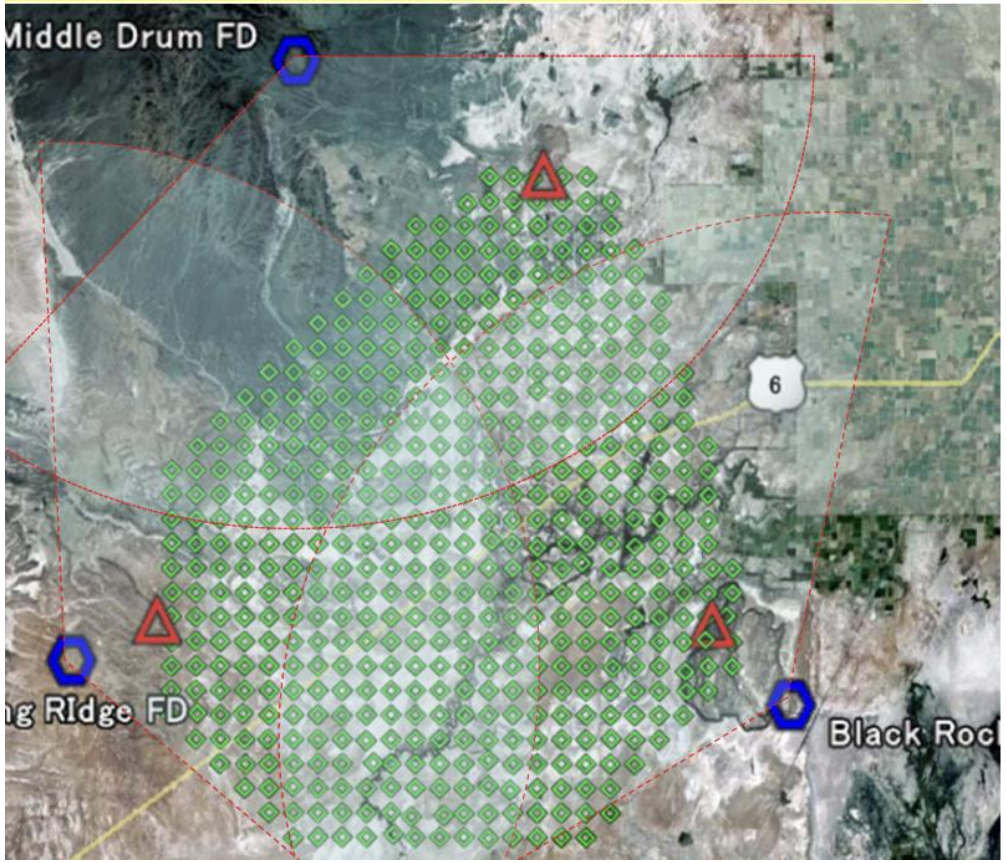
TA実験空気シャワー解析における深層学習を用いた空気シャワー再構成手法の開発

空気シャワー研究会
大阪電気通信大学
中澤　奏駿

テレスコープアレイ実験 (TA実験)

- 超高エネルギー宇宙線における質量組成および起源解明を目的とした実験
- アメリカ合衆国ユタ州の砂漠地帯
  観測面積は約700 km^2
- 観測装置は主に大気蛍光望遠鏡(FD)
  1.2 km 間隔で 507 台設置されている
  地表粒子検出器(SD)の 2 種類

