高エネルギー宇宙物理学のための ROOT 入門

– 第 5 回 –

奥村 曉

名古屋大学 宇宙地球環境研究所

2017 年 6 月 1 日

名古屋大学 NAGOYA UNIVERSITY

ISEE Institute for Space-Earth Environmental Research

oxon@mac.com

git pull して最新版にしてください

```
$ cd RHEA
$ git pull

zsh の場合
$ for i in {009..049}; do curl -O https://raw.githubusercontent.com/akira-
okumura/RHEA-Slides/master/photons/lat_photon_weekly_w${i}
_p302_v001_extracted.root; done

bash の場合
$ for i in {10..49}; do curl -O https://raw.githubusercontent.com/akira-okumura/
RHEA-Slides/master/photons/lat_photon_weekly_w0${i}_p302_v001_extracted.root;
done
```

TTree

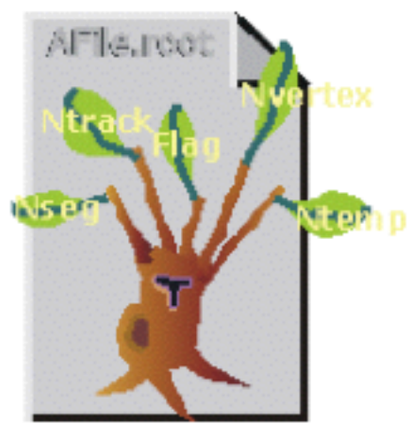
TTree とは

- TH1D や TGraph と違い同じ概念を持つものが他のソフトウェアには（多分）存在しない
- Event の概念を持つ実験データには欠かせない
- 非常に大雑把に説明すると表計算ソフト（Excel など）のシートや FITS の table のようなもの

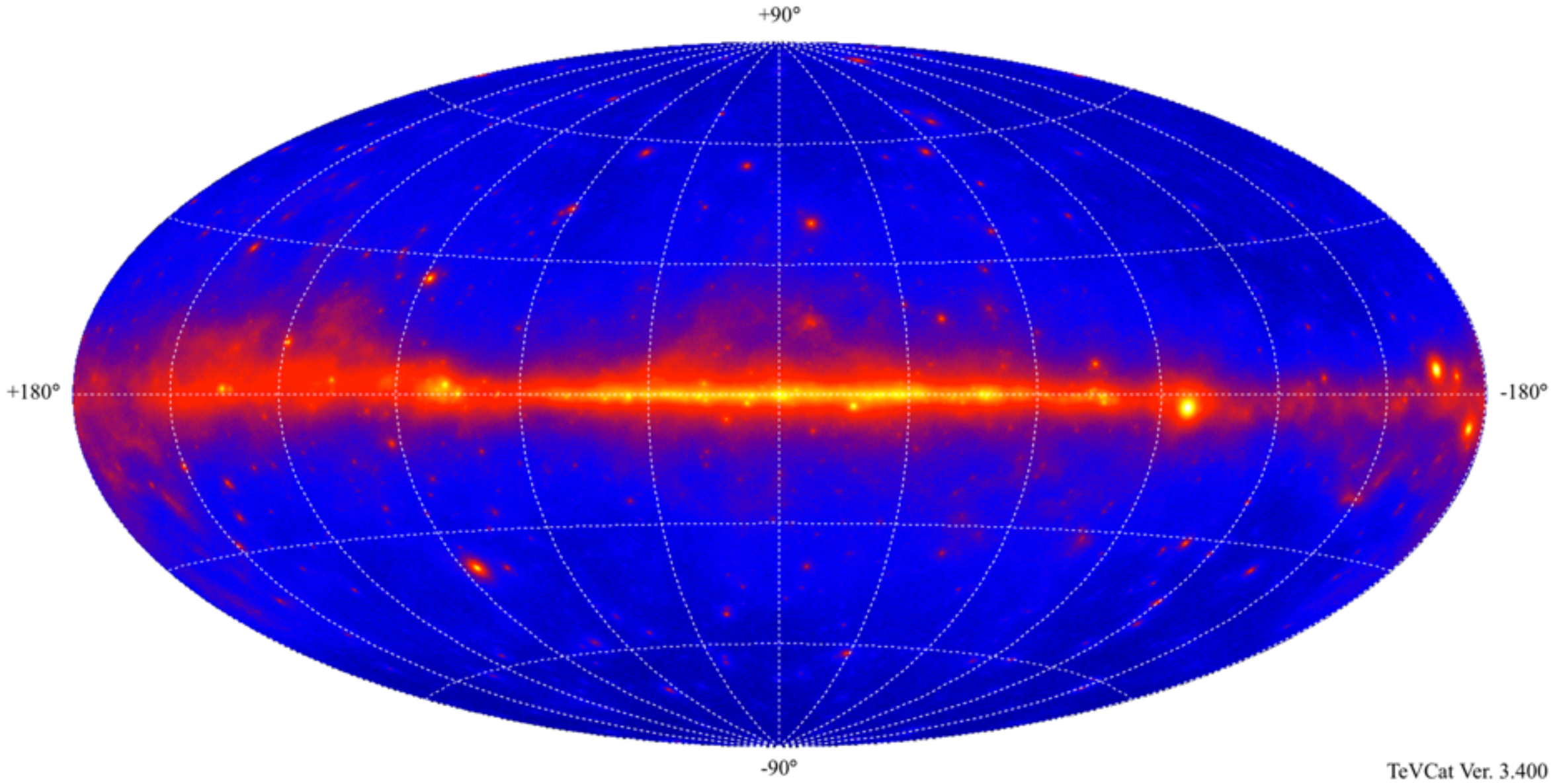
Fermi/LAT のガンマ線イベントデータの例

Event No.	Energy(MeV)	Gal. Longitude (°)	Gal. Latitude (°)	Time (ns)
0	44.5018	123.252	11.7771	239557417.793099
1	241.2553	61.90714	60.7625	239557417.953302
2	105.5841	49.0666	78.24632	239557418.516744
3	248.1058	184.6369	13.48609	239557419.156299

- ただし、演算機能、データの可視化、ROOT クラスの保存など、TTree でしか実現できない機能が多くある
- どうして ROOT を使うのかという問いへの 1 つの答え
- 数字以外にも、class を保存することもできる



Fermi/LAT



- 2008 年に打ち上げられた宇宙線ガンマ線観測用の検出器
- 20 MeV から 300 GeV までを全天でサーベイ観測

まずは遊んでみる（Fermi/LAT のデータ例）

```
$ root misc/lat_photon_weekly_w009_p302_v001.root
root [0]                                          ❶ TTree の含まれる ROOT ファイルを引数にする
Attaching file misc/lat_photon_weekly_w009_p302_v001.root as _file0...
(TFile *) 0x7fc0a2f05aa0                          ❷ TFile が自動的に生成され、ファイルが開かれる
root [1] .ls
TFile**         misc/lat_photon_weekly_w009_p302_v001.root
 TFile*         misc/lat_photon_weekly_w009_p302_v001.root
  KEY: TTree    photons;1 LAT PASS8 Photons       ❸ “photons” という名前の TTree がいる
root [2] photons->Print()
******************************************************************************
*Tree    :photons   : LAT PASS8 Photons                                       *
*Entries :   177778 : Total =        27471504 bytes  File  Size =   27453414 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :ENERGY    : ENERGY[1]/F                                            *
*Entries :   177778 : Total  Size=     713624 bytes  File Size  =     712860 *
*Baskets :       23 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    1 :RA        : RA[1]/F                                                *
*Entries :   177778 : Total  Size=     713516 bytes  File Size  =     712768 *
*Baskets :       23 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
```

❹ “ENERGY” という branch がいる

- 元ファイルは FITS 形式で、わざわざ ROOT に変換して解析する必要はないファイルですが、演習目的です
- 実際のデータで遊べるチュートリアルはそこらへんに落ちてない

