

高エネルギー宇宙物理学 のための ROOT 入門

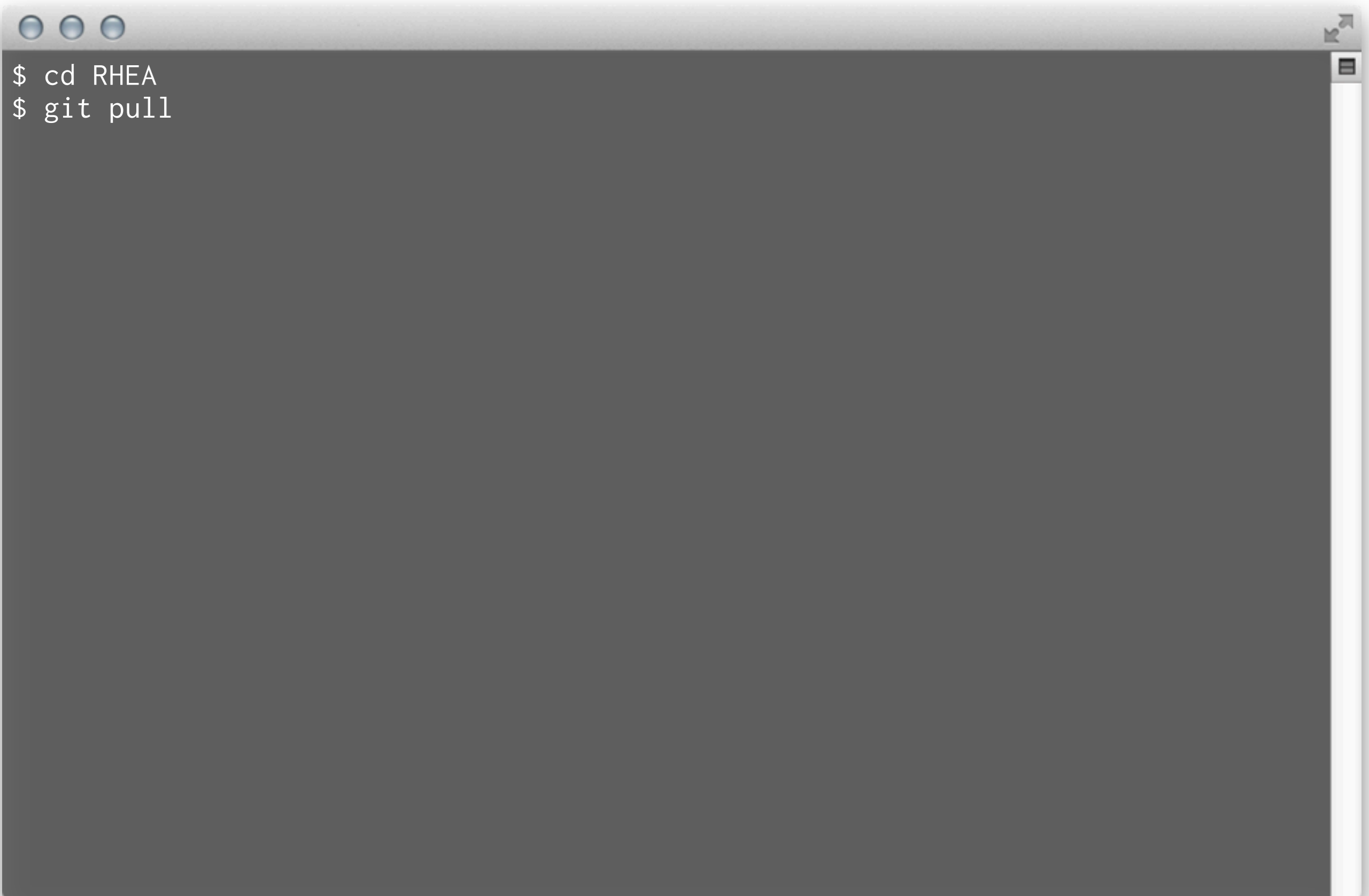
– 第 3 回 –

奥村 暁

名古屋大学 宇宙地球環境研究所

2016 年 5 月 25 日

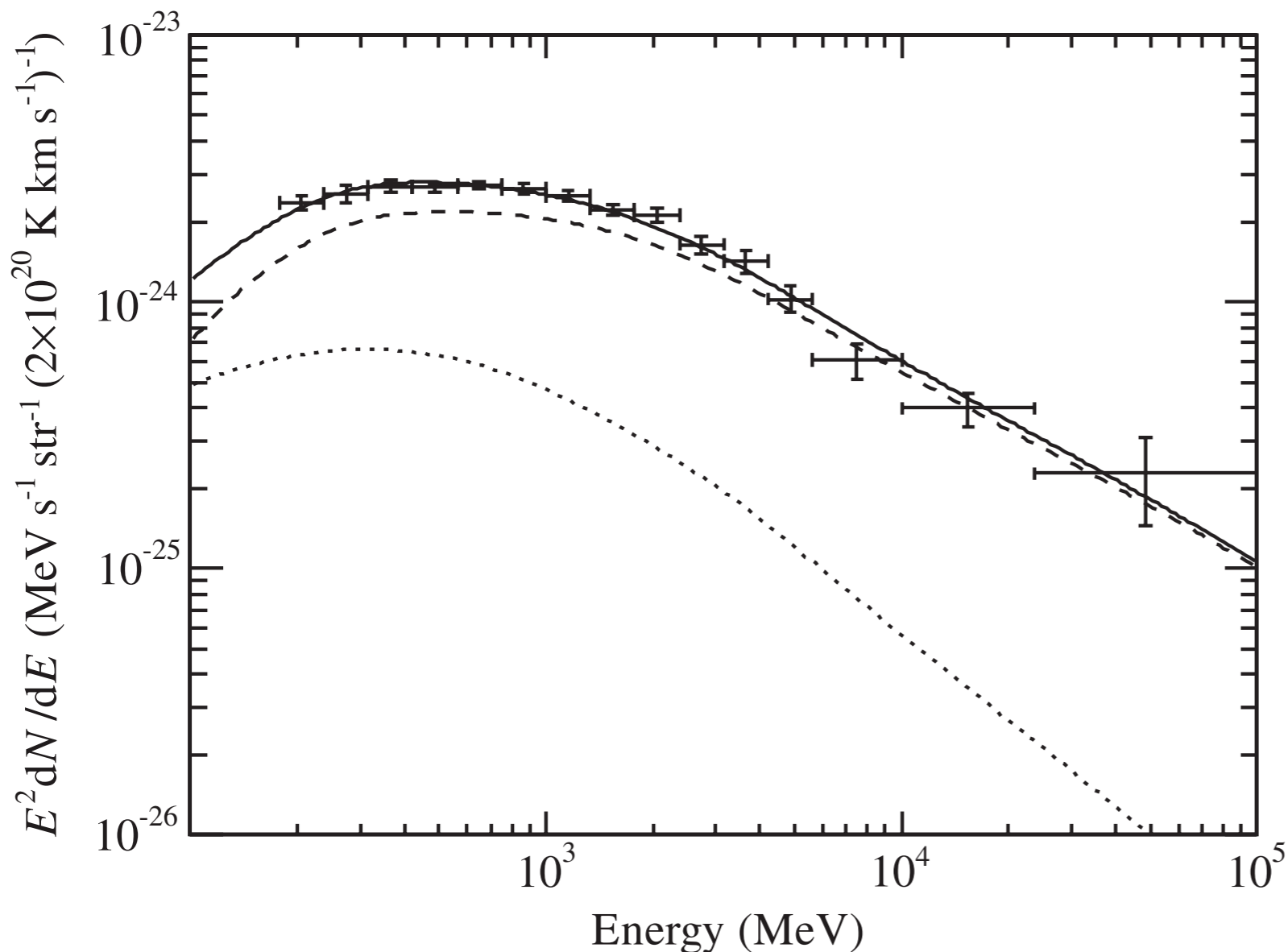
最新版に git pull してください

A terminal window with a dark gray background and light gray text. The window has three circular window control buttons (red, yellow, green) in the top-left corner and a standard macOS window title bar with a maximize button in the top-right corner. The terminal content shows two lines of text: "\$ cd RHEA" followed by "\$ git pull".

```
$ cd RHEA
$ git pull
```

グラフ

グラフ (graph) とは何か？



Ackermann et al. (2012)

- 得られたデータの変数を図表化したもの
- 狭義には2つ以上の変数の関係を示すために軸とともにデータ点を表示したもの
- 実験での使用例
 - ▶ 光検出器の印加電圧と利得の関係
 - ▶ エネルギースペクトル (energy spectrum)

大事なこと

- (2次元の場合) 独立変数 x と従属変数 y の違いを意識する
 - ▶ 例えば光検出器の利得 (従属変数) は、印加電圧 (独立変数) を変化させることで変化する
 - ▶ 滅多に見かけないが、これらを入れ替えて作図しない
- 無闇にデータ点を線で結ばない
 - ▶ 測定値には誤差がつきものなので、折れ線グラフはデータ解釈に先入観を持たせる
- 誤差棒の付け方 (第2回資料参照)
- エネルギースペクトルの横軸誤差棒はビン幅の場合あり