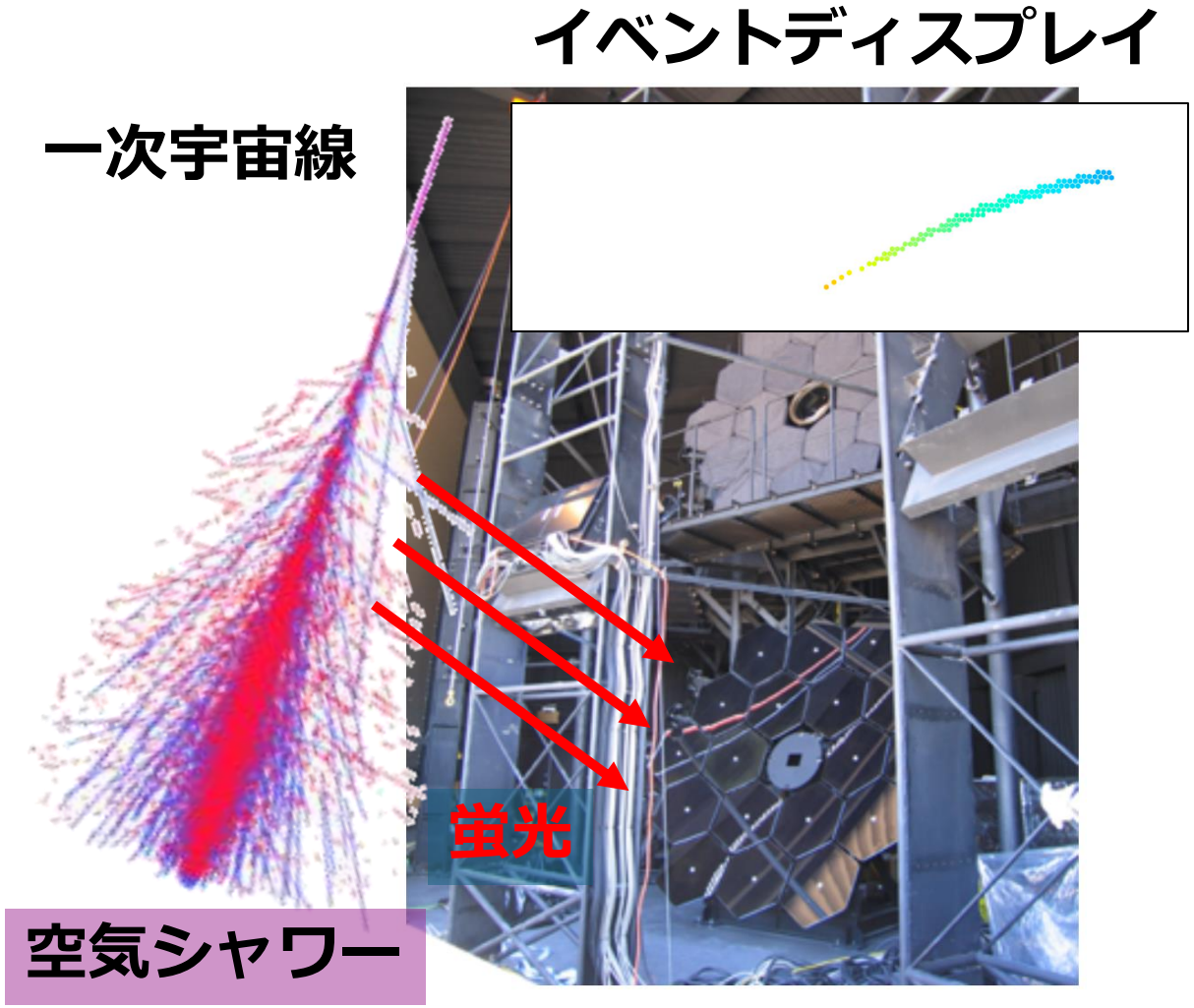


FD：青六角形
SD：緑四角形

FD

SD

超高エネルギー宇宙線の観測手法



- 宇宙線が地球に到来した際に、地球の大気と相互作用することで起こる空気シャワー現象を観測
- 大気蛍光望遠鏡は、荷電粒子が大気と相互作用した際に発生する大気蛍光を球面鏡でカメラに集光し観測
- 空気シャワー現象より観測した、光子数や入射時間などから宇宙線のエネルギーや到来方向などを求めることを再構成という
- 大気蛍光望遠鏡の視野内を通過した空気シャワーの軌跡を画像化したものをイベントディスプレイという

研究目的

現状の大気蛍光望遠鏡の再構成では時間を要する

再構成の時間を短縮することで、リアルタイム解析が可能になり

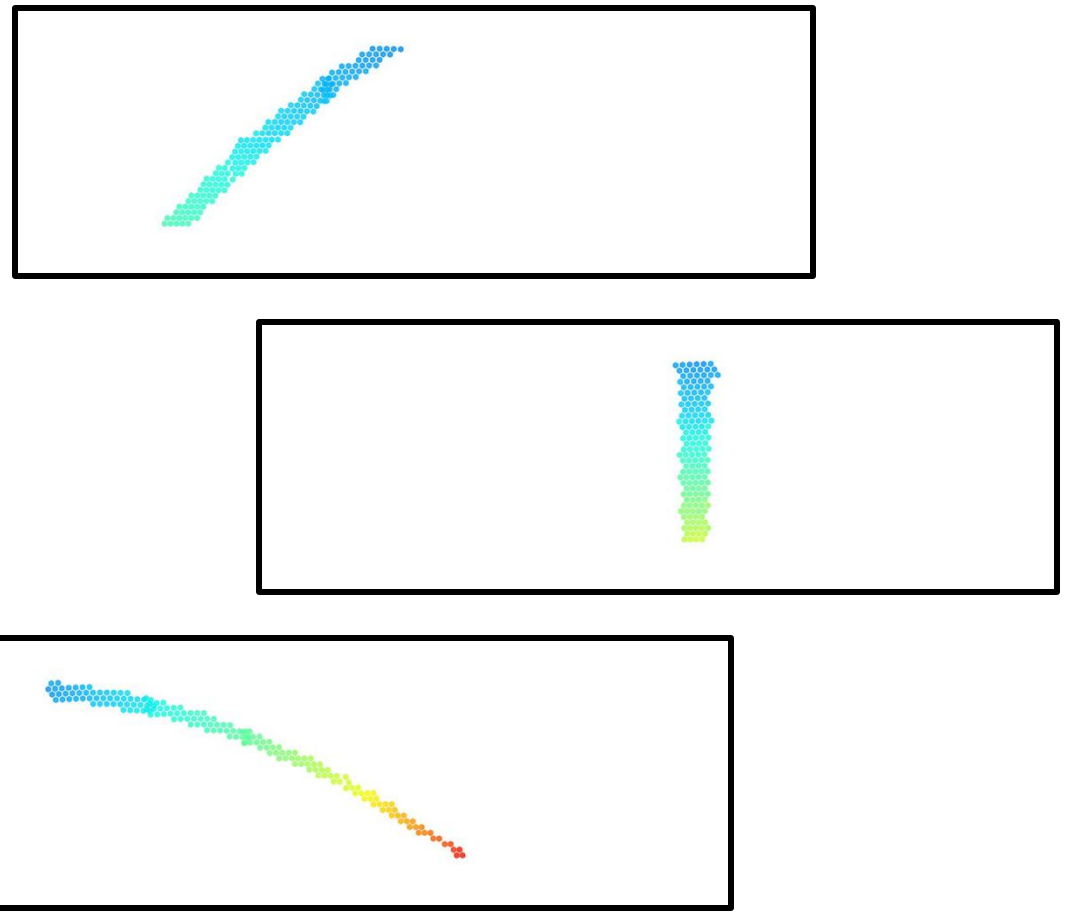
マルチメッセンジャー天文学への貢献が期待できる

イベントディスプレイ を学習用データとし、
深層学習を用いた再構成手法の開発

研究手順

1. 空気シャワーシミュレーションを行い、疑似データを作成
2. 作成した疑似データから、イベントディスプレイを作成
3. イベントディスプレイから学習のためのデータセットを作成
4. 空気シャワーの画像をマシンに入力し、画像の特徴を学習させる
5. 学習した特徴から空気シャワーのパラメータを推定する
6. 推定した値と真値の比較などを行いモデルの評価を行う

