解析ツールとのインターフェイス

Geant4 10.2準拠

Geant4　HEP/Space/Medicine 講習会資料

KEK 大学共同利用機関法人 高エネルギー加速器研究機構

本資料に関する注意

- 本資料の知的所有権は、高エネルギー加速器研究機構およびGeant4 collaborationが有します
- 以下のすべての条件を満たす場合に限り無料で利用することを許諾します
  - 学校、大学、公的研究機関等における教育および非軍事目的の研究開発のための利用であること
    - Geant4の開発者はいかなる軍事関連目的へのGeant4の利用を拒否します
  - このページを含むすべてのページをオリジナルのまま利用すること
    - 一部を抜き出して配布したり利用してはいけません
  - 誤字や間違いと疑われる点があれば報告する義務を負うこと
- 商業的な目的での利用、出版、電子ファイルの公開は許可なく行えません
- 本資料の最新版は以下からダウンロード可能です
  - http://geant4.kek.jp/lecture/
- 本資料に関する問い合わせ先は以下です
  - Email: lecture-feedback@geant4.kek.jp

目標

- 一般に、HEP実験では、実験生データ(raw data)からエッセンスだけを絞り出したDST(Data Summary Tape)を作り、それを解析する手順をとる
  - 数百万から数千万チャンネルのADCデータのほとんどは意味あるデータを持たないし、圧縮したDSTでも1イベントあたりのデータ量は大きくばらつく。
  - このようなデータの編成の方法はいろいろ
    - 例えばROOTではTTREEなど
- 本講義では、Geant4に簡単なデータ構造の「DSTデータ」を生成させ、それらを外部解析ツールで扱えるように書き出す手法を学ぶ。
  - SteppingActionやEventActionでヒストグラム(一次元や二次元)を作る一番初歩的な方法
  - 興味ある物理量のNtupleファイルを作る方法
- 解析ツールROOTは仮想マシンにインストールされているので自習のこと

目次

1. Geant4と解析ツール
    1. analysis カテゴリー
2. 解析の流れをコードで読む
    1. 解析マネージャと解析ツールへの接続の選択
    2. ユーザアクションでやること
3. 関数群
    1. ヒストグラムとntuple、定義、Fill, 出力
4. /analysis/ コマンド
5. 解析ツールROOTと表計算
6. 参考となるexamples

[謝辞] 本講義で使用している資料は、過去にSLAC、CERN、IN2P3、ESAなどが主催したGeant4チュートリアルで使用されたスライドの内容を多く含みます。これらのスライド作成に寄与したGeant4 Collaborationメンバーに謝意を表します。

Geant4と解析ツール

analysisカテゴリ

- Geant4は外部ソフトを使う立場であって、外部ソフトに依存はしないという基本方針を貫いている
  - 解析ツールへのドライバとしてはAIDA(Abstract Interface for Data Analysis)を提供していたが、従来のAIDA は維持が十分でなく、インストールや利用についても不満が多かった。
- Geant4の10.0版からはg4toolsを使う新しいインターフェイス を採用し、analysisカテゴリーを設けた
  - 次の出力ファイル形式とがいぶソフトに対応する
    - CSV 形式: 表計算ソフト(Office, LibreOffice/OpenOfficeなど), ROOT, GnuPlot
    - ROOT形式 : ROOT
    - XML形式 : JAS, iAIDA, OpenScientist, rAIDA
    - HBOOK形式: HBOOK

解析インターフェイスの特長

- Analysis カテゴリーに解析マネージャG4AnalysisManagerを置き、使用する解析ツールによらずに共通の方法で以下の操作ができる関数群を提供する
  - ヒストグラムの定義
  - ヒストグラムへの記入
  - Ntuple の定義
  - ntupleへの追加
  - ファイルへの出力など
- G4toolsパッケージへの統一的なインターフェイスとして提供されているので、解析ツールによって異なる外部実装コードへの依存関係はない
- 会話型のコマンドの利用もできる
  - /analysis/
- どの形式を選ぶかはanalysis.hh定義ファイルのなかで
  - g4root.hh, g4xml.hh または g4csv.hh の一つを include するだけ
    - 外部ライブラリは必要ない
- マルチスレッドでもスレッドごとの結果を自動的に集めてくれる