続き

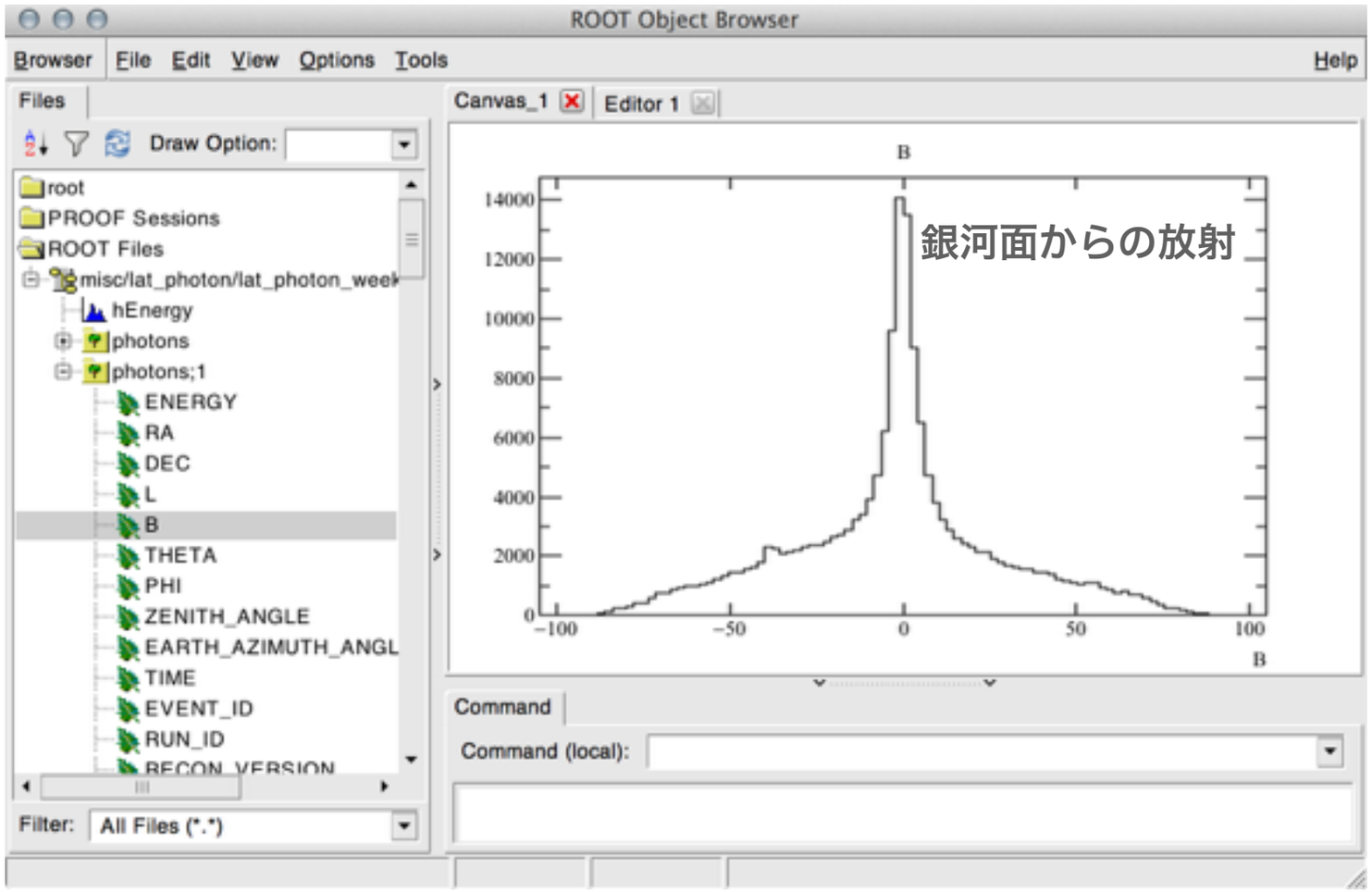
```
root [3] photons->GetEntries()
(Long64_t) 177778
root [4] photons->Show(0)
======> EVENT:0
 ENERGY             = 44.5018
 RA                 = 14.0396
 DEC                = 74.6459
 L                  = 123.252
 B                  = 11.7771
 THETA              = 43.9611
 PHI                = 166.852
 ZENITH_ANGLE       = 70.4655
 EARTH_AZIMUTH_ANGLE = 343.811
 TIME               = 2.39557e+08
 EVENT_ID           = 52785
 RUN_ID             = 239557414
 RECON_VERSION      = 0
```