イベントディスプレイ

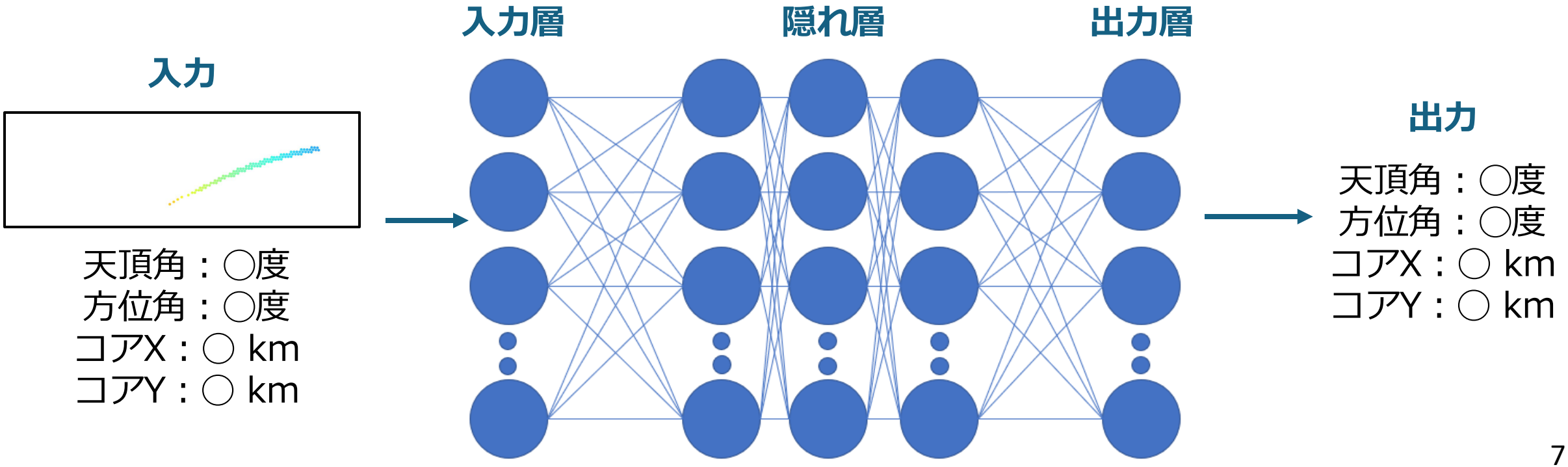


シミュレーションより得られた様々な
ジオメトリの空気シャワーイベントディスプレイ

- 大気蛍光望遠鏡からは各PMTに入射した蛍光量と時間の情報が得られ、その情報を用いて空気シャワーの軌跡を可視化した画像
- 左図のカラーマップは時間経過を表し、青から赤に向けて時間が経過している
- 作成の際、信号が入った PMT の本数が 15 本以上、各 PMT カメラの信号を検出した PMT の本数が 256 本中 128 本以下という条件をつけている

深層学習

- 人間の脳の仕組みを再現したニューラルネットワークを用いた機械学習の手法の一つ
- 学習データ (入力) が十分であれば、データの特徴を自動で抽出
- 層を多層にすることができ、データに含まれる特徴をより深く学習
- 本研究で扱うモデルは回帰モデルであり、大量のデータに最も沿うような回帰式を求めて数値を予測する学習手法



モデル構造の選定

現在のモデル構造

ジオメトリ（天頂角、方位角、シャワーコア）の推定が可能

エネルギーの推定を行えない

エネルギー推定に必要な特徴量が不足している

必要な特徴量を学習させたいが、現在のモデル構造では不可能

モデル構造の選定、実装を行った

モデル構造の選定

◆Sequential API
- 層を順番に積み重ねるシンプルなモデル構築
- コーディングが容易に行える
- 分岐・マルチ入力ができないなどの制約あり

◆Functional API
- 層を関数のように接続するモデル構築
- 柔軟なモデル構築が可能（分岐・マルチ入力・スキップ接続）
- モデルの構造が明確で、エラーが出たときに特定しやすい

◆Model Subclassing API
- 柔軟かつカスタマイズ性の高いモデル構築
- 他のAPIでは実現しにくい構造も自在に設計できる（条件分岐・ループ）
- モデルの構築難易度が高く、既存のメソットが動作しないことがある

モデル構造の選定

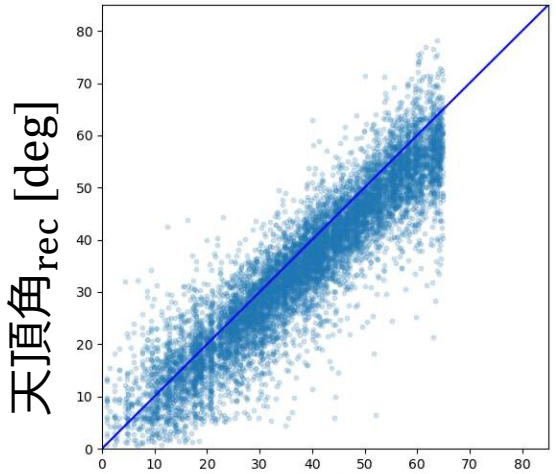
◆Sequential API
- 層を順番に積み重ねるシンプルなモデル構築
- コーディングが容易に行える
- 分岐・マルチ入力ができないなどの制約あり

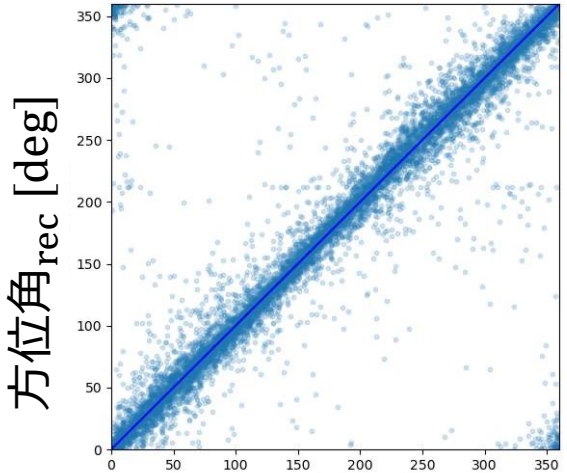
◆Functional API
- 層を関数のように接続するモデル構築
- 柔軟なモデル構築が可能（分岐・マルチ入力・スキップ接続）
- モデルの構造が明確で、エラーが出たときに特定しやすい