解析結果に関する現在の制限

- ✧ Histogram の種類: 1D, 2D, 3D 倍精度数データ
- ✧ Profile の種類: 1D, 2D　倍精度数データ
- ✧ Ntuple のカラムデータの種類: int, float, double, G4String　のデータ
  - ✧ 新規追加　std::vector<int>, std::vector<float>, std::vector<double>
- ◆ ディレクトリに分けて管理する場合、ヒストグラム用とNtuple用に各々一つに限られる。例えば
  - ◆ myHisto/energy1, energy2, position1, …..
  - ◆ myNtuple/hit1, hit2,
- ◆ このカテゴリーは精力的に開発が進められているので最新のリリースノートを見ておくと良い

解析コードの流れ:

解析利用の段取り

- 段取り
  1. 解析に関わるユーザアクション定義クラス(例えばRunAction.hh)のなかで、使用する解析ツールに対応するヘッダファイルanalysis.hh をincludeし、
  2. Step/Event/Runなどユーザアクションの中でその都度解析マネージャG4AnalysisManagerのインスタンスを作り、解析マネージャのメソッド(定義, Fill, 保存など)を呼び出す
- 関係するユーザアクション
  1. ユーザRunActionのコンストラクタ
     - ランの始めに解析マネージャインスタンスを呼び出してヒストなどを定義
  2. UserRunAction::BeginOfRun()
     - 出力ファイル名前の指定とファイルオープンを行う
  3. ユーザSteppingActionやEventActionのSensitiveDetector::ProcessHits(), EndOfEvent()
     - 解析マネージャのインスタンスを呼び出してFillする
  4. ユーザEventAction::EndOfEvent
     - ヒストをフィル(fill)する
  5. UserRunAction::EndOfRun
     - 解析マネージャのインスタンスを呼び出して
     - ランの終わりにヒストを書き出す
  6. ユーザランアクションのデストラクタで解析マネージャをキルする
- ランを終了したら結果がファイルに出力される

解析マネージャの選択

例えばROOTマネージャを使うには
analysis.hh　でその行を生かす！！

```
#include "g4root.hh"
//#include "g4xml.hh”
//#include “g4csv.hh”
```

解析マネージャのメソッドを使うときは、その都度インスタンスを呼び出す

```
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
```

二つまでなら解析マネージャを同時に使えるが、別々のインスタンスを呼び出すには

```
#include "G4CsvAnalysisManager.hh"
#include "G4XmlAnalysisManager.hh"

G4CsvAnalysisManager* csvManager = G4CsvAnalysisManager::Instance();
G4XmlAnalysisManager* xmlManager = G4XmlAnalysisManager::Instance();
```

ランの初めに出力ファイルを準備する

```
Void RunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  // Open an output file
  //
  G4String fileName = "P05";   // ファイルの種別を表す記述子root, csvなどが自動的に付加される
  analysisManager->OpenFile(fileName);
}
```

ランの後始末とランアクションのデストラクタ

- EndOfRunAction

```
Void RunAction::EndOfRunAction(const G4Run* aRun)
{
// save histograms & ntuple
  //
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
     analysisManager->Write();
     analysisManager->CloseFile();          出力ファイル名　“P05.root”

他の報告など
}
```

```
RunAction::~RunAction()
          デストラクタで解析マネージャのシングルトンを殺す
{
  delete G4AnalysisManager::Instance();
}
```

RunAction　コンストラクタでヒストなどを用意する

■　ランアクションのコンストラクタで解析マネージャを呼び出して準備する

```
RunAction::RunAction()
 : UserRunAction()
{
 // Create analysis manager
  // The choice of analysis technology is done via selectin of a namespace
  // in analysis.hh
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
                G4cout << "Using " << analysisManager->GetType() << G4endl;
       analysisManager->SetVerboseLevel(1);
       analysisManager->SetFirstHistoId(1);                ヒストのIdを１から始める
 // Creating histograms
     analysisManager->CreateH1("1","Edep", 100, 0., 1000*MeV);
// Creating ntuple  カラム指定は未完のまま
  //
  //   analysisManager->CreateNtuple(”BGO", "Edep and TrackLength");
  //   analysisManager->FinishNtuple();
}
```