❶ この TTree には 177,778 イベントが含まれる

❷ ひとつひとつのイベントを見たいとき

- 「イベント」ごとに、色々な情報が入っている
- 数百から数億イベントになってくると、TTree を使う事のありがたみが分かってくる

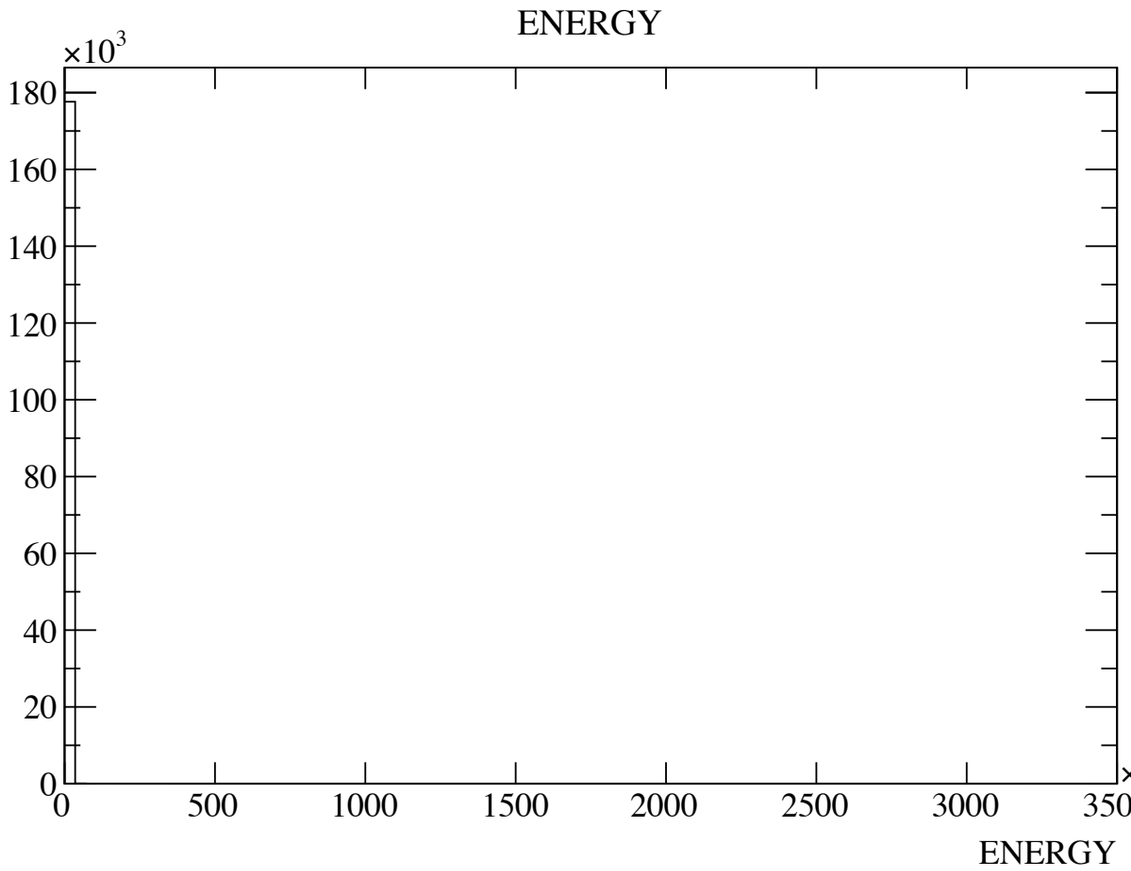
続き

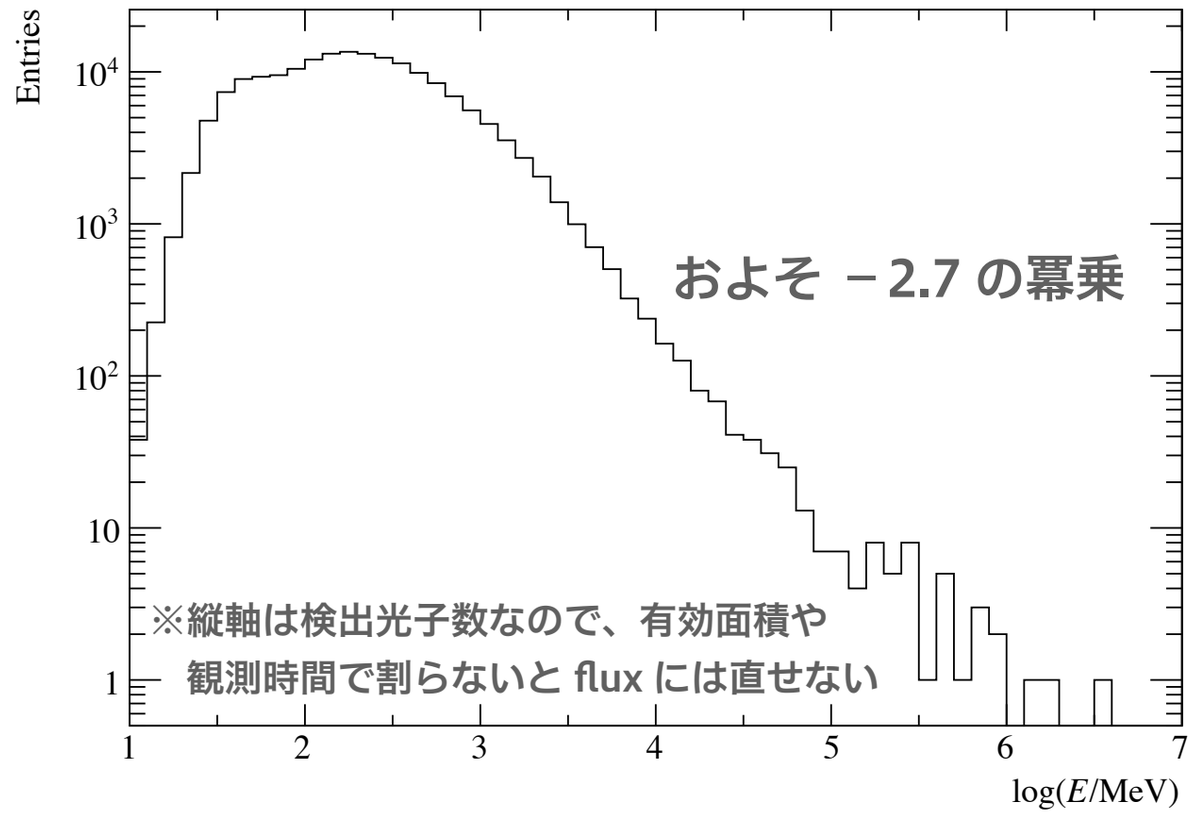



```
root [7] TBrowser b
```

❶ ROOT ファイルを見るためのブラウザが立ち上がる

まずはガンマ線のエネルギー分布を見てみる





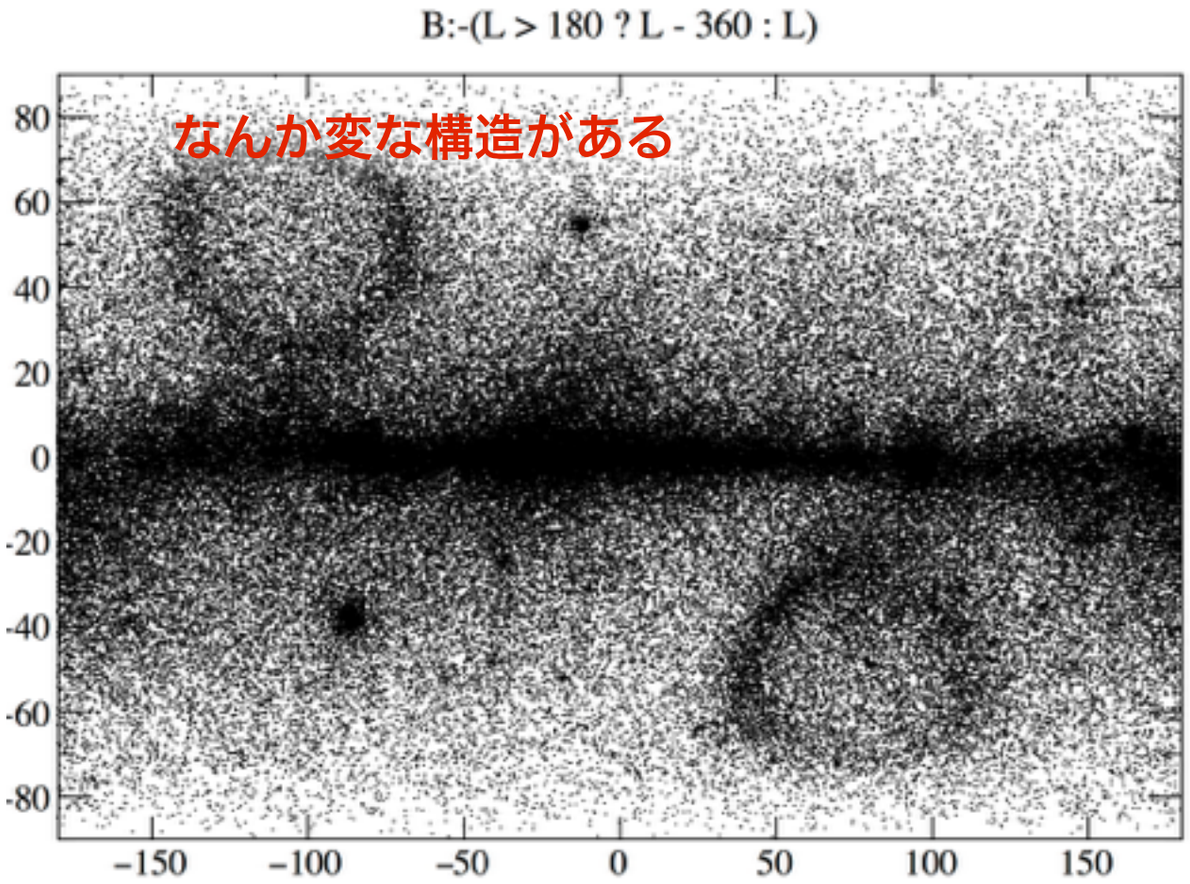

```
root [3] photons->Draw("ENERGY")
```

❶ 好きな branch でヒストグラムを書くことができる
❷ ただし、ビン幅などは勝手に計算される

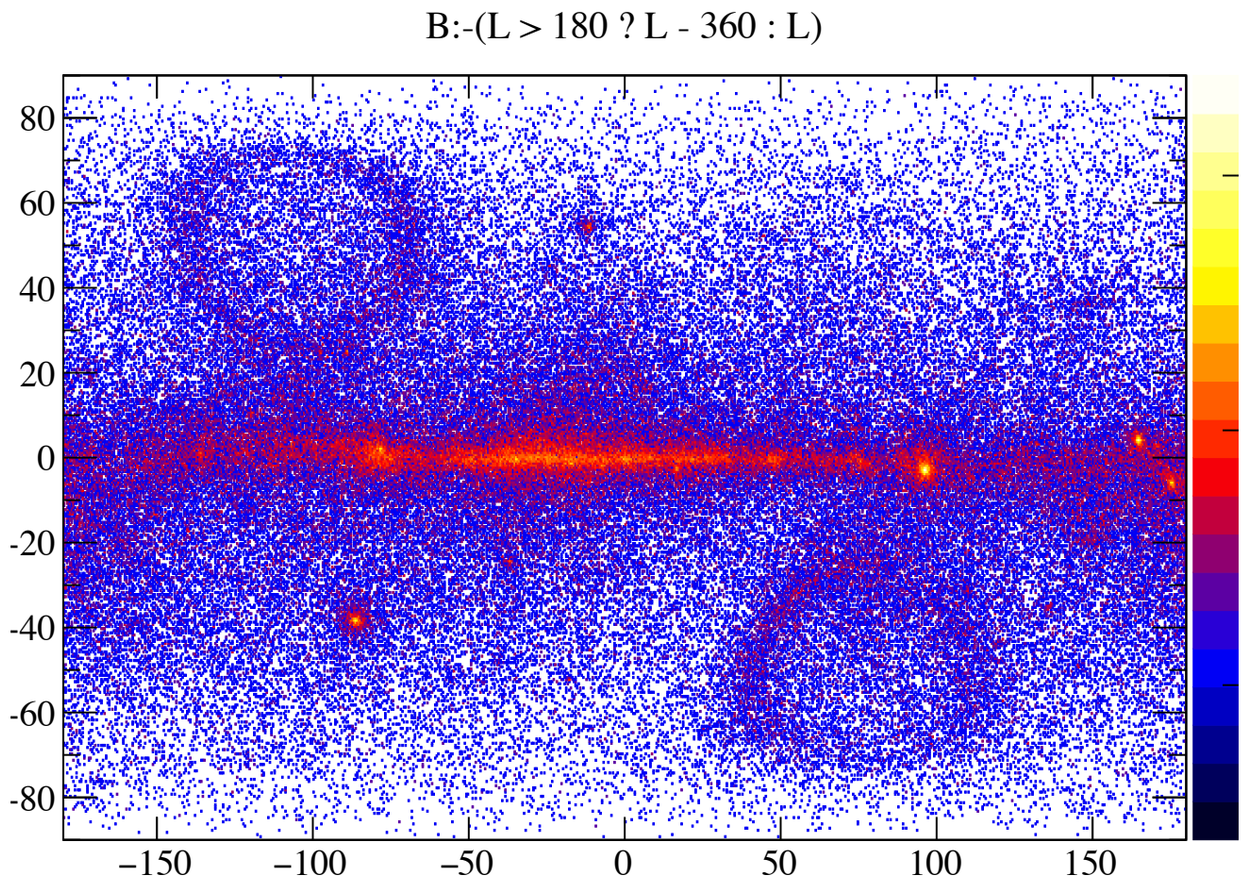
```
root [4] TH1D* hEnergy = new TH1D("hEnergy", ";log(#it{E}/MeV);Entries", 60, 1,
7)
root [5] photons->Draw("log10(ENERGY)>>hEnergy")
root [6] gPad->SetLogy(1)
```