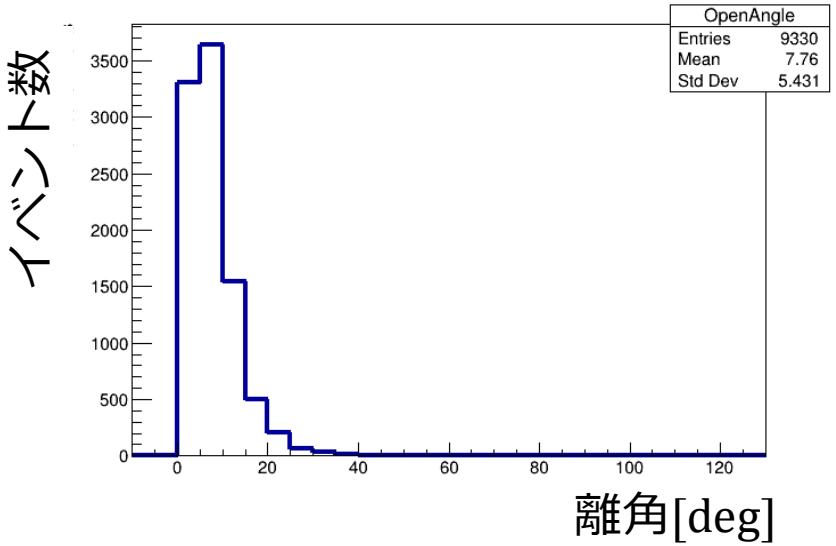
◆Model Subclassing API
- 柔軟かつカスタマイズ性の高いモデル構築
- 他のAPIでは実現しにくい構造も自在に設計できる（条件分岐・ループ）
- モデルの構築難易度が高く、既存のメソットが動作しないことがある