ステップの中やイベントの終わり毎にfill

- EndOfEventActionから関係部分を一部抜粋。ステップごとにエネルギー付与を求め、積算してイベントの最後にヒストに記入する

```
void EventAction::BeginOfEventAction(const G4Event* /*event*/)
{
   sum_eDep = 0.;
}

void EventAction::EndOfEventAction(const G4Event* /*event*/)
{
  G4cout << "sum_eDep = "<< sum_eDep << G4endl;
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
   analysisManager->FillH1(1, sum_eDep);


}
```

解析マネージャの関数群

定義して、Fillして、出力する

一次元ヒストグラム

- 書式　CreateH1(“name”, “title”, nBins, xmin, xmax
  [, unitName=“none”, fcnName=“none”])

  unitName, fcnNameは省略可能で、使い方はマニュアル9.2.1.5
- ヒストグラム識別子 id
  - G4AnalysisManager::CreateH1()でヒストグラムを作った時に自動的に作られ、戻り値となる。規定値は0から始まり、新しいヒストを作る毎に1増える。
  - この識別子がfillするときに使われる
    - FillH1(id, value, weight)
  - 注意！CreateH1() の引数で与える文字列のnameは識別子とは無関係
- ヒストグラムオブジェクト
  - G4AnalysisManager::GetH1(G4int id) でオブジェクトが得られる
  - ヒストグラムオブジェクトではmean(), rms()　などが使える

１次元ヒストグラムの操作関数一覧

```
G4int CreateH1(const G4String& name, const G4String& title,
               G4int nbins, G4double xmin, G4double xmax,
               const G4String& unitName = "none",
               const G4String& fcnName = "none",
               const G4String& binSchemeName = "linear");
G4bool SetH1(G4int id,
               G4int nbins, G4double xmin, G4double xmax,
               const G4String& unitName = "none",
               const G4String& fcnName = "none",
               const G4String& binSchemeName = "linear");
G4bool SetH1Title(G4int id, const G4String& title);
   G4bool SetH1XAxisTitle(G4int id, const G4String& title);
   G4bool SetH1YAxisTitle(G4int id, const G4String& title);
G4bool FillH1(G4int id, G4double value,  G4double weight = 1.0);
G4int GetH1Id(const G4String& name, G4bool warn = true) const;
```

二次元ヒストグラム

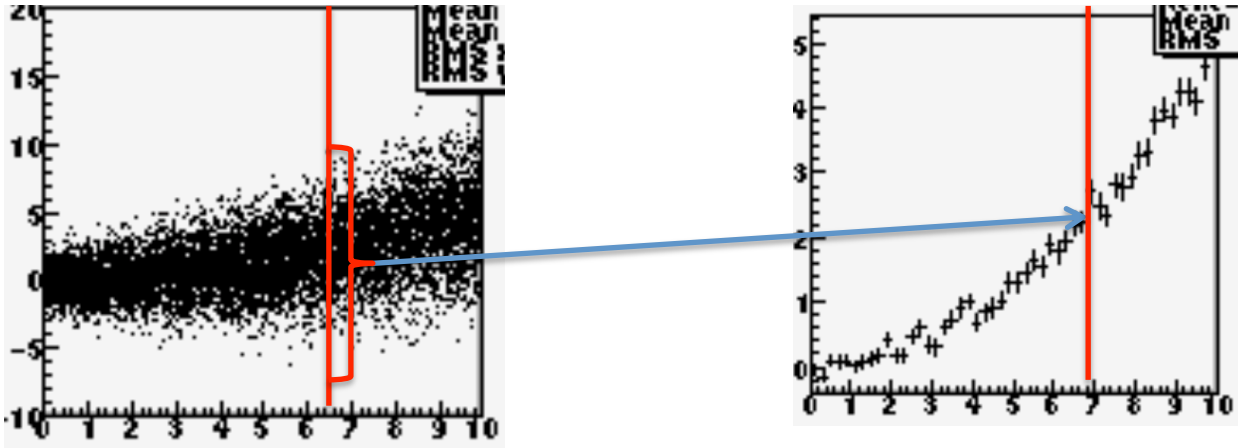
- CreateH2, SetH2

```
G4int CreateH2(const G4String& name, const G4String& title,
               G4int nxbins, G4double xmin, G4double xmax,
               G4int nybins, G4double ymin, G4double ymax,
               const G4String& xunitName = "none",
               const G4String& yunitName = "none",
               const G4String& xfcnName = "none",
               const G4String& yfcnName = "none",
               const G4String& xbinScheme = "linear",
               const G4String& ybinScheme = "linear");

G4bool SetH2(G4int id,
             G4int nxbins, G4double xmin, G4double xmax,
             G4int nybins, G4double ymin, G4double ymax,
             const G4String& xunitName = "none",
             const G4String& yunitName = "none",
             const G4String& xfcnName = "none",
             const G4String& yfcnName = "none",
             const G4String& xbinSchemeName = "linear",
             const G4String& ybinSchemeName = "linear");
```