❸ あらかじめ好きなヒストグラムを作っておくと…
❹ そこに TTree::Draw の結果を詰めることができる

銀河座標でガンマ線の分布を見てみる



※ ROOT は TGaxis を使わないと、正から負に向かう軸を書けません



※ ROOT でこのような図を吐くと PDF が非常に重いので注意

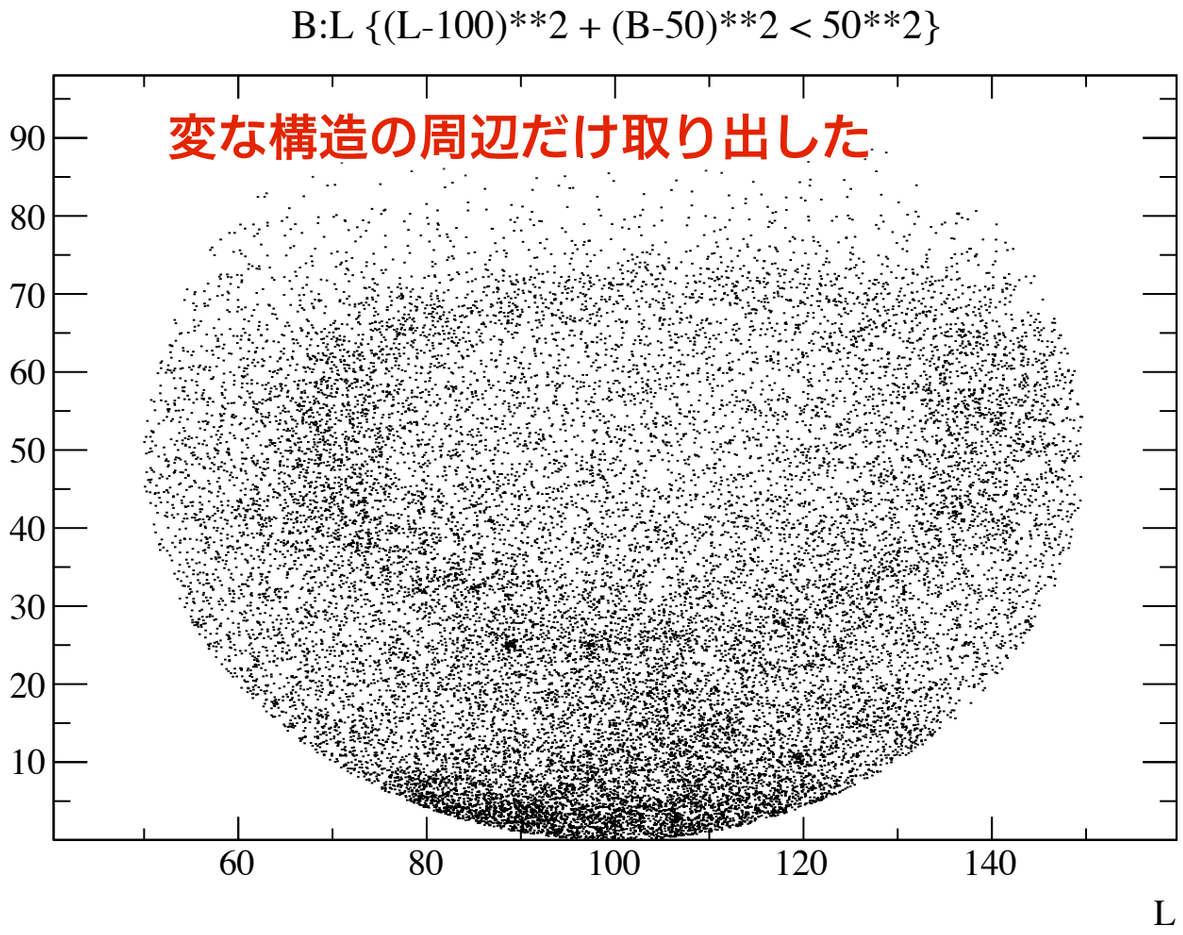
```
root [7] photons->Draw("B:-(L > 180 ? L - 360 : L)>>hGal(720, -180, 180, 360,
-90, 90)")
root [8] hGal
(TH2F *) 0x7fe63ed034c0
root [9] photons->Draw("B:-(L > 180 ? L - 360 : L)>>hGal(720, -180, 180, 360,
-90, 90)", "", "colz")
root [10] gPad->SetLogz()
```