解析精度の比較：天頂角、方位角

Sequential API （旧）

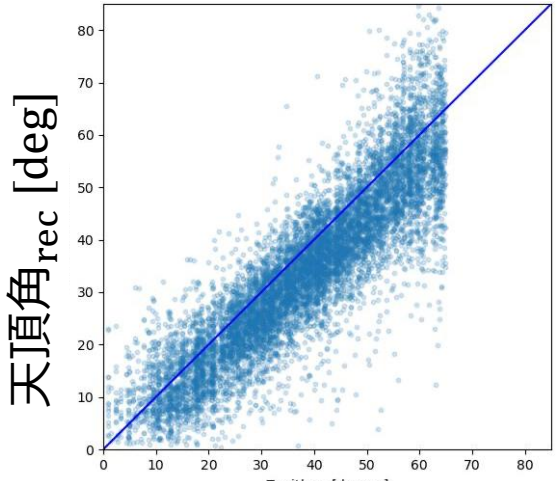


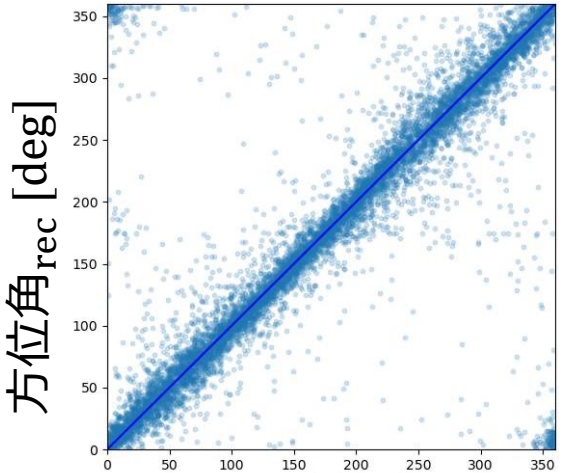


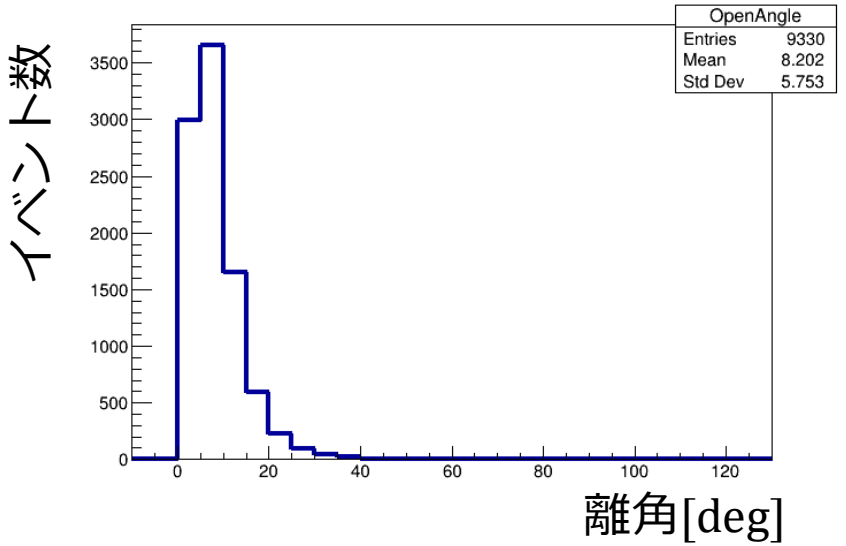


解析精度
8.9 deg

Functional API （新）



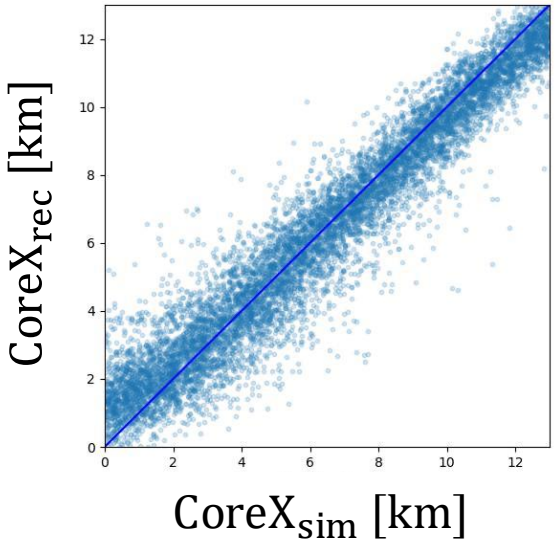




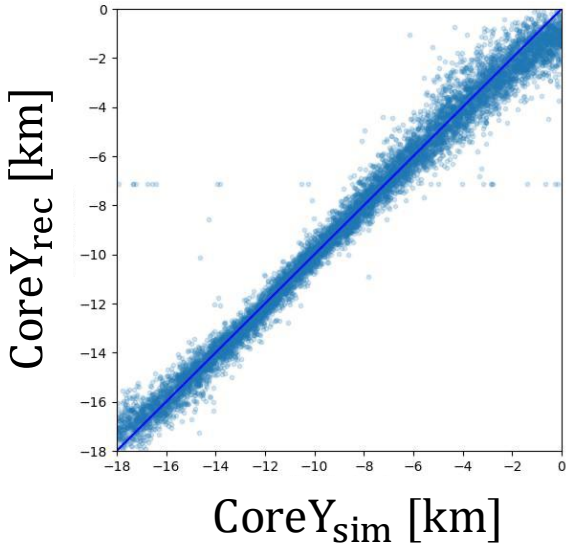
解析精度
9.3 deg

解析精度の比較：シャワーコア

Sequential API （旧）

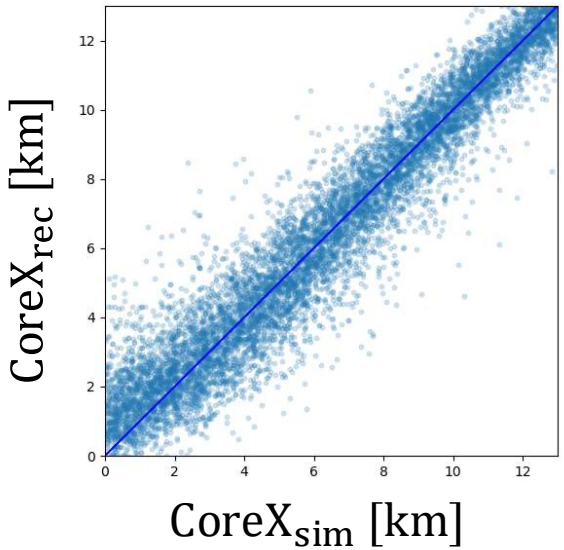


解析精度
-0.03 ± 1.0 km

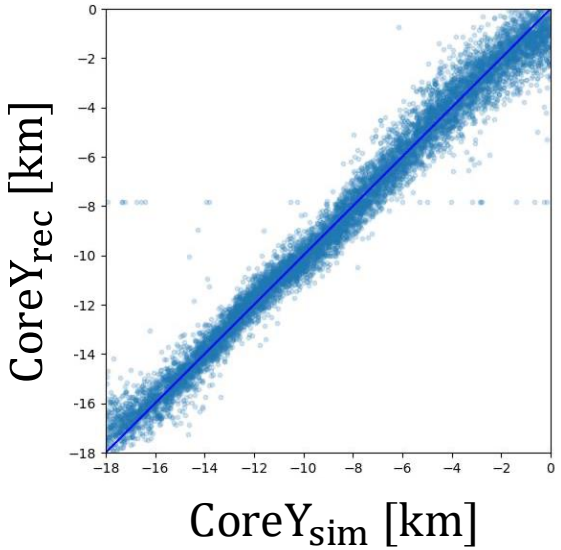


解析精度
-0.04 ± 0.6 km

Functional API （新）



解析精度
-0.09 ± 1.0 km



解析精度
-0.03 ± 0.7 km

まとめと今後

◆まとめ

- Functional API を用いたモデル構造の実装

	離角	CoreX	CoreY
Sequential API (旧)	$8.9\ \mathrm{deg}$	$-0.03\ \pm 1.0\ \mathrm{km}$	$-0.04\ \pm 0.6\ \mathrm{km}$
Functional API (新)	$9.3\ \mathrm{deg}$	$-0.09\ \pm 1.0\ \mathrm{km}$	$-0.03\ \pm 0.7\ \mathrm{km}$

