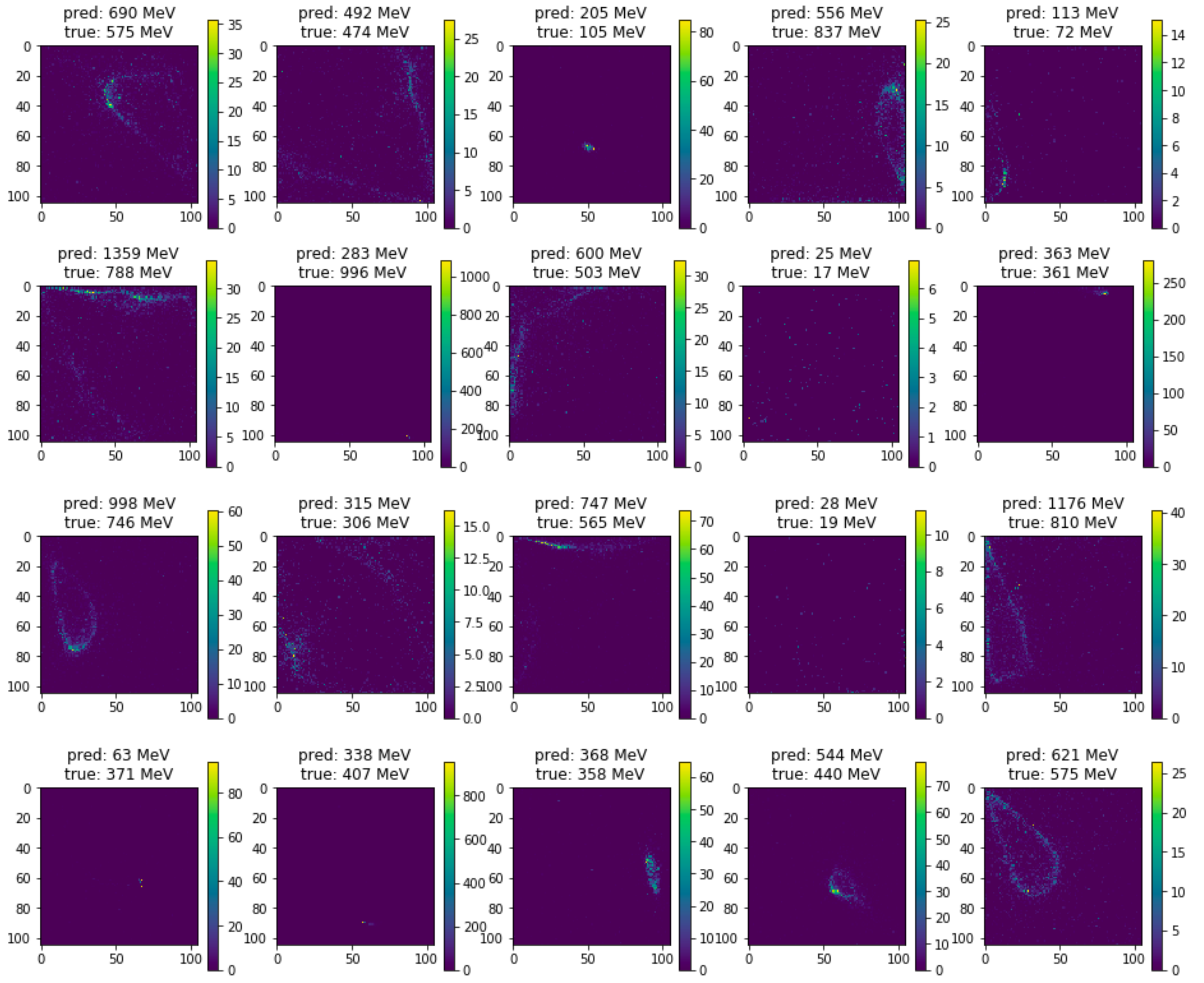


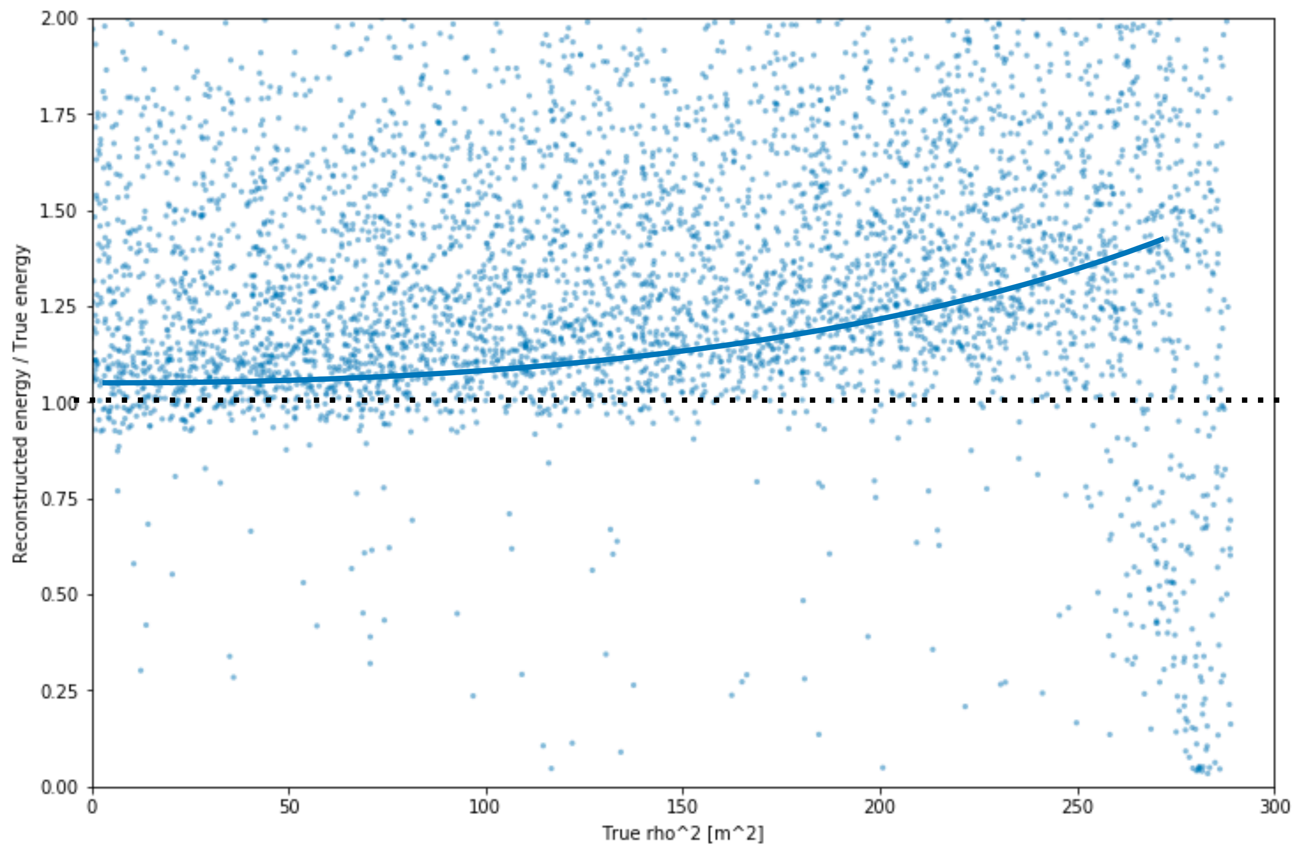
- E_{rec} close to E_{true}, and it seems to take PID into account.
- For **electrons** resolution is worse than total charge?
- For **pions** network does better job than just looking at total charge?
- Probably better to choose different loss-scaling factor based on PID.
- For whatever reason the reconstructed energy never goes below 200 MeV. Maybe easier to reconstruct kinetic energy.