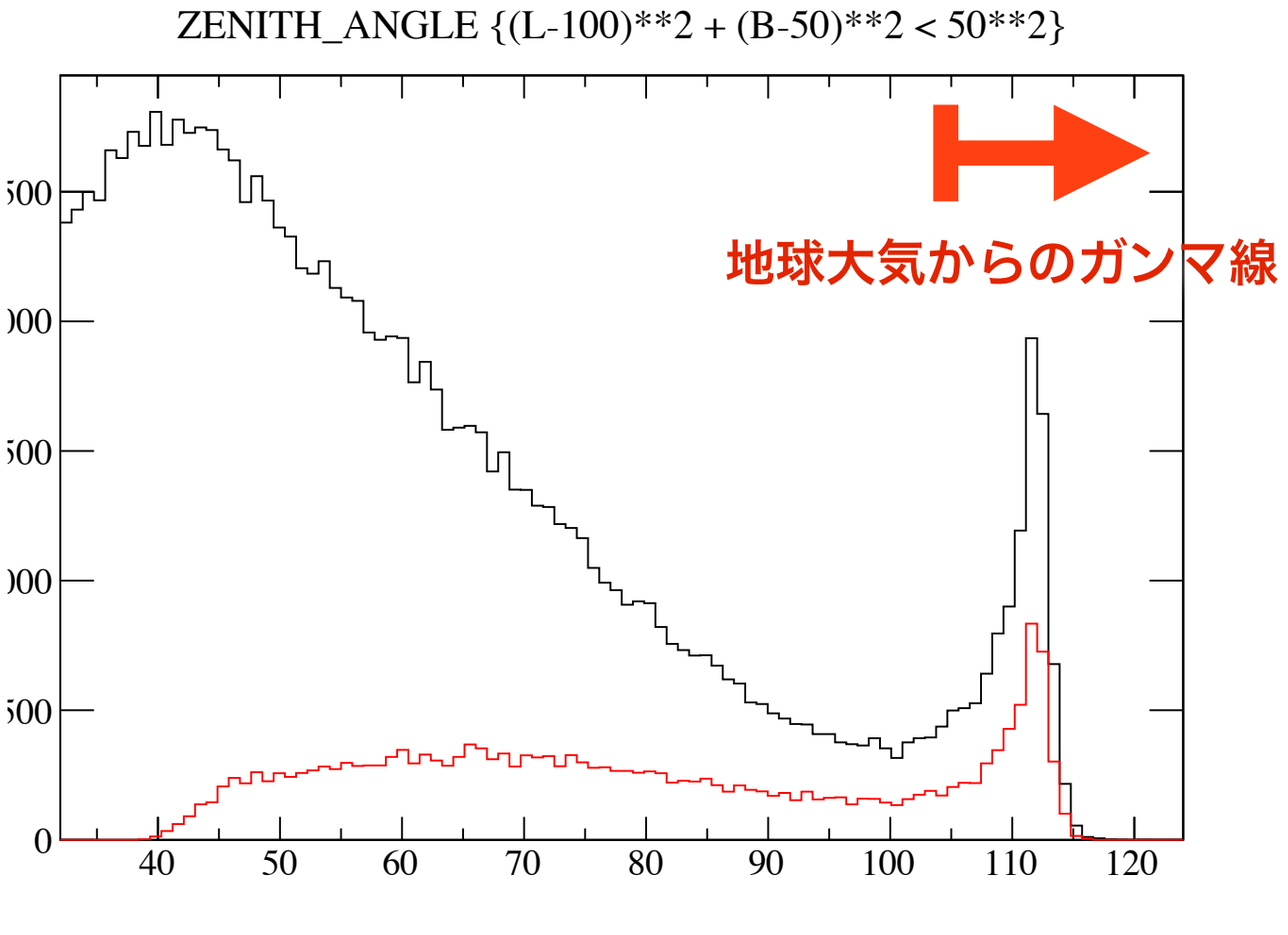
❶ 2 変数使う　❷ 銀河中心を図の真ん中に持ってくる

❸ TH2F（float の 2 次元）が自動で生成された

❹ “colz” をつけて色をつける

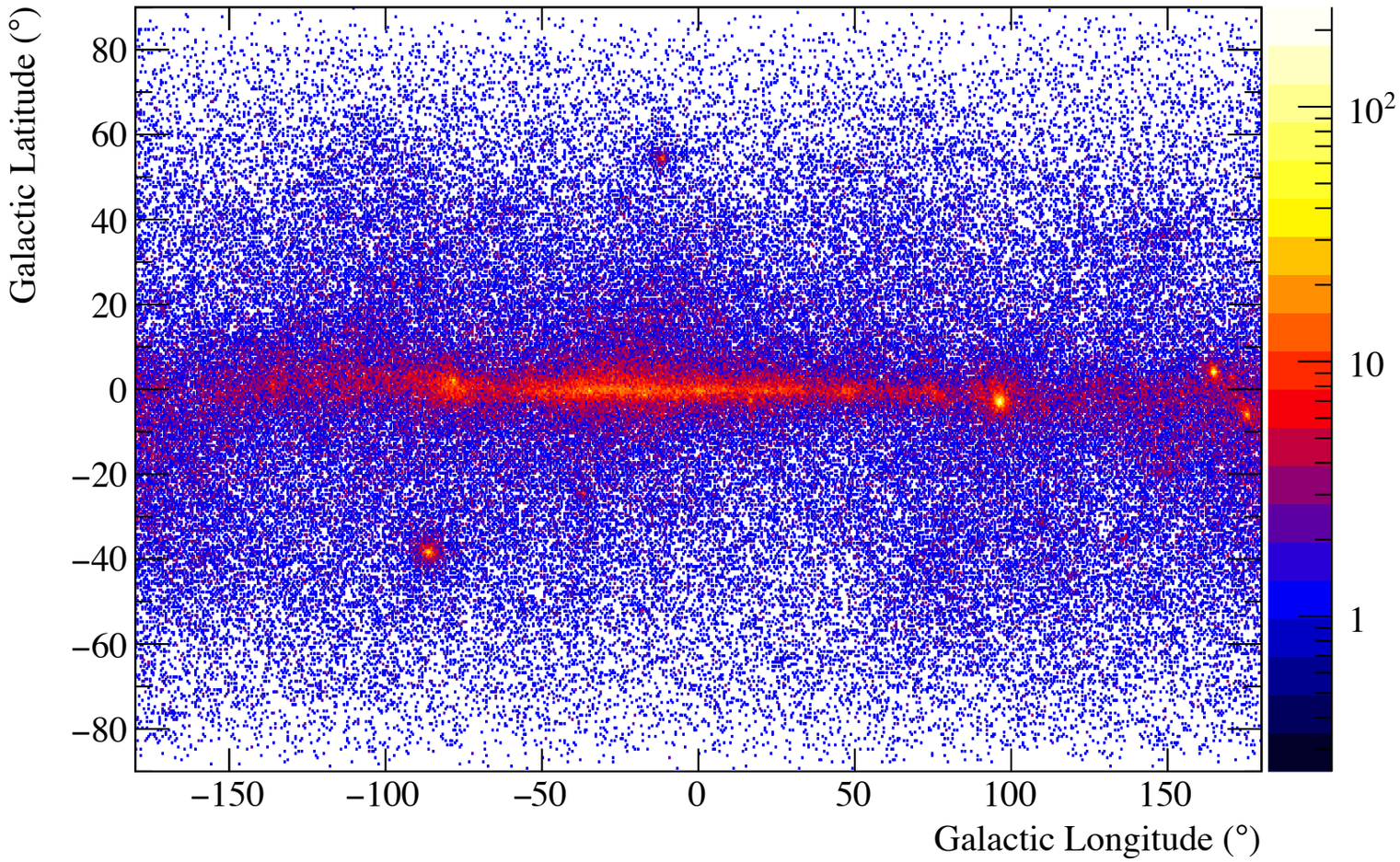
変な構造の原因を探る






```
root [11] photons->Draw("B:L", "(L-100)**2 + (B-50)**2 < 50**2")
                                ❶ 第二引数に条件（cut）を指定する
(Long64_t) 20676                ❷ cut に当てはまったイベントの数が返り値

root [28] photons->Draw("ZENITH_ANGLE>>z1") ❸ 全ガンマ線の天頂角分布
root [29] photons->Draw("ZENITH_ANGLE>>z2", "(L-100)**2 + (B-50)**2 < 50**2",
"same")                                     ❹ 変な構造周辺のガンマ線のみの天頂角分布
root [30] z2->SetLineColor(2)
```