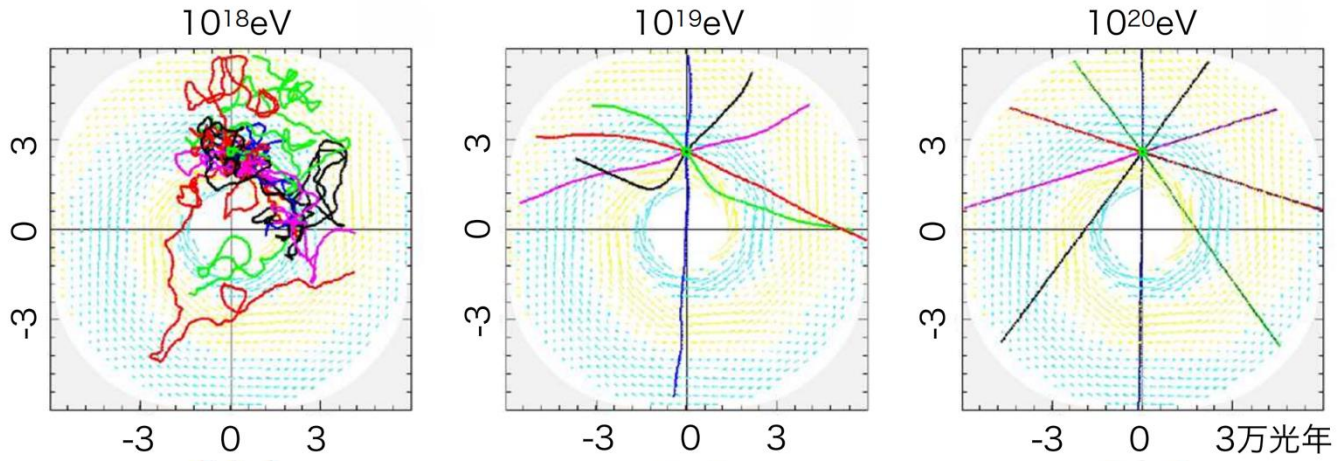
◆今後

- マルチ入力モデルの作成
- エネルギー用のイベントディスプレイの選定
- 解析可能パラメータにエネルギーなどを追加

Buck up

超高エネルギー宇宙線

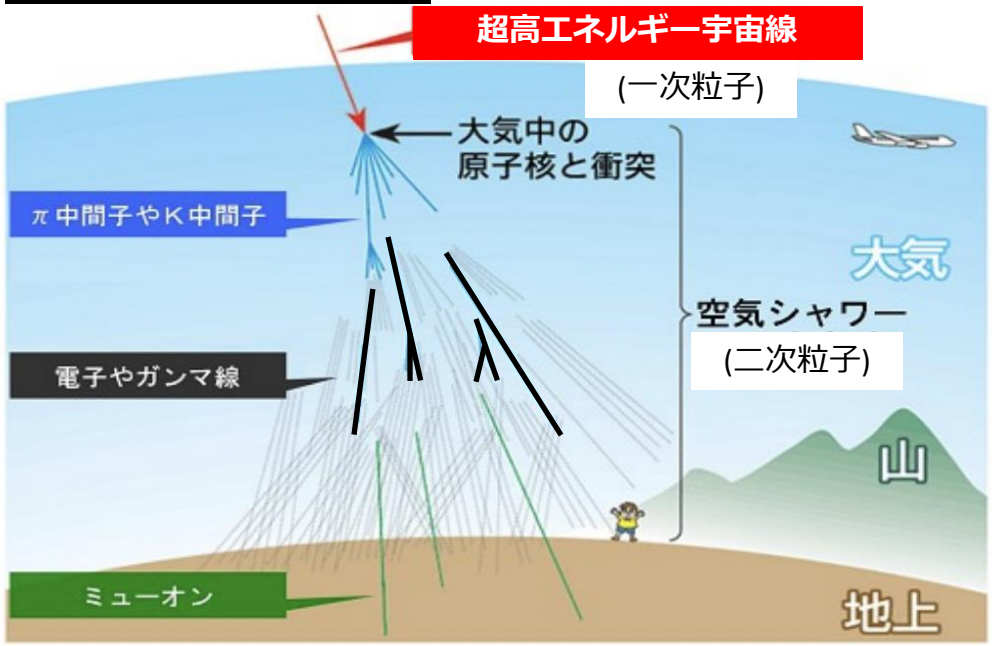
- 宇宙空間から降り注ぐ高エネルギーの放射線
- 荷電粒子であるため、宇宙空間の磁場の影響を受け、軌道が曲げられる
- 10^{18} eV 以上のエネルギーを持つものを**超高エネルギー宇宙線**という
  - エネルギーが高いほど磁場の影響を受けにくく、ほぼ真っ直ぐ進むことができる
  - 起源天体の解明が期待されている
- マルチメッセンジャー天文学の参入も期待されている



宇宙線の軌道のシミュレーション

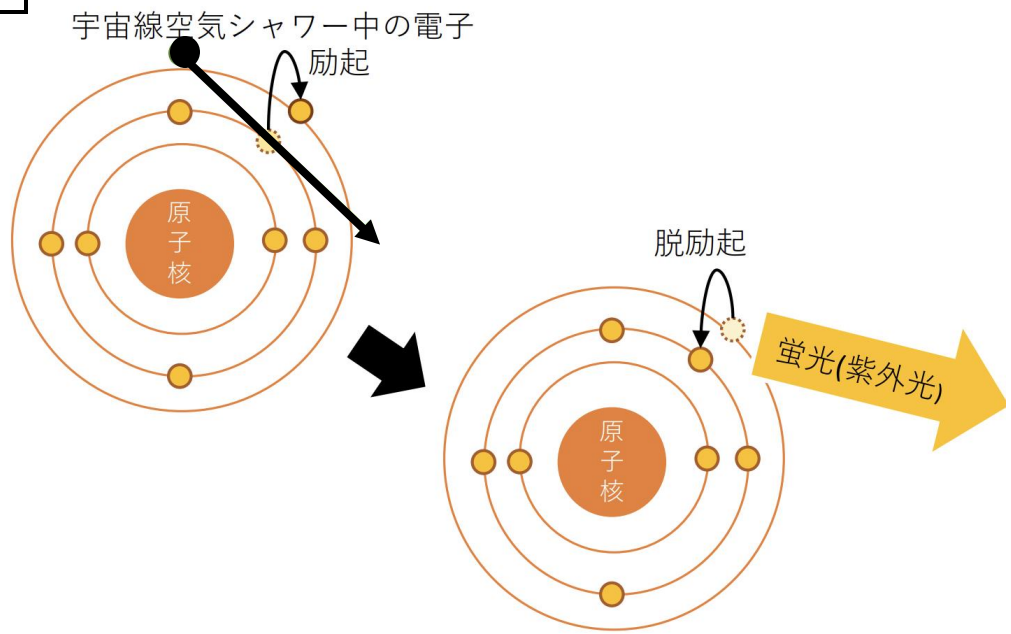
空気シャワー現象

空気シャワー現象



1. 超高エネルギー宇宙線が大気に入射する
2. 大気中の原子核と相互作用を起こす
3. 相互作用により粒子が生成される
4. 2. と 3. が繰り返され、たくさんの粒子を生成する