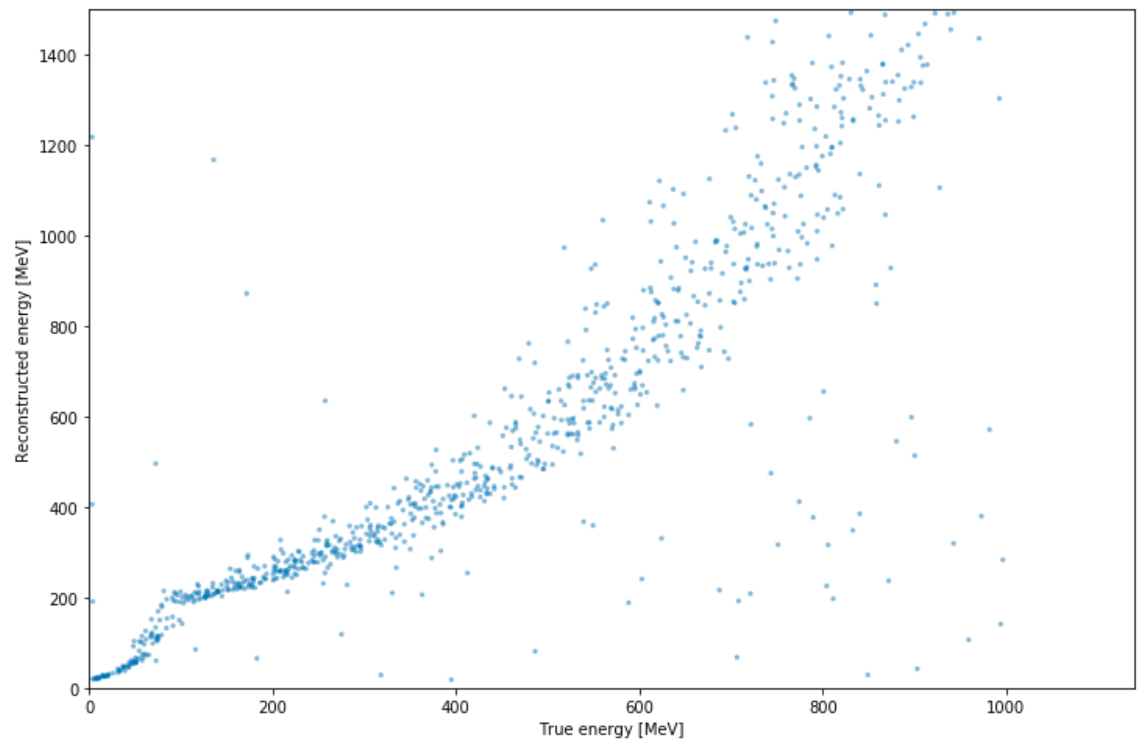
PID + Erec

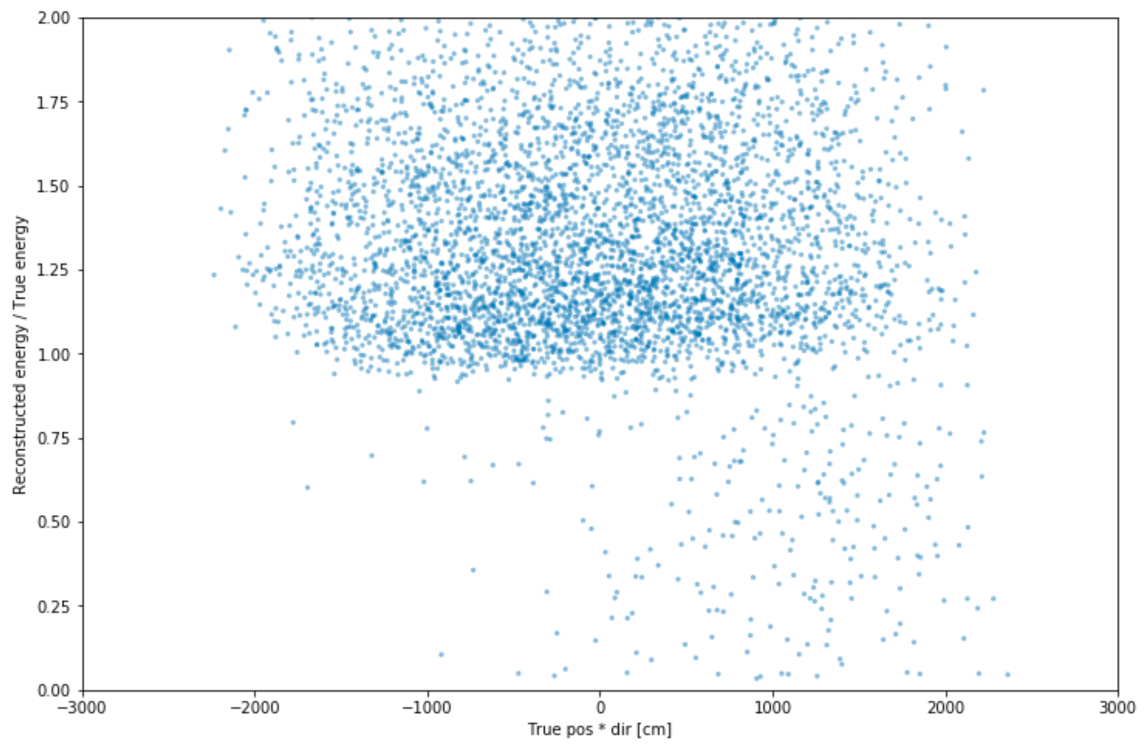

✔ pred: π+, true: π+ pred: 696 MeV true: 1047 MeV
✔ pred: e, true: e pred: 873 MeV true: 732 MeV
✔ pred: π+, true: π+ pred: 407 MeV true: 521 MeV
✔ pred: e, true: e pred: 338 MeV true: 279 MeV
✔ pred: π+, true: π+ pred: 585 MeV true: 454 MeV
✔ pred: μ, true: μ pred: 512 MeV true: 544 MeV
✖ pred: μ, true: π+ pred: 217 MeV true: 143 MeV
✔ pred: μ, true: μ pred: 438 MeV true: 396 MeV
✔ pred: e, true: e pred: 941 MeV true: 908 MeV
✔ pred: π+, true: π+ pred: 898 MeV true: 665 MeV
example where energy reconstruction fails
✔ pred: π+, true: π+ pred: 839 MeV true: 995 MeV
✖ pred: π+, true: μ pred: 257 MeV true: 714 MeV
✖ pred: e, true: μ pred: 254 MeV true: 46 MeV
✔ pred: e, true: e pred: 599 MeV true: 547 MeV
✔ pred: π+, true: π+ pred: 316 MeV true: 361 MeV
✔ pred: e, true: e pred: 363 MeV true: 236 MeV
✔ pred: μ, true: μ pred: 933 MeV true: 977 MeV
✖ pred: π+, true: μ pred: 221 MeV true: 289 MeV
✖ pred: π+, true: μ pred: 652 MeV true: 885 MeV
✔ pred: π+, true: π+ pred: 228 MeV true: 372 MeV