ROOT のクラス

- TGraph

- ▶ 2次元のグラフ
- ▶ 誤差棒無し

- TGraphErrors

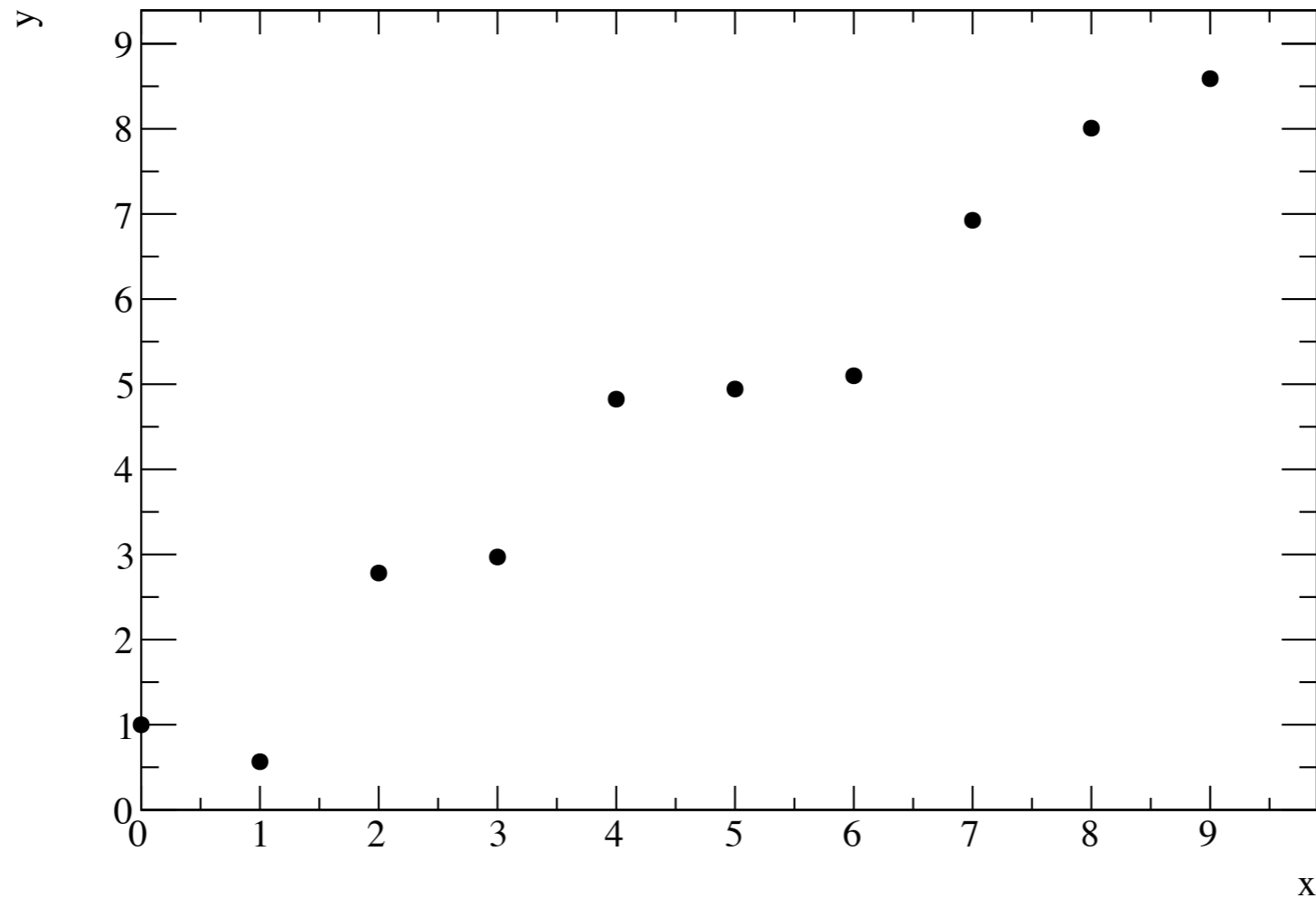
- ▶ 誤差棒あり

- TGraph2D と TGaph2DErrors

- ▶ それぞれ 3次元版
- ▶ 名前が紛らわしいが、x/y/z の 3つの値を持つ

2次元グラフ

単純な例

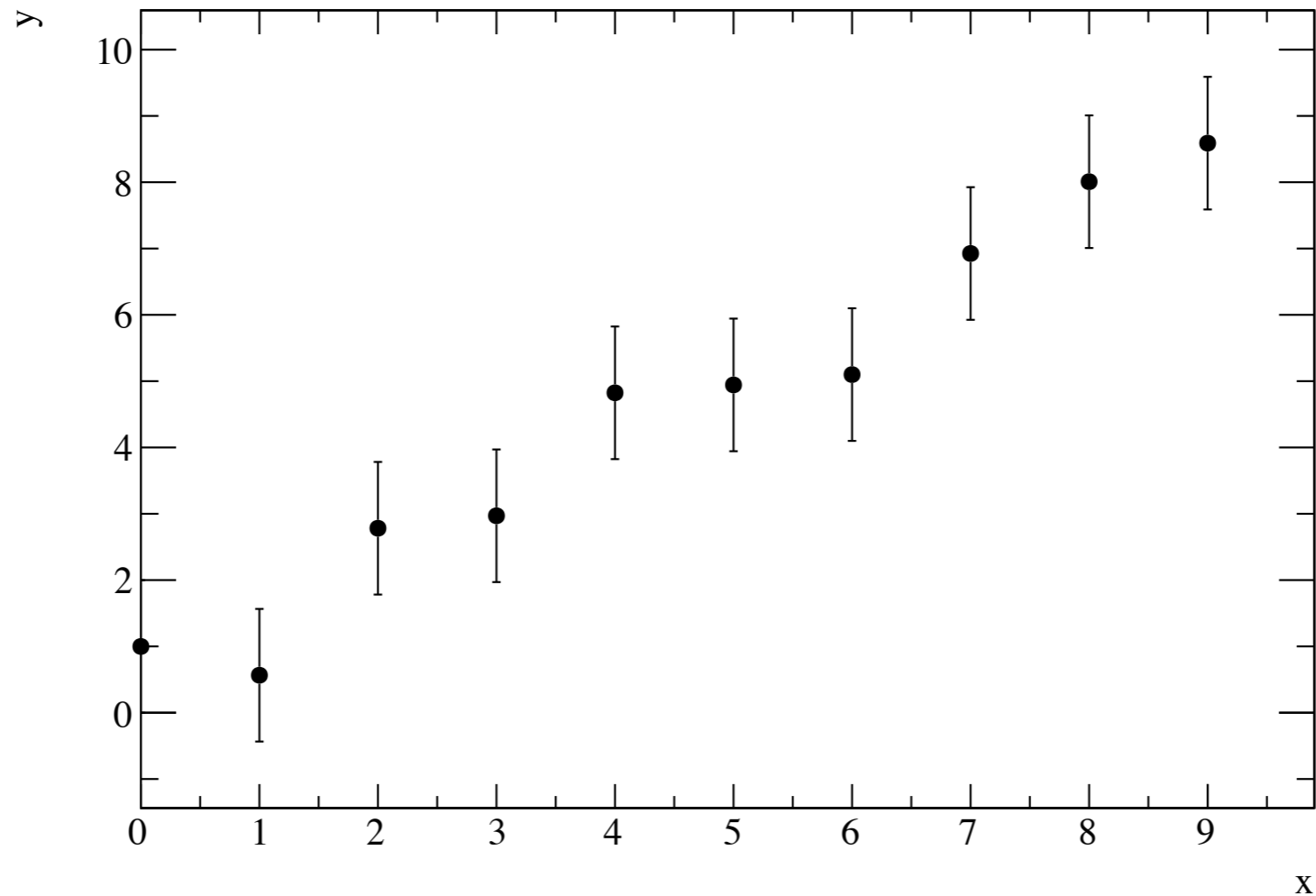


```
$ root
root [0] TGraph* graph = new TGraph;
root [1] for (int i = 0; i < 10; ++i) {
root (cont'ed, cancel with .@) [2] double x = i;
root (cont'ed, cancel with .@) [3] double y = i + gRandom->Gaus();
root (cont'ed, cancel with .@) [4] graph->SetPoint(i, x, y);
root (cont'ed, cancel with .@) [5]}
root [6] graph->SetTitle(";x;y;")
root [7] graph->SetMarkerStyle(20)
root [8] graph->Draw("ap")
```

- ① 適当に値を作り
- ② 点を追加する

- ③ タイトルはコンストラクタ外で