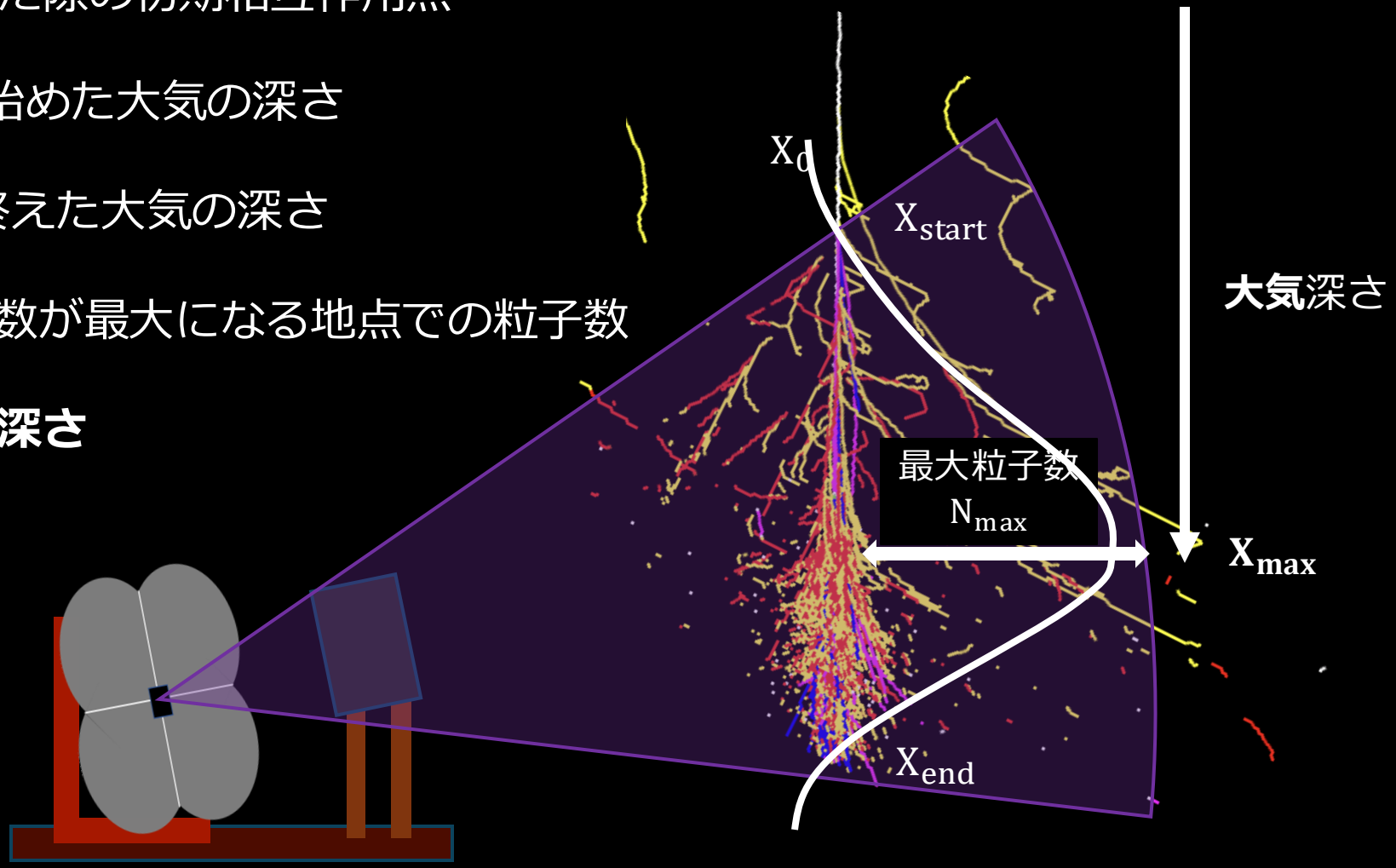
大気蛍光



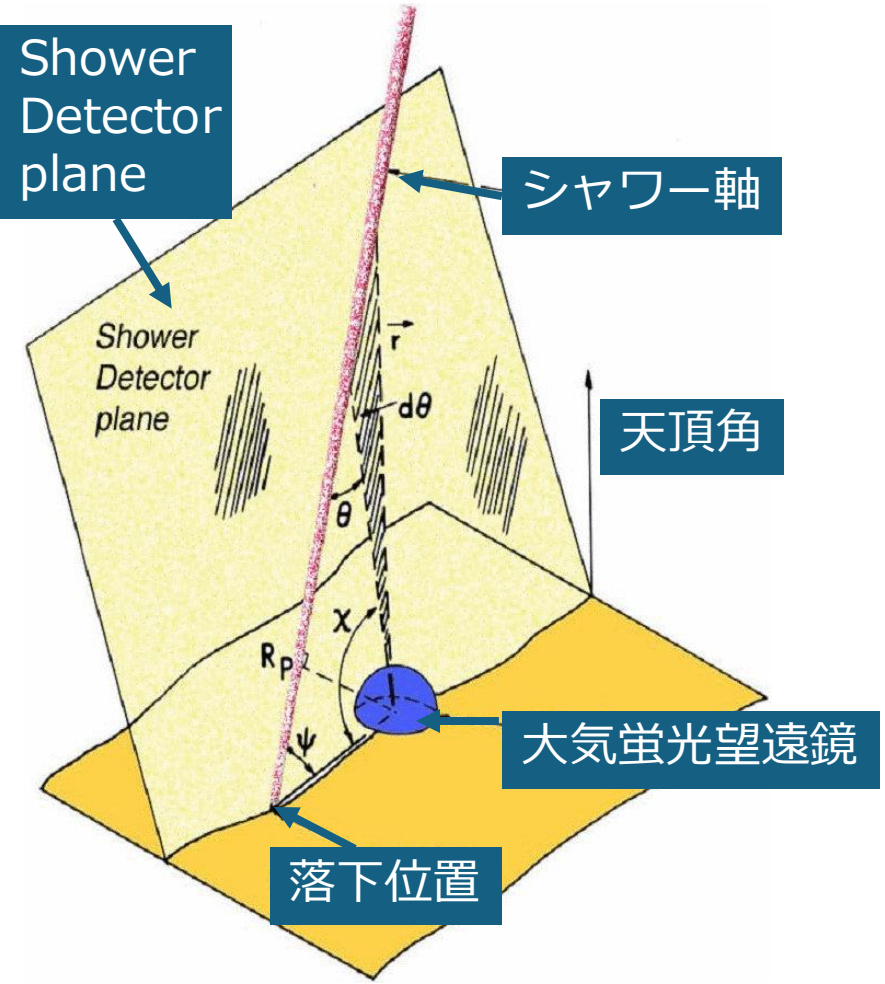
1. 空気シャワー中の粒子はほとんどが電子であり、
   大気中の窒素などが持つ電子と相互作用し、
   電子がエネルギーを得て励起する。
2. 電子の状態が不安定になり、脱励起をおこし蛍光を放出