プロファイル　P1

- P1では二次元プロットのYの平均値とYのσをXのビンへ入れる
  （KEK藤井さんの「猿にも使えるROOT」からの実例



```
G4int CreateP1(const G4String& name, const G4String& title,
               G4int nbins, G4double xmin, G4double xmax,
               G4double ymin = 0, G4double ymax = 0,
               const G4String& xunitName = "none",
               const G4String& yunitName = "none",
               const G4String& xfcnName = "none",
               const G4String& yfcnName = "none",
               const G4String& xbinSchemeName = "linear");
```

Ntuple

- 作る 4つのデータ型またはベクトル型のカラムが作れる
  - NtupleのIDと名前
    1. CreateNtuple(“name”, “title”)　　ntuple ID = 0から
  - カラムの定義
    1. CreateNtupleDColumn(“column_1_Name”) column ID = 0から
    2. CreateNtupleIColumn(“column_2_Name”) column ID = 1
    3. CreateNtupleFColumn(“column_3_Name”) column ID = 2
    4. CreateNtupleSColumn(“column_4_Name”) column ID = 3
    5. G4int CreateNtupleDColumn(
       “column_5_Name”, std::vector<double>& vector); column ID = 4
  - このNtuple 定義のおしまい
    2. FinishNtuple()
  - Ntuple column ID は作られた順序に付けられる
  - カラムに入れるデータの型に応じてメソッドが異なる。現在4つのデータ型だけD倍精度、I整数、F単精度、S文字列型をサポートしている
  - 複数のNtupleを作るときはCreateNtuple(G4int id, const G4String& name);
- Fill
  - FillNtupleDColumn(id, value) ここでもデータ型を明示的に含む関数名

ヒストグラムの活性化／不活性化

- 最初にすべてのヒストグラムを活性化せずに作っておいて、アプリの実行時にコマンドで必要なものだけを活性化することができる。こうすれば、必要で無い結果の出力を抑制できる。

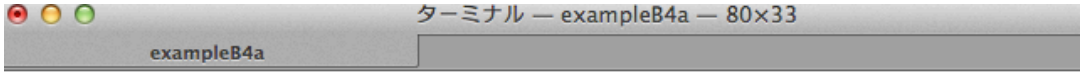
ランマネージャのコンストラクタで

```
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
 analysisManager->SetActivation(true);
 // define histogram parameters name, title, nbins, vmin, vmax
 G4int id = analysisManager->CreateH1(name, title, nbins, vmin, vmax);
 analysisManager->SetH1Activation(id, false);
```

実行開始後のコマンドを使ってもできる

```
/analysis/h1/set 1 100 0   50 cm
/analysis/h1/setActivation 1 true
```

/analysis/ コマンド




```
Guidance :
analysis control

 Sub-directories :
   /analysis/h1/   1D histograms control
   /analysis/h2/   2D histograms control
 Commands :
   setFileName * Set name for the histograms & ntuple file
   setHistoDirName * Set name for the histograms directory
   setNtupleDirName * Set name for the ntuple directory
   setActivation * Set activation.
When this option is enabled, only the histograms marked as activated
are returned, filled or saved on file.
No warning is issued when Get or Fill is called on inactive histogram.
   verbose * Set verbose level
Idle> ls /analysis/h1
Command directory path : /analysis/h1/


Guidance :
1D histograms control

 Sub-directories :
 Commands :
   create * Create 1D histogram
   set * Set parameters for the 1D histogram of #Id :
   setAscii * Print 1D histogram of #Id on ascii file.
   setTitle * Set title for the 1D histogram of #Id
   setXaxis * Set x-axis title for the 1D histogram of #Id
   setYaxis * Set y-axis title for the 1D histogram of #Id
   setActivation * Set activation for the 1D histogram of #Id
   setActivationToAll * Set activation to all 1D histograms.
```