- ④ 初期値はドットなので変更する
- ⑤ axis と point を描く

誤差棒を足す



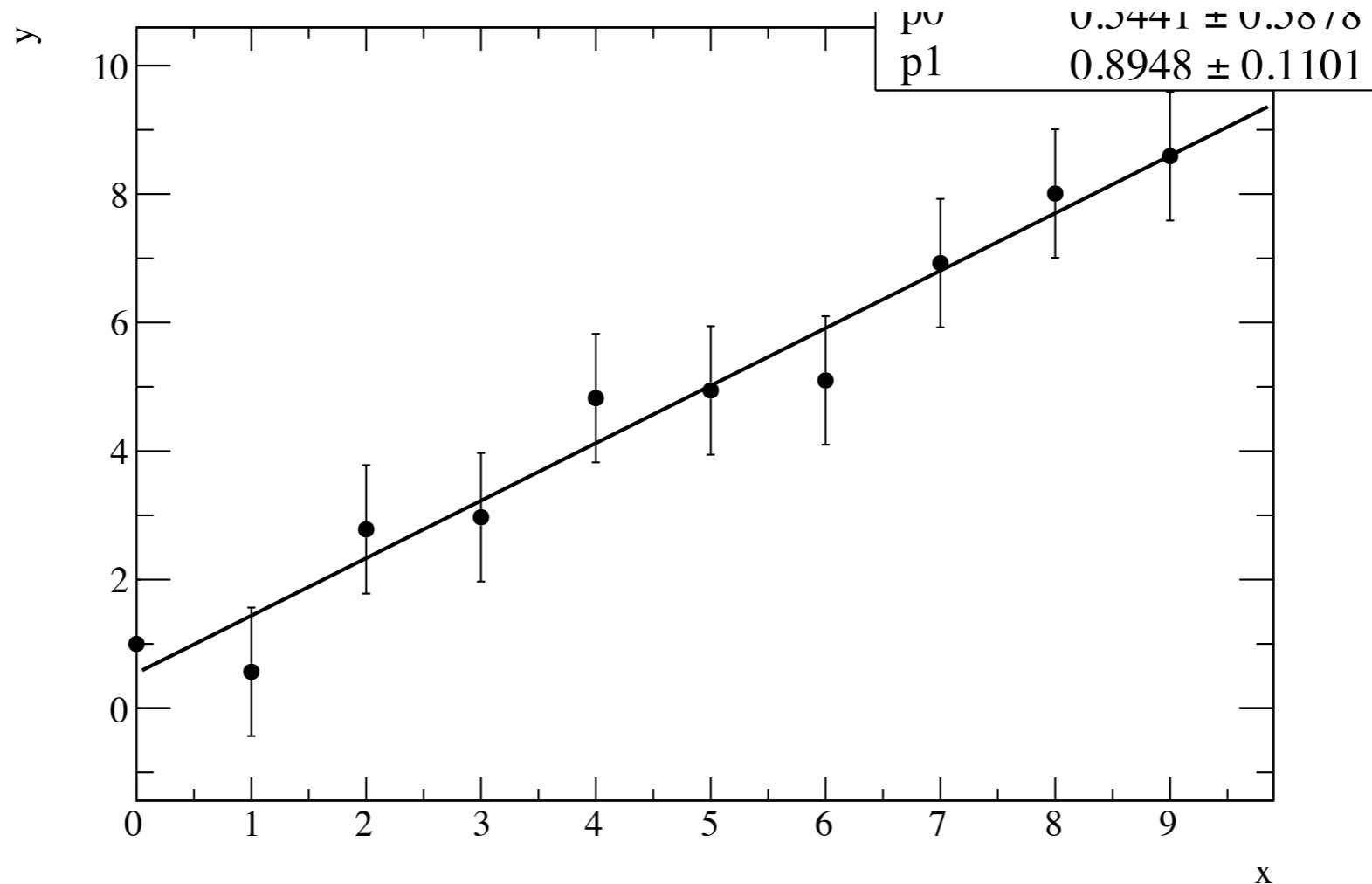
```
root [0] TGraphErrors* graph = new TGraphErrors
root [1] for (int i = 0; i < 10; ++i) {
root (cont'ed, cancel with .@) [2] double x = i;
root (cont'ed, cancel with .@) [3] double y = i + gRandom->Gaus();
root (cont'ed, cancel with .@) [4] double ex = 0;
root (cont'ed, cancel with .@) [5] double ey = 1.;
root (cont'ed, cancel with .@) [6] graph->SetPoint(i, x, y);
root (cont'ed, cancel with .@) [7] graph->SetPointError(i, ex, ey);
root (cont'ed, cancel with .@) [8]}
root [9] graph->SetTitle(";x;y;")
root [10] graph->SetMarkerStyle(20)
root [11] graph->Draw("ap")
```