空気シャワーを測定して得られるパラメータ

- 大気深さ: 大気の最上層から地表までの垂直方向の距離
- X_0: シャワーを GH フィットした際の初期相互作用点
- X_{start}: FD がシャワーを観測し始めた大気の深さ
- X_{end}: FD がシャワーを観測し終えた大気の深さ
- N_{max}: シャワー中の二次粒子の数が最大になる地点での粒子数
- **X_{max}: 粒子数が最も多い大気の深さ**



大気蛍光望遠鏡の再構成手法



シャワージオメトリ

再構成手順

1. 信号の入ったPMTの選別
   - PMTの信号は宇宙線の蛍光によるものの他に、夜光などのノイズも含まれるため。
2. シャワー検出器平面の決定
   - シャワー検出面を決まれば、シャワー軸を求めることが簡単になる。
3. シャワージオメトリ (天頂角、方位角、落下位置) の決定
4. モンテカルロシミュレーションとデータを比較し、空気シャワーの縦方向発達の再構成

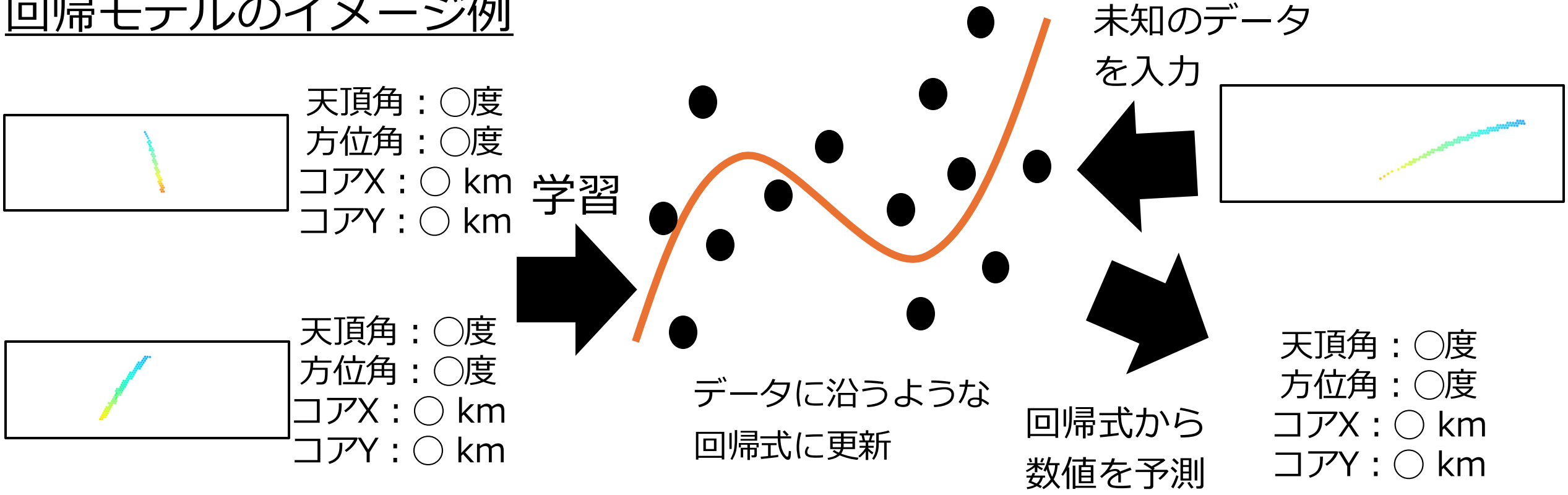
大量のシミュレーション結果と比較を行うため時間がかかる

回帰モデル

本研究で扱うモデルは回帰モデルであり、大量のデータに最も沿うような回帰式を求めて数値を予測する学習手法である

なお、本モデルは教師あり学習である

回帰モデルのイメージ例



Sequential API

設定学習回数　：50
学習率.　：0.001
層.　：9
学習データ.　：35000
検証データ.　：1500
テストデータ　：1000

```
model = Sequential()

weight_decay = 1e-7

model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same',
          input_shape=(ximage, yimage, color_setting),
          kernel_regularizer=regularizers.l2(weight_decay),
          activation='relu'))
model.add(Conv2D(128, (3, 3), padding='same',
          kernel_regularizer=regularizers.l2(weight_decay),
          activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), padding='same',
          kernel_regularizer=regularizers.l2(weight_decay),
          activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(256, activation='relu'))
model.add(Dense(4, activation='linear'))   # change later to variable

learning_rate = 0.001 #学習率

model.compile(loss='mse', optimizer=Adam(lr=learning_rate), metrics=['mse'])

model.summary()
```

Functional API

```
#モデル構造

weight_decay = 1e-7

#inputs = Input(shape=(96, 32, 3))
inputs = Input(shape=(ximage, yimage, color_setting))

x = Conv2D(filters=64, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizers.l2(weight_decay), activation='relu')(inputs)
x = Conv2D(filters=64, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizers.l2(weight_decay), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(filters=256, kernel_size=(3, 3), padding='same', kernel_regularizer=regularizers.l2(weight_decay), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)

outputs = Dense(4, activation='linear')(x)

model = Model(inputs=inputs, outputs=outputs)

learning_rate = 0.001
model.compile(loss='mse', optimizer=Adam(learning_rate=learning_rate), metrics=['mse'])

model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

history = model.fit(x_train, y_train, batch_size=32, epochs=50, verbose=1, validation_data=(x_test, y_test), callbacks=[early_stopping])
```