

ROOT Geant4 Seminar

Sho Nagao

2023 version

目次

- 1. はじめに
- 2. Operating System
- 3. Windows
- 4. ROOT, Geant4 導入方法
- 5. Linux 環境設定
- 6. 鍵認証システム
- 7. バージョン管理システム
- 8. ROOTの使い方
- 9. モンテカルロ法
- 10. Geant4の使い方
- 11. CADの使い方？
- 12. Spiceの使い方？

このセミナーの目標

Linux System の基礎的な知識を学ぶ

解析等で実際に使用するソフトウェアの基本的な使い方を学ぶ

数値シミュレーションの代表的な手法であるモンテカルロ法について学ぶ

C、C++言語を頻繁に使いますが、気軽な気持ちで学びましょう

利用するもの

パソコン

(最低 CPU: Core-i5相当、Memory: 8 GB、Strage:128 GB)

(推奨 CPU: Core-i7相当、Memory: 16 GB、Strage:256 GB)

Windowsの方

ssh環境、X-window環境（VcXsrv）、Windows Subsystem for Linux (WSL v2)、Ubuntu

Linux系の方

特になし

Macの方

X-window環境（XQuartz）

ROOT、Geant4の環境が手元にあるとなお良い

※ インストール方法もセミナー内でサポートします

※ Macの方はリモートPCに接続して Geant4 を動かしたとき、Viewer が上手に表示されない問題がありますのでローカルに持つておくこと推奨。ただし Mac のサポートは不可