① TGraphErrors にする

② y のばらつきと同じ量

③ 誤差を追加

既存の関数でのフィット

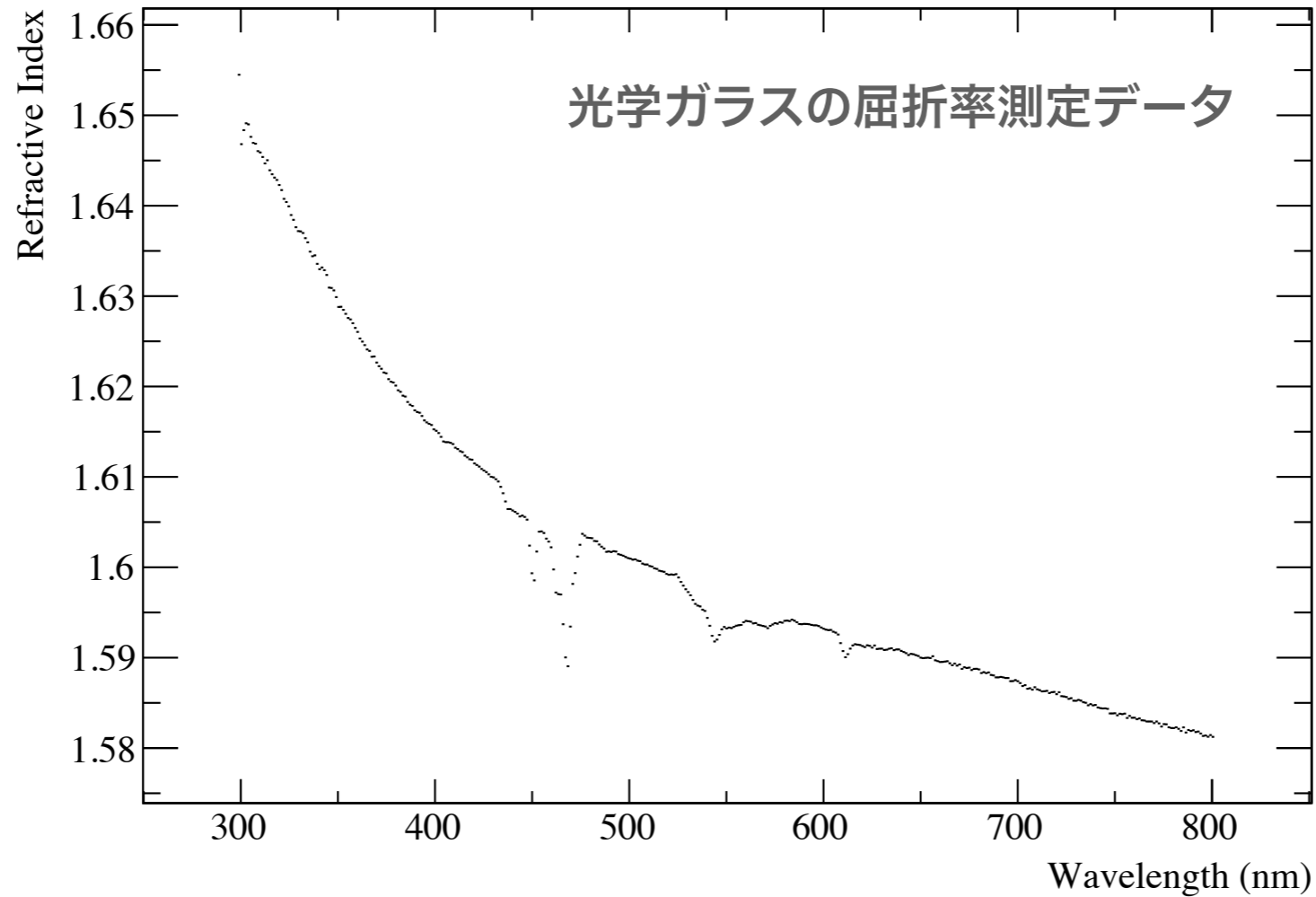


```
root [13] gStyle->SetOptFit()
root [14] graph->Fit("pol1")
*****
Minimizer is Linear
Chi2          =      2.50415
Ndf           =           8
p0            =      0.544079  +/-   0.587754
p1            =      0.894761  +/-   0.110096
(TFitResultPtr) @0x7fb24d517d50
root [15] TMath::Prob(2.504, 8)
(Double_t) 0.961544
root [22] graph->GetFunction("pol1")->GetProb()
(Double_t) 0.961537
```

① 1次関数 (pol1) でフィット
 $f(x) = p_1 x + p_0$

② χ^2 フィットの確率を確認

ファイルの読み込み



```
$ head -n 2 src/UVC-200B.csv
```

```
299.78,1.65449,.0363084
```

```
300.99,1.64681,.1093
```

```
$ root
```

```
root [0] TGraph* graph = new TGraph("src/UVC-200B.csv", "%lg,%lg,%*lg")
```