解析ツール

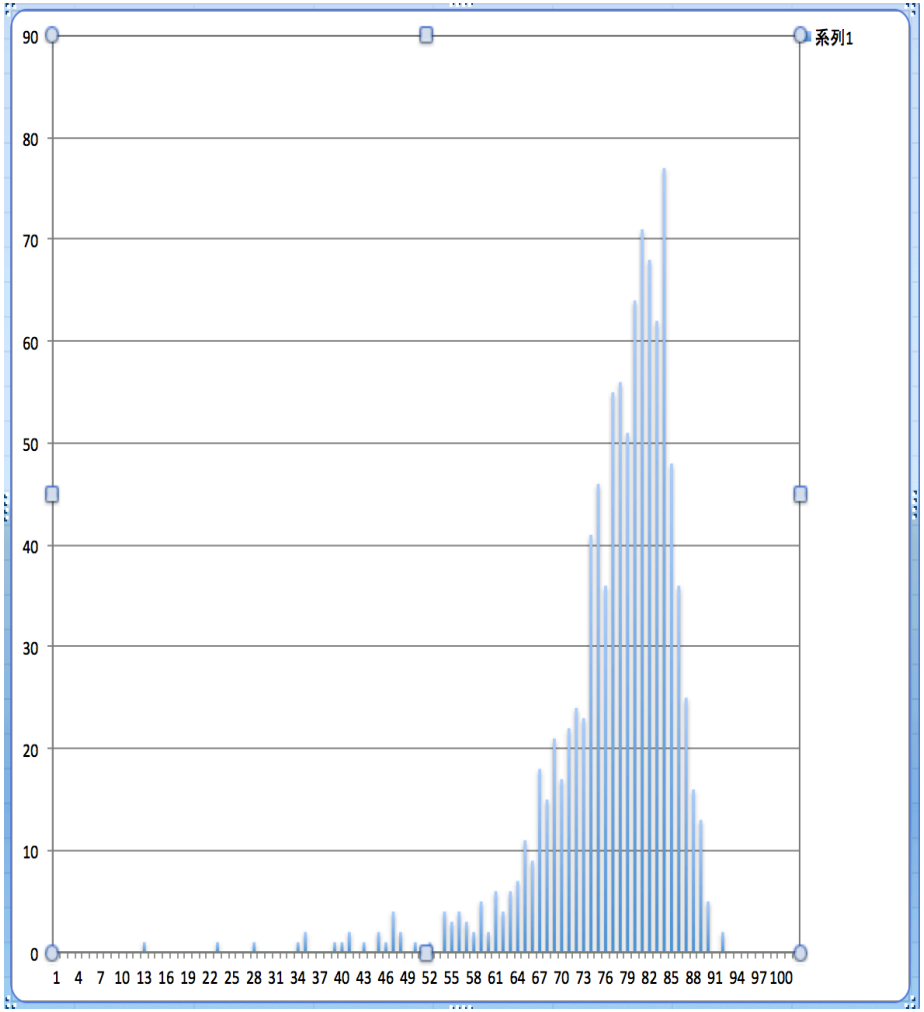
P05例題のCSVファイル出力からExcelでグラフ作成

1 GeV 電子のeDep分布だが、表計算では色々と足りないものがある。例えば

- 横軸はエネルギーでなく行番号
- 目盛りの数値が気に食わない
- 誤差を重ねて表示できない
- 関数当てはめが単純なものだけ
- などなど

やはりROOTを使うと発表用のグラフが簡単にできるなど、HEP専用の機能が使える。変更は次のファイルで一行だけ

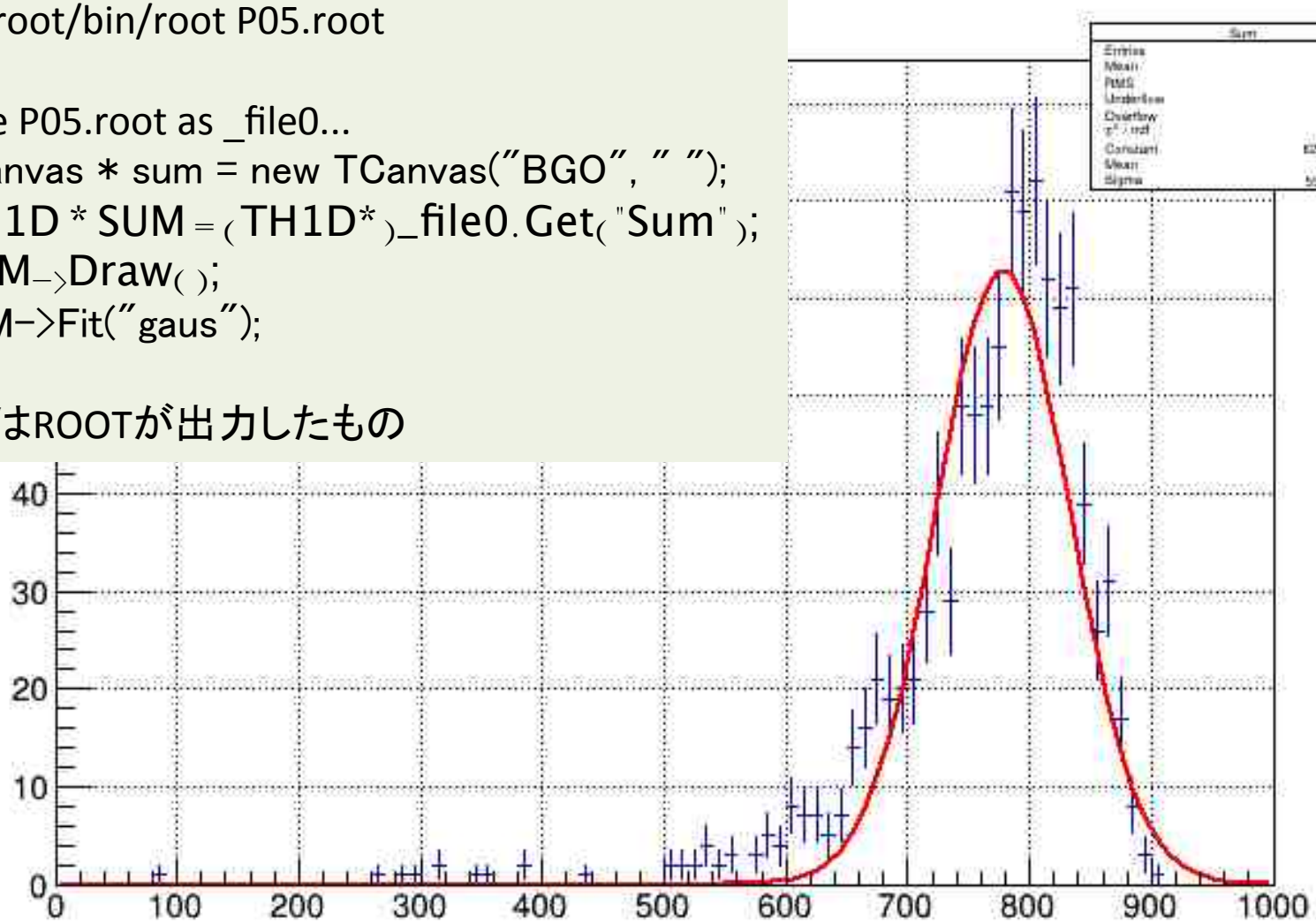
- Analysis.hh



P05例題でROOTを使う

```
yoshida% ~/root/bin/root P05.root
root [0]
Attaching file P05.root as _file0...
root [1] TCanvas * sum = new TCanvas("BGO", " ");
root [2] TH1D * SUM = (TH1D*)_file0.Get("Sum");
root [3] SUM->Draw();
root [4] SUM->Fit("gaus");
```