資料、参考プログラムは GitHub 上からダウンロード可

<https://github.com/shonagao-nex/RootGeant-Seminar.git>

直接アクセスするか、Clone する

Clone 方法

ターミナル上で

\$ git clone https://github.com/shonagao-nex/RootGeant-Seminar.git

※ ダウンロードしてきたファイルのことを「リポジトリ上にあるファイル」と呼ぶことにします

※ GitHub アカウントを持っていない方は今すぐ作りましょう

※ Git の使い方も少しやります

記載のルール

コマンドを記載する場合、

ユーザー権限で実行するコマンドは頭に「\$」が、

管理者 (root) 権限で実行するコマンドは頭に「#」もしくは sudo コマンドが、

ROOT内で実行するものに関しては「root [0]」が

付きます

各自で変更が必要な個所はイタリックで記載します。

スペースが入るときには明示的に「_」記号を入れるときがあります。

- 
1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方

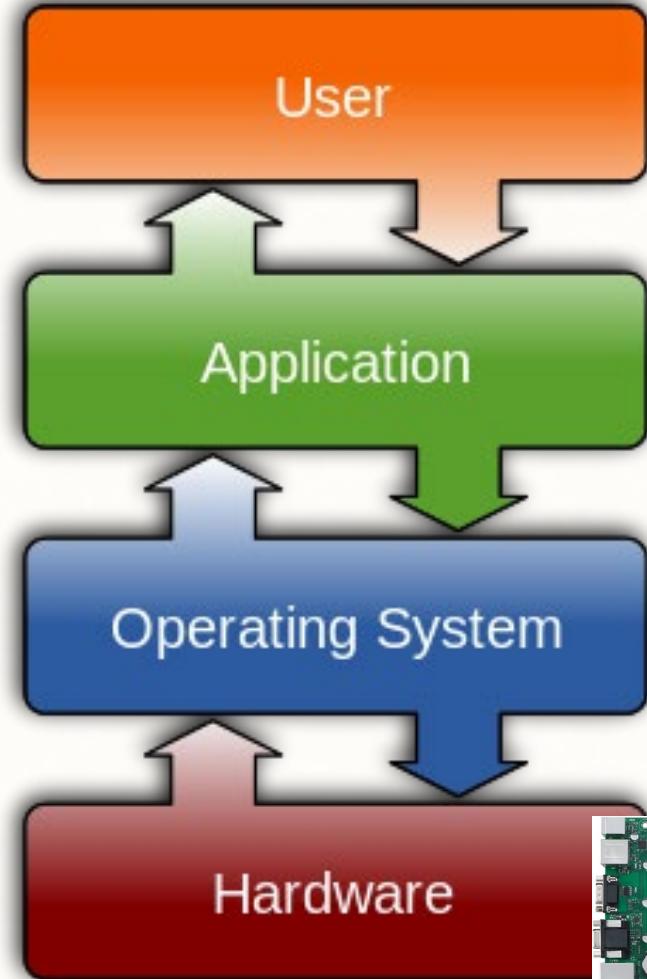
パソコンの構成



Windows、Mac、Linux

マザーボード、CPU、GPU、
メモリ、ストレージ、電源

ROOT、Geant4



プログラミング言語

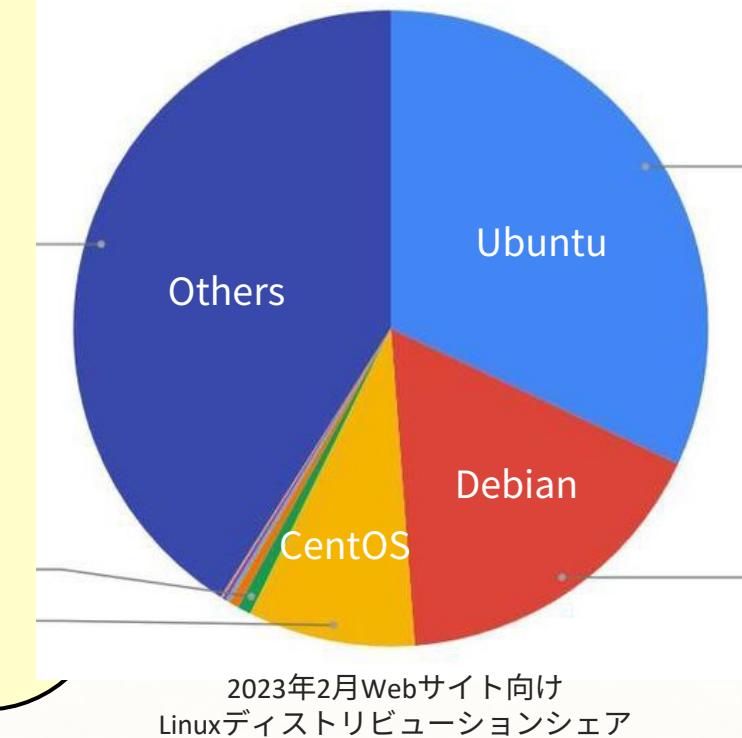
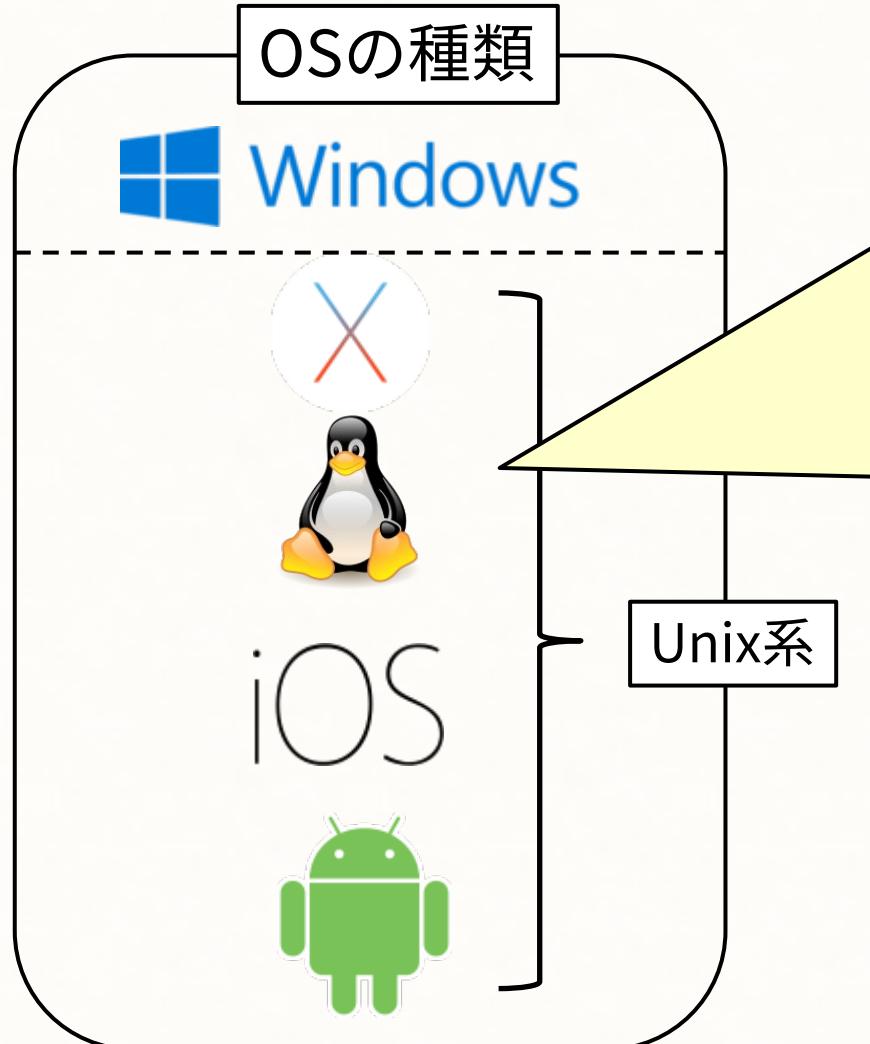
アセンブリ言語