① ファイル名

② フォーマット指定

```
root [1] graph->SetTitle(";Wavelength (nm);Refractive Index;")
```

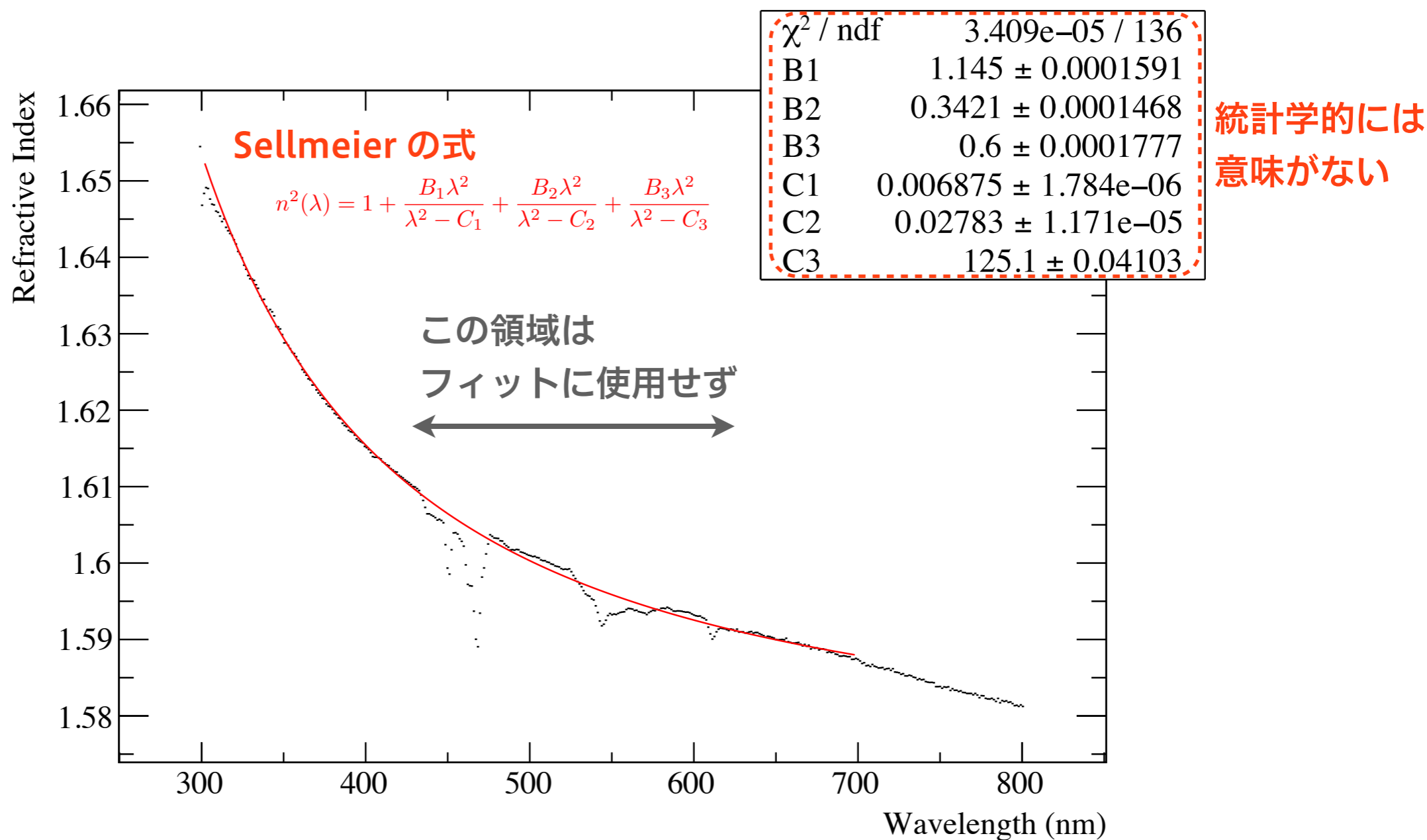
```
root [2] graph->Draw("ap")
```

ついでに好きな関数形でフィットしてみる

```
$ cat Sellmeier.C
(略)
Double_t SellmeierFormula(Double_t* x, Double_t* par) { ① フィット用関数の定義
(略)
    Double_t lambda2 = TMath::Power(x[0] / 1000., 2.); ② 変数 x[] とパラメータ par[] から計算
    return TMath::Sqrt(1 + par[0] * lambda2 / (lambda2 - par[3]) +
        par[1] * lambda2 / (lambda2 - par[4]) +
        par[2] * lambda2 / (lambda2 - par[5]));
}

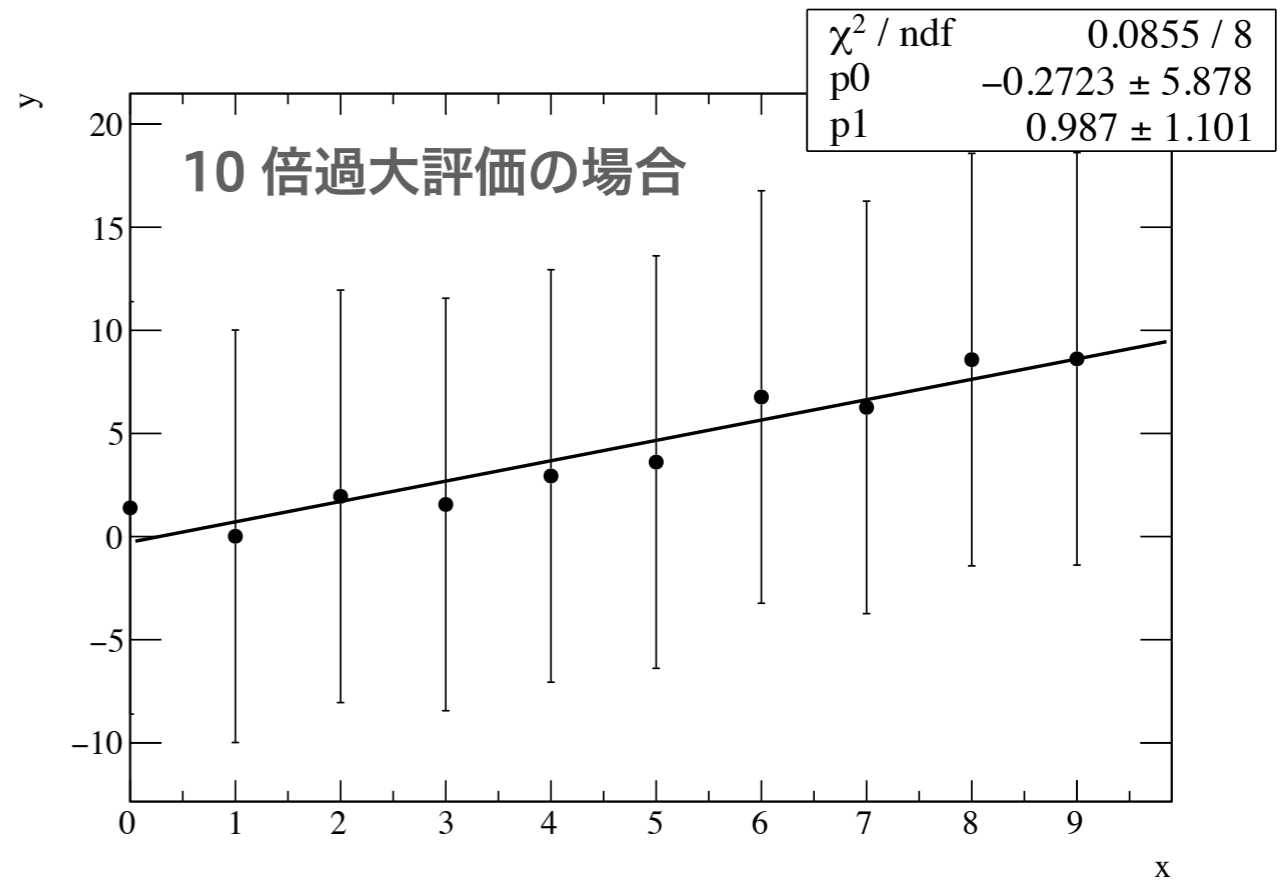
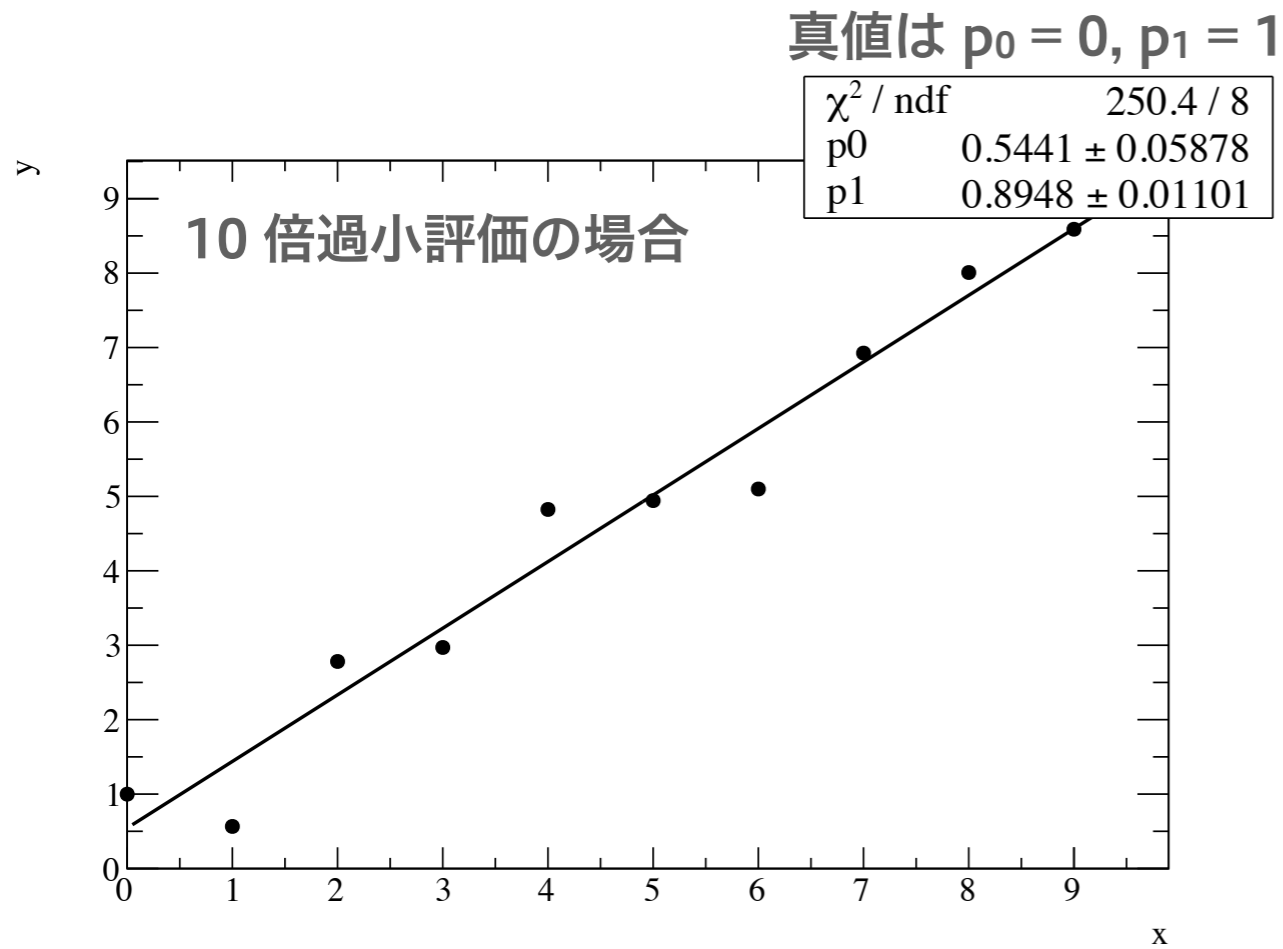
void Sellmeier() {
(略)
    TF1* sellmeier = new TF1("sellmeier", SellmeierFormula, 300, 800, 6);
    sellmeier->SetParameter(0, 1.12); ③ 関数の初期値を与える
    sellmeier->SetParLimits(0, 0.8, 1.2);