右のショットはROOTが出力したもの



ROOT解析例： B4 サンプリングカロリメータ

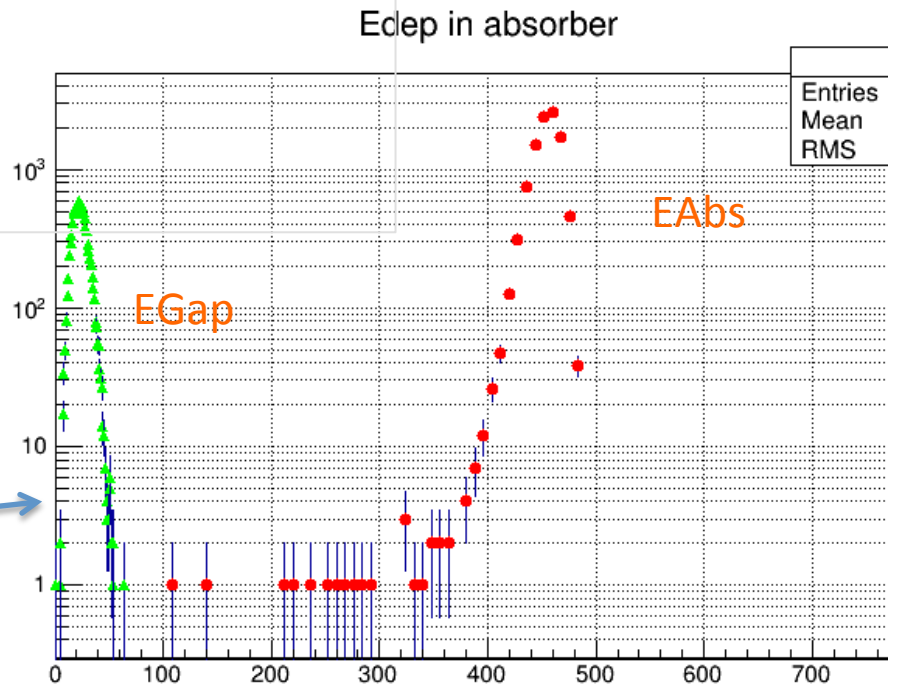
%./exampleB4a –m run3.mac ← 電子 500 MeV 10,000 events

---> End of event: 9900
Absorber: total energy: 437.658 MeV total track length: 31.6785 cm
Gap: total energy: 23.653 MeV total track length: 12.0724 cm

----> print histograms statistic

EAbs : mean = 453.583 MeV rms = 15.3121 MeV
EGap : mean = 23.7266 MeV rms = 7.11997 MeV
LAbs : mean = 32.4348 cm rms = 1.22231 cm
LGap : mean = 11.8072 cm rms = 3.67781 cm

B4.root
ファイル出力をROOTで処理
重ね描き



ROOTの使い方 藤井さんの「猿にも」で自習する

■ B4.rootをROOTに食わせる

```
[:~/G4Apps/B4-build/B4d] myName% root B4.root
..(省略)
root [0]
Attaching file B4.root as _file0...
root [1] .ls
TFile**         B4.root
 TFile*         B4.root
  OBJ: TH1D     1     Edep in absorber : 0 at: 0x7fc631f069f0
  OBJ: TH1D     2     Edep in gap : 0 at: 0x7fc631f67dd0
  OBJ: TH1D     3     trackL in absorber : 0 at: 0x7fc631f68f70
  KEY: TTree    B4;1 Edep and TrackL
  KEY: TH1D     1;1   Edep in absorber
  KEY: TH1D     2;1   Edep in gap
  KEY: TH1D     3;1   trackL in absorber
  KEY: TH1D     4;1   trackL in gap
root [2] TCanvas* b4 = new TCanvas("b4", "  ");
root [3] TH1D* Edep1 = (TH1D*)_file0.Get("1");
root [4] Edep1->Draw();
```

参考になるexamples

- Examples/basic/B4　サンドウィッチカロリメータ　（先のROOT解析例）
  - ギャップと吸収体でのエネルギー付与
- Examples/basic/B5：２アームスペクトロメータ
  - ワイヤチェンバ、ホドスコープ　での位置測定
  - カロリメータでのエネルギー測定
  - これらをntupleのカラムにベクトルとして記入
- Examples/extended/analysis
  - B4と同様のサンドウィッチカロリメータを材料にして
    - HBOOKへの接続
    - ROOTへの直接の接続
    - AIDA互換