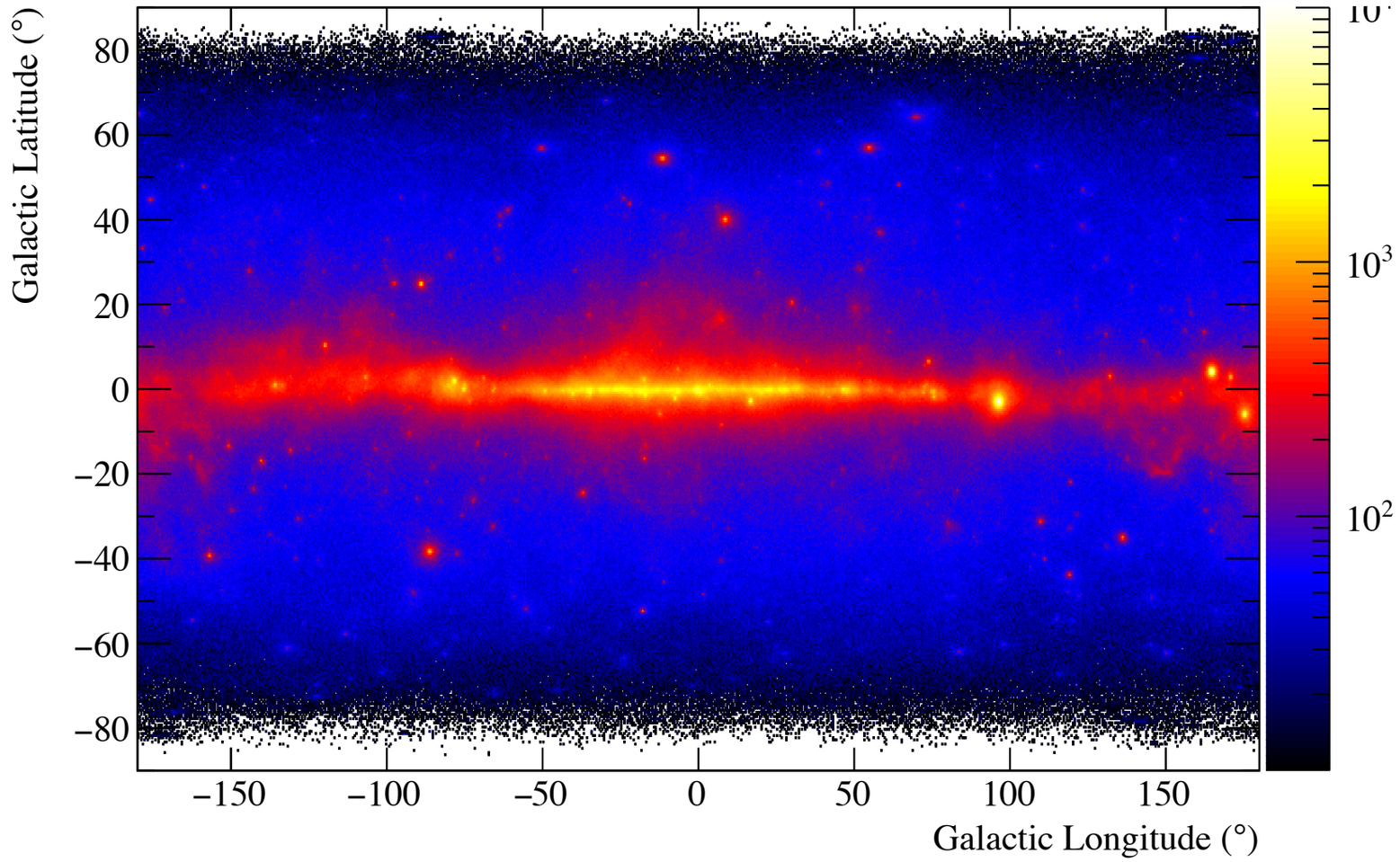
Cut をかけて銀河座標で再度




```
root [6] photons->Draw("B:-(L > 180 ? L - 360 : L)>>hGal", "ZENITH_ANGLE<100",
"colz")
root [7] gPad->SetLogz(1)
```

❶ 天頂角でカットをかけることで必要なデータのみ得られた

複数の ROOT ファイルから TTree を結合する（TChain）




```
root [0] TChain chain("photons")
(TChain &) Name: photons Title:
root [1] for(int i = 9; i < 50; i++) chain.Add(Form("lat_photon_weekly_w
%03d_p302_v001_extracted.root", i));
root [2] photons->Draw("B:L>>hGal(720,-180,180,360,-90,90)", "", "colz")
root [3] hGal->SetContour(100)
root [4] hGal->SetMinimum(10)
root [5] hGal->SetMaximum(1e4)
root [6] hGal->SetTitle(";Galactic Longitude (#circ);Galactic Latitude (#circ)")
root [7] gPad->SetLogz(1)
```

❶ まず同名の TChain を作る

❷ 同名の TTree が含まれる ROOT ファイルを追加する

❸ 後は TTree と同様にできる

もう少し cut の練習（TCut を使う）

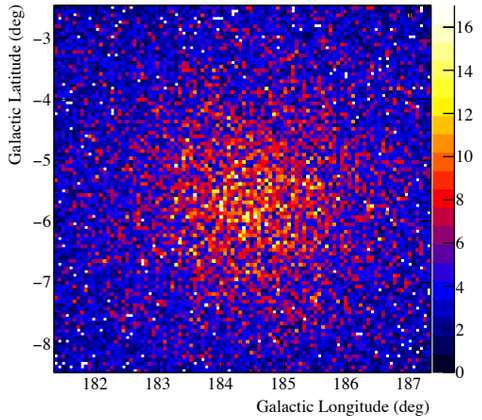
```
void lat_resolution(const char* directory) {
  TChain* chain = new TChain("photons");
  for(int i = 9; i < 50; i++) chain->Add(Form("%s/lat_photon_weekly_w%03d_p302_v001_extracted.root", directory, i));

  TCut cut1("cut1", "ENERGY > 200");
  TCut cut2("cut2", "ENERGY > 1000");

  TCanvas* can = new TCanvas("can", "can", 800, 800);
  can->Divide(2, 2);

  TH2F* hCrab[3];
  TH1D* prox[3];

  for(int i = 0; i < 3; i++) {
    const double kLongitude = 184.33;
    const double kLatitude = -5.47;
    hCrab[i] = new TH2F(Form("hCrab%d", i),
                        ";Galactic Longitude (deg);Galactic Latitude (deg)",
                        100, kLongitude - 3, kLongitude + 3,
                        100, kLatitude - 3, kLatitude + 3);
    can->cd(i + 1);
    if (i == 0) chain->Draw("B:L>>hCrab0", !cut1, "colz");
    else if(i == 1) chain->Draw("B:L>>hCrab1", cut1&&(!cut2), "colz");
    else chain->Draw("B:L>>hCrab2", cut2, "colz");

    prox[i] = hCrab[i]->ProjectionX(Form("pro%d", i));
    prox[i]->SetMinimum(0);
    prox[i]->SetMarkerColor(i + 1);
    prox[i]->SetLineColor(i + 1);

    can->cd(4);
    prox[i]->Draw(i == 0 ? "e" : "e same");
  }
}
```

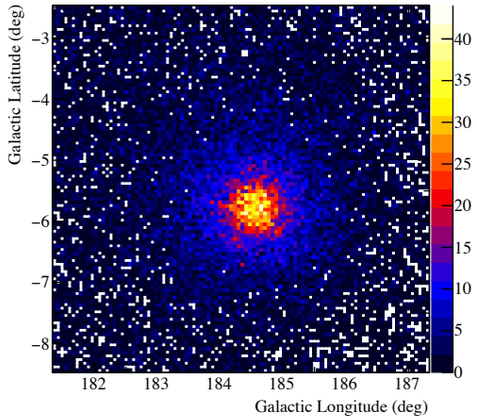
❶ TCut を使ってカットを事前に定義する

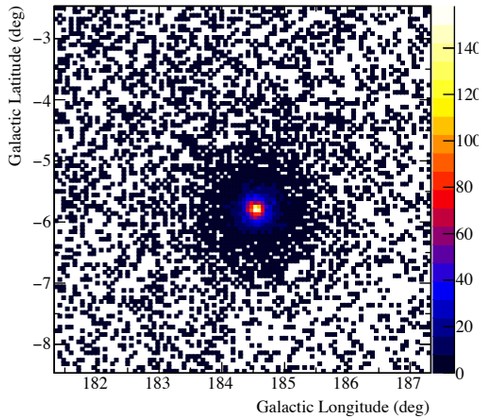
❷ TCut を第二引数に使う

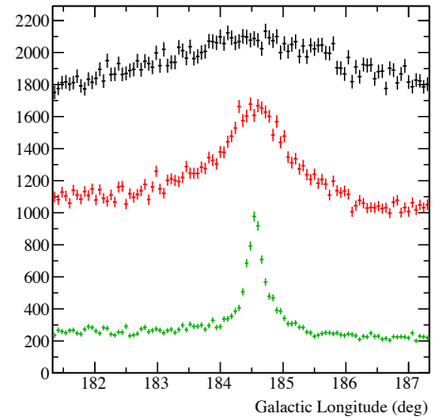
❸ 論理和や論理積も使える

もう少し cut の練習










```
$ root
root [0] .x lat_resolution.C("~/git/RHEA-Slides/photons")
```