    sellmeier->SetParName(0, "B1");
(略)
    TGraph* graph = new TGraph("UVC-200B.csv", "%lg,%lg,%*lg"); ④ ファイルの読み込み
    graph->SetTitle(";Wavelength (nm);Refractive Index;");
    graph->Draw("ap");
    graph->Fit("sellmeier", "w m e 0", "", 300, 700); ⑤ フィット
(略)
    TF1* sellmeier2 = new TF1("sellmeier2", SellmeierFormula, 300, 700, 6);
    sellmeier2->SetParameters(sellmeier->GetParameters());
    sellmeier2->SetLineWidth(1);
    sellmeier2->SetLineColor(2);
    sellmeier2->Draw("l same");
}
```

ついでに好きな関数形でフィットしてみる



- 測定値に誤差がついていない場合、ROOT は全てのデータ点に誤差 1 をつける
- したがって、 χ^2/ndf の値は統計学的にあまり意味がない
- 得られたパラメータの誤差もあまり意味がない
- 大雑把なパラメータを知るには良いが「精度良くパラメータが求まった」とか言わない

誤差の過小評価、過大評価が与える影響



```
$ root
root [0] .x WrongErrorEstimate.C(0.1)
Probability = 1.40682e-49
root [2] .x WrongErrorEstimate.C(10)
Probability = 1
```