ビット情報



Operating System (OS)

ハードウェアを利用者に分かりやすいよう翻訳する機構



- 
1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方

Windowsの環境構築

別端末からLinux端末で作業するためには、コマンド入力するための ssh 環境と、画像をコントロールする X window system が必要。

Linux OS では両方環境が整っているが、Windowsでは環境の構築が必要。

ssh環境（上から簡単かつ軽量）

PowerShell：Windows10で軽量ならコレ。あまり困らない。

Putty, TeraTerm：以前までの定番。現状PowerShellがあれば使わないかも。

Cygwin：以前までの定番のひとつ。今やWSLの下位互換。

WSL：Windows Subsystem for Linux。ほぼLinux環境。OSの選択肢少ない、根っこ部分はいじれないなどの制限もある。

VMware：フルLinuxを使いたい場合はコレ。これさえあれば何でもできる。

X window環境

Xming：以前までの定番。今は、VcXsrvがお勧め。

VcXsrv：無償で全機能が使える。Xmingの完全上位互換。

昔の方法（今も現役）	必要スペック	汎用性
PuTTY+xming	: ssh 環境のみ	低 ×
Cygwin	: Linuxもどき	
coLinux	: ほぼLinux	
VMware, VirtualBox	: Linuxそのもの	高 ○