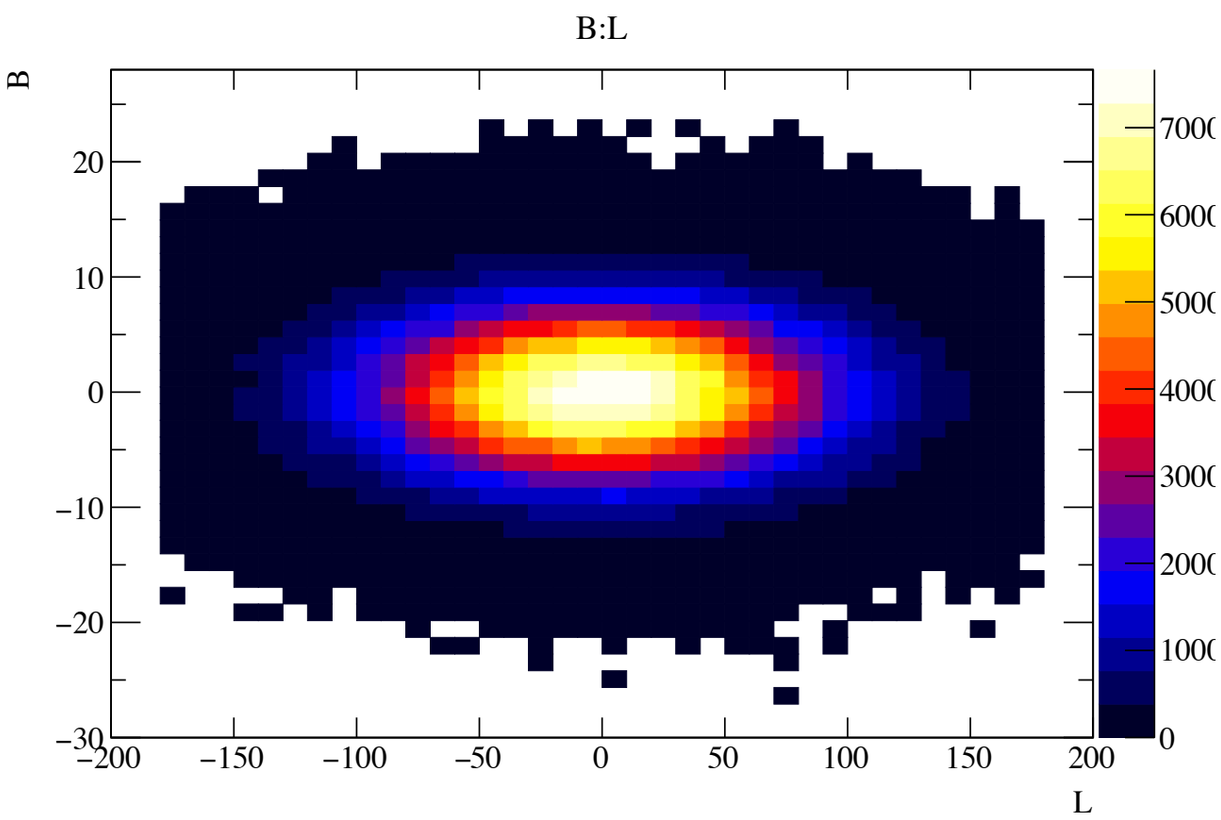
TTree の作り方

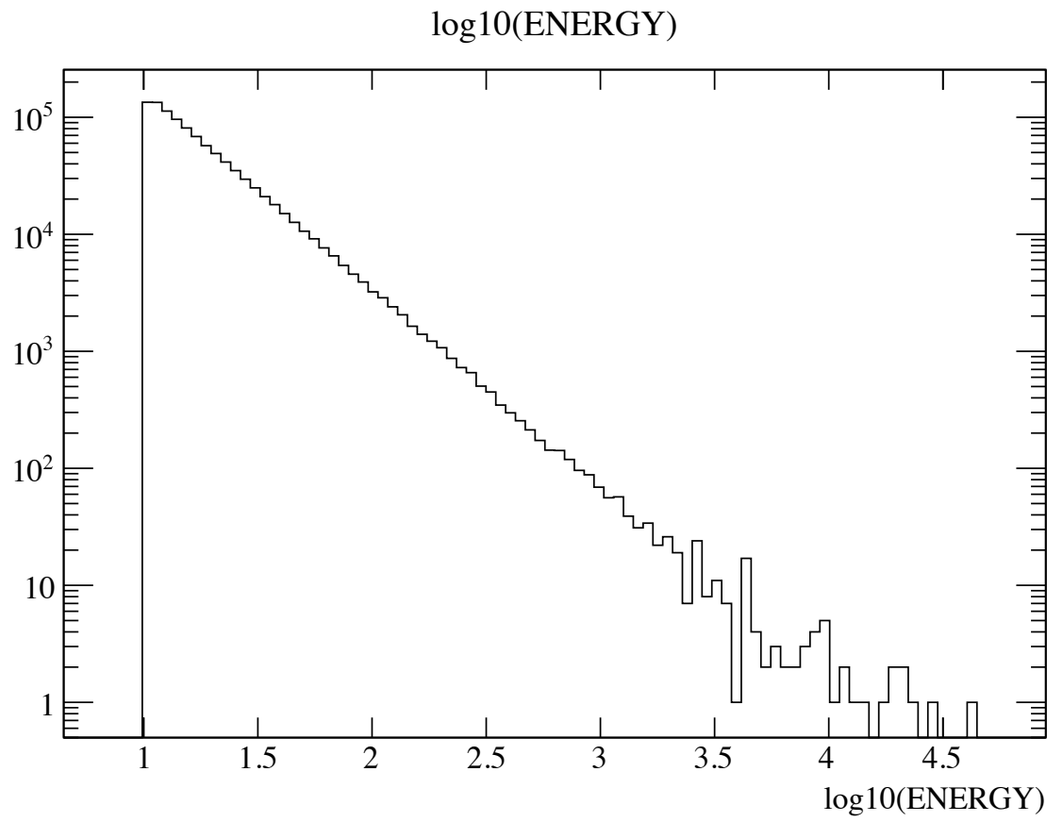
TNtuple

TNtuple とは

- Q. なんで TTree じゃなくて TNtuple からやるの?
  A. そっちのほうが簡単だから
- TNtuple は TTree の派生クラス
- TTree には int でも double でも ROOT のクラスで
  も詰められるが、TNtuple は float しか詰められない
  - やれることが非常に限られる
  - その分、TTree（に近い概念）を理解するのが楽
- 使いどころ
  - 手早く解析したいとき
  - データの型を気にしなくて良く、データ構造が単純なとき

単純な例（Fermi/LAT データのようなもの）






```
root [0] TNtuple nt("nt", "test", "ENERGY:L:B")
root [1] TF1 f1("f1", "x**(-2.7)", 10, 1000000)
root [2] while(nt.GetEntries() < 1000000){
root (cont'ed, cancel with .@) [3] float e = f1.GetRandom();
root (cont'ed, cancel with .@) [4] float l = gRandom->Gaus(0, 60);
root (cont'ed, cancel with .@) [5] float b = gRandom->Gaus(0, 5);
root (cont'ed, cancel with .@) [6] if(abs(l) <= 180 && abs(b) <= 90) nt.Fill(e, l, b);
root (cont'ed, cancel with .@) [7]}
root [8] nt.Draw("B:L", "", "colz")
root [9] nt.Draw("log10(ENERGY)")
root [10] gPad->SetLogy(1)
```

❶ TNtuple 作成。第三引数は float の変数名一覧。

❷ -2.7 乗の冪に従う乱数発生用の一変数関数

❸ 詰めたい変数値を
イベントごとに代入し

❹ Fill することでイベントを増やす

❺ 後は TTree と同様に遊ぶ

TTree の読み書き（1）

```
$ root misc/lat_photon_weekly_w009_p302_v001.root
root [1] photons->Print()
******************************************************************************
*Tree    :photons   : LAT PASS8 Photons                                       *
*Entries :   177778 : Total =        27471504 bytes  File  Size =   27453414 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :ENERGY    : ENERGY[1]/F                                            *
*Entries :   177778 : Total  Size=     713624 bytes  File Size  =     712860 *
*Baskets :       23 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    1 :RA        : RA[1]/F                                                *
*Entries :   177778 : Total  Size=     713516 bytes  File Size  =     712768 *
*Baskets :       23 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    2 :DEC       : DEC[1]/F                                               *
*Entries :   177778 : Total  Size=     713543 bytes  File Size  =     712791 *
*Baskets :       23 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
(省略)
$ curl -O https://raw.githubusercontent.com:443/akira-okumura/RHEA-Slides/master/photons/
lat_photon_weekly_w009_p302_v001_extracted.root
$ root lat_photon_weekly_w009_p302_v001_extracted.root
root [1] photons->Print()
******************************************************************************
*Tree    :photons   :                                                         *
*Entries :   166224 : Total =         2001714 bytes  File  Size =    1830019 *
*        :          : Tree compression factor =   1.09                       *
******************************************************************************
*Br    0 :ENERGY    : ENERGY/F                                               *
*Entries :   166224 : Total  Size=     667210 bytes  File Size  =     608569 *
*Baskets :       21 : Basket Size=      32000 bytes  Compression=   1.10     *
*.............................................................................*
*Br    1 :L         : L/F                                                    *
*Entries :   166224 : Total  Size=     667085 bytes  File Size  =     594905 *
*Baskets :       21 : Basket Size=      32000 bytes  Compression=   1.12     *
*.............................................................................*
*Br    2 :B         : B/F                                                    *
*Entries :   166224 : Total  Size=     667085 bytes  File Size  =     625487 *
*Baskets :       21 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
```