ありえないほど小さい確率

ありえないほど大きい確率

```
$ cat WrongErrorEstimate.C
void WrongErrorEstimate(Double_t error = 1.0) {
    TGraphErrors* graph = new TGraphErrors;
    for (int i = 0; i < 10; ++i) {
        double x = i;
        double y = i + gRandom->Gaus(); // Add fluctuation with a sigma of 1
        double ex = 0;
        double ey = error;
        graph->SetPoint(i, x, y);
        graph->SetPointError(i, ex, ey);
    }

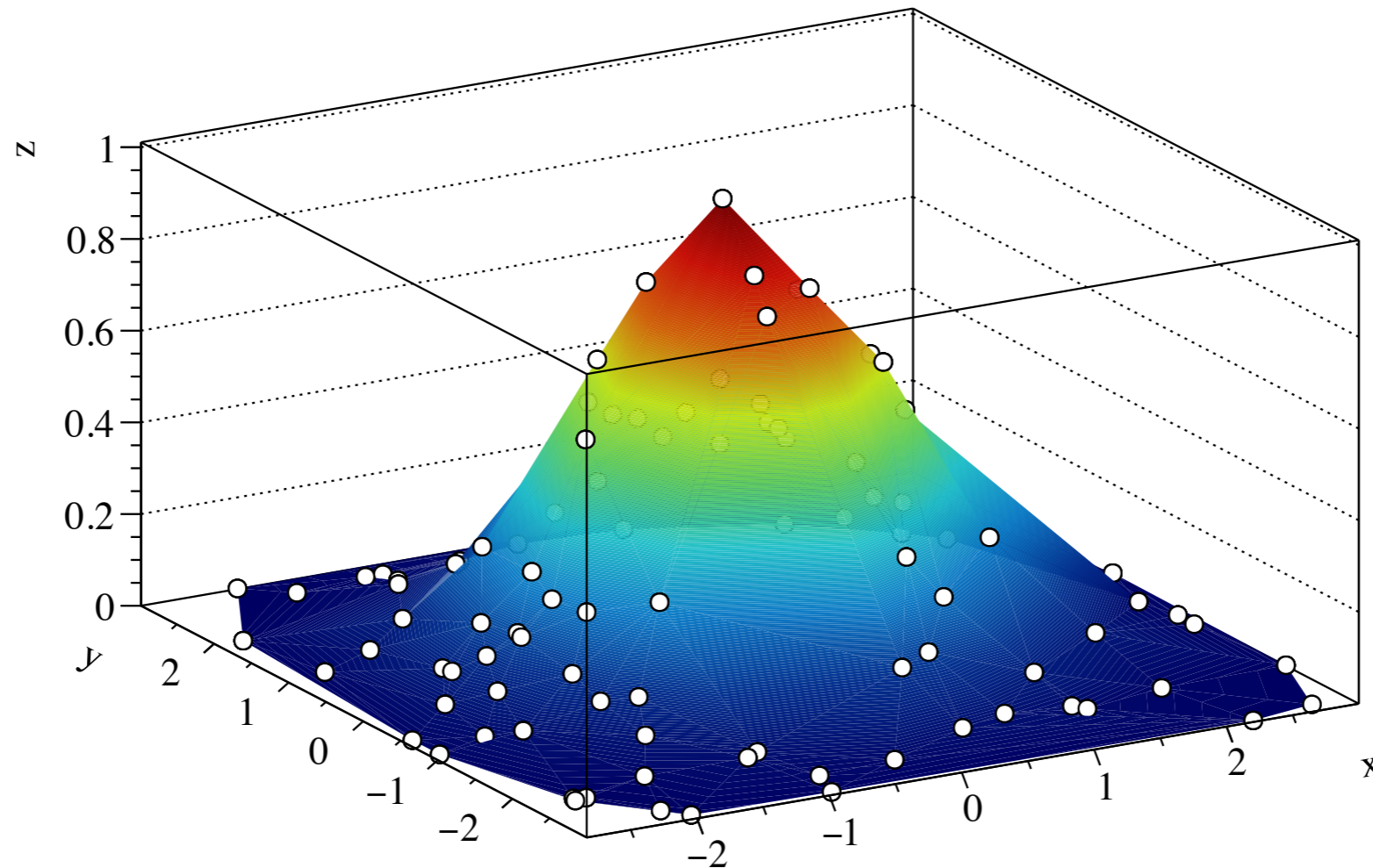
    graph->SetTitle(";x;y;");
    graph->SetMarkerStyle(20);
    graph->Draw("ap");

    gStyle->SetOptFit();
    graph->Fit("pol1");

    std::cout << "Probability = " << graph->GetFunction("pol1")->GetProb() <<
std::endl;
}
```


3 次元グラフ

単純な例



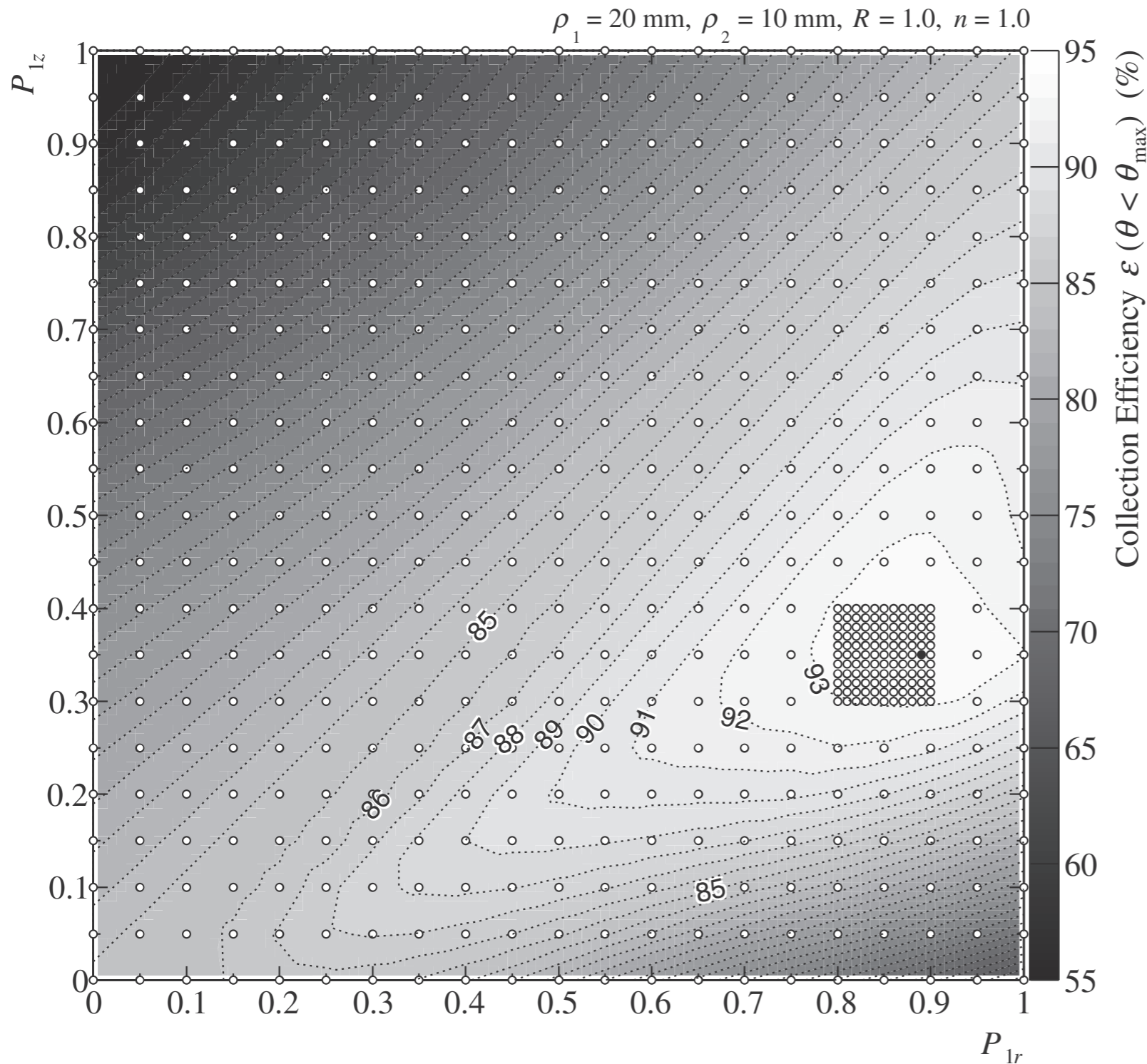
```
$ root
root [0] TGraph2D* graph = new TGraph2D
root [1] for (int i = 0; i < 100; ++i) {
root (cont'ed, cancel with .@) [2] double x = gRandom->Uniform(-3, 3);
root (cont'ed, cancel with .@) [3] double y = gRandom->Uniform(-3, 3);
root (cont'ed, cancel with .@) [4] double z = TMath::Exp(-(x*x + y*y)/2.);
root (cont'ed, cancel with .@) [5] graph->SetPoint(i, x, y, z);
root (cont'ed, cancel with .@) [6]}
root [7] graph->SetTitle(";x;y;z;")
root [8] graph->Draw("p0 tri2")
```