Erec only with variance function







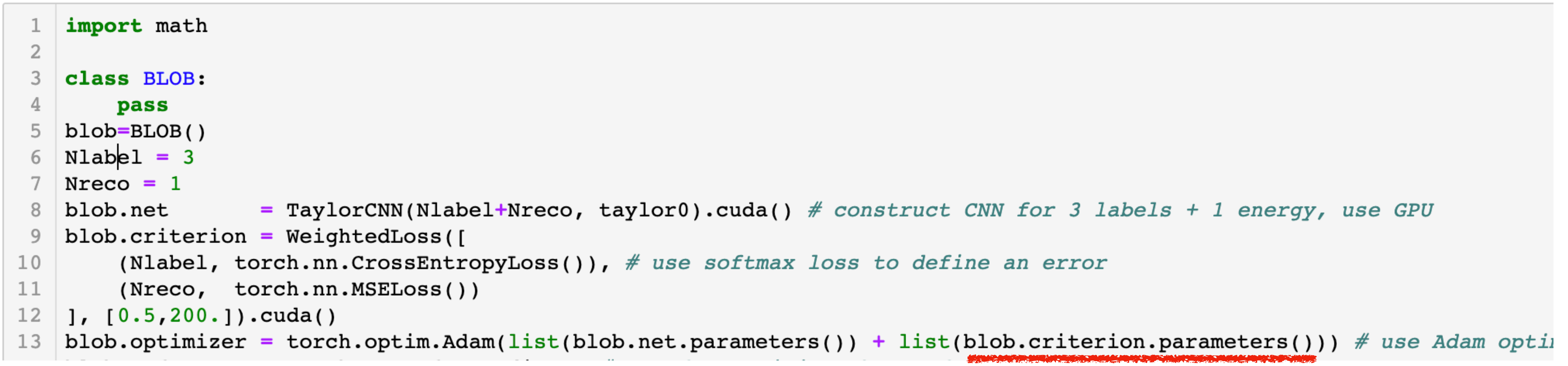


- Overall resolution is worse, but that might be because I changed from reconstructing $\sqrt{E}$ to $\log E$ (expecting the network would learn this by itself). Also should train longer to get rid of kink.

- bias is still present?

- underprediction when moving toward wall →

```python
class WeightedLoss(torch.nn.Module):
    # losses should be a list of 2-tuples (Npred,Loss)
    # where Npred is the number of prediction params to use for this loss,
    # and Loss is the loss function to evaluate.
    # The loss functions are applied to the input value in the given order.
    #
    # the initScale can be a list of typical scales for the losses, which is used to initialize logVar=log(initSca
    #
    # from A. Kendall et al., "Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semant
    # the idea is to add weight parameters to the loss function as
    # totalLoss = sum_i [L_i/V_i + log V_i]
    # such that
    # 0 = d(totalLoss)/dV_i
    #   = -1/V_i^2 [L_i - V_i]
    # so by optimizing V_i, it becomes equal to the typical scale of L_i
    def __init__(self,losses,initScale=None):
        super(WeightedLoss, self).__init__()

        self.predPartitions = []
        self.losses = []
        predCumsum = 0
        for Npred,loss in losses:
            self.predPartitions.append((predCumsum,Npred)) # start and length
            predCumsum = predCumsum + Npred
            self.losses.append(loss)

        Nlosses = len(losses)
        if initScale is None:
            initScale = np.ones(Nlosses)
        self.logVar = torch.nn.Parameter(torch.log(torch.tensor(initScale, dtype=torch.float)))
        self.logVar.requires_grad = True

    def forward(self, pred, true):
        # pred is a 1d tensor with length sum(Npred)
        # true is a normal python list of the true values to pass to each loss function (probably each a tensor)
        oneOverVar = torch.exp(-self.logVar)
        totalLoss = self.logVar.sum()
        for i,loss in enumerate(self.losses):
            thisPred = pred.narrow(1,self.predPartitions[i][0],self.predPartitions[i][1])
            thisTrue = true[i]
            totalLoss = totalLoss + oneOverVar[i]*loss(thisPred,thisTrue)

        return totalLoss

    # the default implementation of cpu() and cuda() only passes this along to nn.Module instances
    # since self.logSigma is a tensor, we need to take care of it ourselves

    def cpu(self):
        super(WeightedLoss, self).cpu()
        self.logVar.cpu()
        return self

    def cuda(self):
        super(WeightedLoss, self).cuda()
        self.logVar.cuda()
        return self
```



```python
import math

class BLOB:
    pass
blob=BLOB()
Nlabel = 3
Nreco = 1
blob.net       = TaylorCNN(Nlabel+Nreco, taylor0).cuda() # construct CNN for 3 labels + 1 energy, use GPU
blob.criterion = WeightedLoss([
    (Nlabel, torch.nn.CrossEntropyLoss()), # use softmax loss to define an error
    (Nreco,  torch.nn.MSELoss())
], [0.5,200.]).cuda()
blob.optimizer = torch.optim.Adam(list(blob.net.parameters()) + list(blob.criterion.parameters())) # use Adam opti
```

- Might also be interesting to use a separate optimizer for the loss weights, since we don't want these weights to change as rapidly as the network weights