❶ 前回の LAT データを TTree にしたもの

❷ Branch が合計 23 個ある

❸ TChain を試すときに使った ROOT ファイル

❹ Branch が合計 3 個

TTree の読み書き（2）

```
$ cat src/tree_extract.C
void tree_extract(const char* input, const char* output) {
  TFile fin(input);
  TTree* photons = (TTree*)fin.Get("photons");


  Float_t energy, l, b, zenith;
  photons->SetBranchAddress("ENERGY", &energy);
  photons->SetBranchAddress("L", &l);
  photons->SetBranchAddress("B", &b);
  photons->SetBranchAddress("ZENITH_ANGLE", &zenith);

  TFile fout(output, "create");
  TTree photons_mod("photons", "");
  photons_mod.Branch("ENERGY", &energy, "ENERGY/F");
  photons_mod.Branch("L", &l, "L/F");
  photons_mod.Branch("B", &b, "B/F");

  for(int i = 0; i < photons->GetEntries(); ++i) {
    photons->GetEntry(i);
    if (zenith < 100.) {
      photons_mod.Fill();
    }
  }

  photons_mod.Write();
  fout.Close();
}
```

❶ 前回の LAT データで一部のみを抜き出したスクリプト

❷ TFile::Get を使って、名前で TTree を取り出す
※ TTree 以外も同様に取り出せる
※ キャスト（cast）という作業をする必要がある

❸ イベントごとにブランチの値を読むには、
適切な型の変数を用意しブランチに紐付ける
※ 必ず変数のポインタを渡すこと

❺ TTree::Branch を呼ぶことで、新しく作った
TTree にブランチを追加することができる
※ ここもポインタを渡す

❹ TTree::GetEntry を実行すると、指定した
ブランチのイベント毎の値が変数に代入される

❻ Fill することで、ブランチに使用している変数の
「現在」の値が詰められる

※ GetEntry する度に energy/l/b/zenith は全て
書き変わっている

型に注意

C type	ROOT typedef	C99/C++11	ROOT TTree	Python array	NumPy	FITS
signed char	Char_t	int8_t	B	b	int8	A or S
unsigned char	UChar_t	uint8_t	b	B	uint8	B
signed short	Short_t	int16_t	S	h or i	int16	I
unsigned short	UShort_t	uint16_t	s	H or I	uint16	U
signed int (32 bit)	Int_t	int32_t	I	l	int32	J
unsigned int (32 bit)	UInt_t	uint32_t	i	L	uint32	V
signed int (64 bit)	Long64_t	int64_t	L	N/A	int64	K
unsigned int (64 bit)	ULong64_t	uint64_t	l	N/A	uint64	N/A
float	Float_t	float	F	f	float32	E
double	Double_t	double	D	d	float64	D
bool	Bool_t	bool	O	N/A	bool_	X

- TTree::Branch を呼ぶときは第 3 引数で型を ROOT に教える必要がある
- C++ はポインタでメモリのアドレスが渡されるだけだと、その型を保

Python の場合（やり方はいくつかあります）

```
$ cat src/tree_extract.py
#!/usr/bin/env python
import ROOT
import numpy

def tree_extract(input_name, output_name):
    fin = ROOT.TFile(input_name)
    photons = fin.Get('photons')

    energy = numpy.ndarray(1, dtype = 'float32')
    l = numpy.ndarray(1, dtype = 'float32')
    b = numpy.ndarray(1, dtype = 'float32')

    fout = ROOT.TFile(output_name, 'create')
    photons_mod = ROOT.TTree('photons', '')
    photons_mod.Branch('ENERGY', energy, 'ENERGY/F')
    photons_mod.Branch('L', l, 'L/F')
    photons_mod.Branch('B', b, 'B/F')

    for i in xrange(photons.GetEntries()):
        photons.GetEntry(i)
        energy[0] = photons.ENERGY
        l[0] = photons.L
        b[0] = photons.B
        zenith = photons.ZENITH_ANGLE
        if zenith < 100.:
            photons_mod.Fill()

    photons_mod.Write()
    fout.Close()
```

❶ tree_extract.C を Python にしたもの

❷ numpy を使うやり方にします

❸ Python だと面倒な cast が不要
※ 些細なことだが慣れると C++ に戻れなくなる

❹ Python 上では直接的に C のポインタを渡せないので
numpy の ndarray を使う

❺ ここは C++ と同様、ただし引数は numpy.ndarray
※ PyROOT がうまいこと変換してくれる

❻ TTree::SetBranchAddress 不要
直接ブランチを触れる

クラスを詰める – より ROOT らしい例

```
$ root
root [0] .x event_class_tree.C+("../misc/lat_photon_weekly_w009_p302_v001.root",
"event.root")
Info in <TMacOSXSystem::ACLiC>: creating shared library /Users/oxon/git/RHEA/
src/./event_class_tree_C.so
root [1] TFile f("event.root")
(TFile &) Name: event.root Title:
root [3] photons->Print()
******************************************************************************
*Tree    :photons   :                                                        *
*Entries :   177778 : Total =         6446728 bytes  File  Size =    3336680 *
*       :          : Tree compression factor =   1.93                       *
******************************************************************************
*Br    0 :event     : PhotonEvent                                            *
*Entries :   177778 : Total  Size=    6446347 bytes  File Size  =    3332478 *
*Baskets :      447 : Basket Size=      16000 bytes  Compression=   1.93     *
*.............................................................................*
root [4] photons->Draw("event.fEnergy")
root [6] photons->Draw("event.fB:-(event.fL > 180 ? event.fL - 360 :
event.fL)>>hGal", "", "colz")
(Long64_t) 177778
```

クラスの詰め方

```
#include "TTree.h"
#include "TFile.h"

class PhotonEvent : public TObject {
private:
  Float_t fEnergy;
  Float_t fL;
  Float_t fB;
  Float_t fZenithAngle;
  Short_t fCalibVersion[3];

public:
  void SetEnergy(Float_t energy) {fEnergy = energy;}
  void SetL(Float_t l) {fL = l;}
  void SetB(Float_t b) {fB = b;}
  void SetZenithAngle(Float_t zenith) {fZenithAngle = zenith;}
  void SetCalibVersion(Short_t* calib) {
    for(int i = 0; i < 3; ++i) {
      fCalibVersion[i] = calib[i];
    }
  }

  ClassDef(PhotonEvent, 1)
};

void event_class_tree(const char* input, const char* output) {
  TFile fin(input);
  TTree* photons = (TTree*)fin.Get("photons");
  Float_t energy, l, b, zenith;
  Short_t calib[3];
  photons->SetBranchAddress("ENERGY", &energy);
  photons->SetBranchAddress("L", &l);
  photons->SetBranchAddress("B", &b);
  photons->SetBranchAddress("ZENITH_ANGLE", &zenith);
  photons->SetBranchAddress("CALIB_VERSION", calib);

  PhotonEvent event;

  TFile fout(output, "create");
  TTree photons_mod("photons", "");
  photons_mod.Branch("event", "PhotonEvent", &event, 16000, 0);

  for(int i = 0; i < photons->GetEntries(); ++i) {
    photons->GetEntry(i);
    event.SetEnergy(energy);
    event.SetL(l);
    event.SetB(b);
    event.SetZenithAngle(zenith);
    event.SetCalibVersion(calib);
    photons_mod.Fill();
  }

  photons_mod.Write();
  fout.Close();
}
```

❶ コンパイルするので必要なものを #include

❷ クラスを作る。TObject から継承しなくても良い。

❸ メンバ変数の型は、メモリサイズの環境依存を
減らすために ROOT で typedef されたものを使う

❹ メンバ変数を private にする場合は setter を

❺ ROOT で class を追加するときのおまじない

❻ クラスのインスタンスのポインタを渡す

❼ クラスのメンバ変数を更新して詰めるだけ