① TGraph2D か TGraph2DErrors を使う

② x/y/z を与える

③ 描画方法は多数ある

実際の使用例



Okumura 2012

- 経験的には、あまり使用機会は多くない
- 2次元ヒストグラムのほうが搭乗頻度は高い
- XY ステージを使った測定など、離散的な測定で使用
- 限られたデータ点数から数値を補間するときにも便利
 - ▶ ドロネー図 (Delaunay diagram) を使って分割される

ROOT オブジェクトの名前

ROOT オブジェクトの名前

```
$ root ① hist: C++ 上の変数名      ② “h”: ROOT の管理する名前
root [0] TH1D* hist = new TH1D("h", ";#it{x};Entries", 5, -5, 5)
(TH1D *) 0x7fdc3c64c040      ③ オブジェクトの実体はメモリ上にある

root [1] TH1D* hist2 = hist      ④ C++ 上で新たに hist2 という変数名を使って
(TH1D *) 0x7fdc3c64c040      同じものを指すことができる

root [2] h      ⑤ ROOT のインタプリタ上では特別に
(TH1D *) 0x7fdc3c64c040      ROOT の管理する名前でもオブジェクトに触れる

root [3] hist->Draw()
root [4] hist2->Draw()      ⑥ 実体はどれも同じなので、結果は同じ
root [5] h->Draw()

root [6] gDirectory->ls()      ⑦ “h” というオブジェクトは、gDirectory に登録されている
OBJ: TH1D      h      : 0 at: 0x7fdc3c64c040

root [7] TGraph* graph = new TGraph      ⑧ TGraph はコンストラクタで命名の必要がない
root [8] graph->SetName("g")      ⑨ 後から名前を付けられる

root [9] gDirectory->GetList()->Add(graph) ⑩ gDirectory に追加すると “g” でもアクセス可
root [10] gDirectory->ls()
OBJ: TH1D      h      : 0 at: 0x7fdc3c64c040
OBJ: TGraph      g      : 0 at: 0x7fdc3c0be610
```

なぜ名前が必要？

- C++ や Python 内での変数名はいつでも変更できてしまう
- 「どのオブジェクトがどれ」と区別をつけるには、ROOT 側で名前をつけておくと便利なことがある
- ROOT オブジェクトを ROOT ファイルに保存するとき、名前がついていないとオブジェクト同士の区別がつかない
- ROOT はヒストグラムと TTree のみに、名前の付与と gDirectory への登録を自動で行う（理由は知らない）

ROOT ファイル

- ROOT のクラスから作られたオブジェクトは、ほとんど全てが ROOT ファイルに保存できる
- 拡張子 .root
- データ収集の際に直接 ROOT ファイルとして保存してしまえば、解析時にいちいち ROOT オブジェクトとして作成し直さなくて良い
 - ▶ 例：オシロの波形を TGraph や TH1 として保存する
- 解析結果も ROOT ファイルにしてしまえば、可搬性が高くなる
- 描画した図も TCanvas のまま保存可能

ROOT ファイルに保存する例

```
$ root
root [0] TH1D* hist = new TH1D("h", ";#it{x};Entries", 5, -5, 5)
root [1] hist->Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [2] TGraph* graph = new TGraph
root [3] graph->SetName("g")
root [4] gDirectory->GetList()->Add(graph)

root [5] TFile f("mydata.root", "recreate") ① ROOT ファイルを新規もしくは作成する
root [6] c1->Write() ② ROOT ファイルにオブジェクトを書き込む
root [7] graph->Write()
root [8] f.Close()
```


ROOT ファイルを開く例

```
$ root
root [0] TFile f("mydata.root")
root [1] f.ls()
TFile**      mydata.root
TFile*       mydata.root
KEY: TCanvas  c1;1c1
KEY: TGraph   g;1
root [2] TGraph* graph = (TGraph*)f.Get("g")
root [3] TCanvas* can = (TCanvas*)f.Get("c1")

root [4] can->Draw()

root [5] TH1* h = (TH1*)can->GetPrimitive("h")
```

① ROOT ファイルを開く

② 中身を確認すると、“c1”という TCanvas と “g”という TGraph が保存されている

③ オブジェクトを取得し、キャストする
Python の場合はキャスト不要
名前がないと、取り出すのが面倒

④ TCanvas は保存時の状態で再度開ける

⑤ TCanvas 内に描画されたオブジェクトも
取り出すことができる

第 3 回のまとめ

- 準備不足でちょっと短かったですが…
- グラフとは何か
- TGraph を使った ROOT でのグラフ作成の例
- カイ二乗フィットと誤差
- ROOT ファイルの扱い

- 分からなかった箇所は、各自おさらいしてください