ここでは、Windows Terminal + WSL2 + Ubuntu + VcXsrv を使った方法を紹介

VcXsrv環境構築

VcXsrvとは

VcXsrvのダウンロード

VcXsrvのインストール

VcXsrvの起動・設定

VcXsrv とは、Windows 上で動作する X window サーバー。

Xming が良く使われてきたが、更新が途絶えている、速度が遅いなどの問題を孕んでいる。

VcXsrv と ssh 環境を組み合わせることで、他のPCからの画像を飛ばすことが可能となる。

<https://sourceforge.net/projects/vcxsrv/>

からダウンロード可能（リンクが生きていれば）。

ダウンロードしたものをダブルクリックしてインストールすればよい。

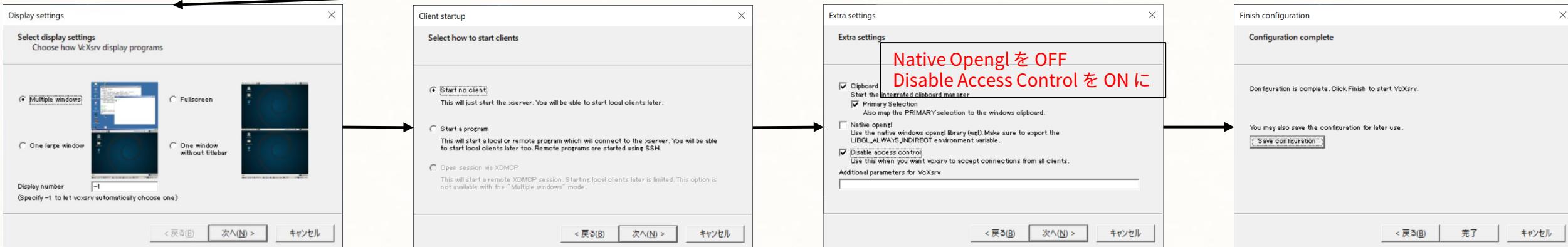
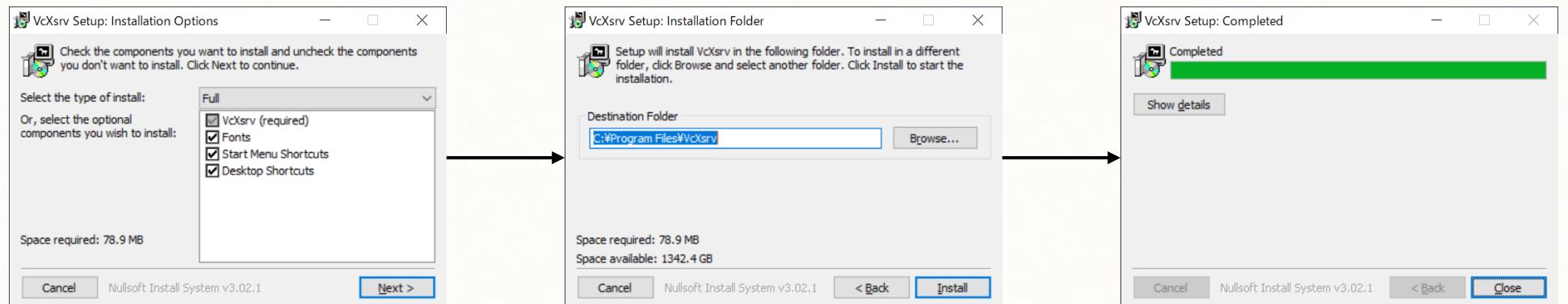
VcXsrv環境構築

VcXsrvとは

VcXsrvのダウンロード

VcXsrvのインストール

VcXsrvの起動・設定



ほぼデフォルトで良いが、OpenGLとAccess Control設定は忘れずに。

Configurationファイルを保存し、スタートアップ登録すれば、自動起動できる。

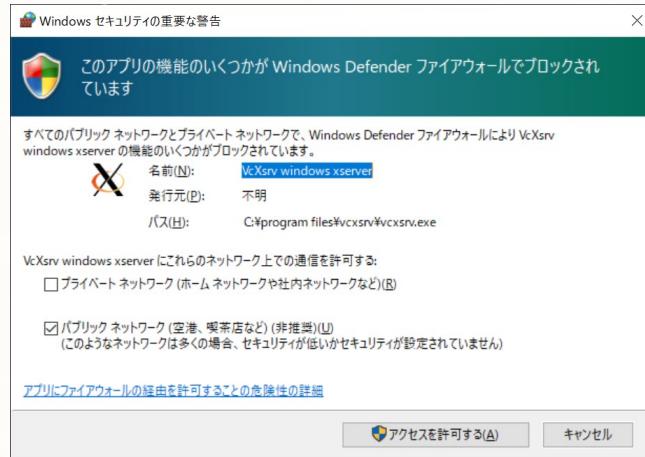
VcXsrv環境構築

VcXsrvとは

VcXsrvのダウンロード

VcXsrvのインストール

VcXsrvの起動・設定



この画面が出たら「アクセスを許可する」をクリック
状況次第でプライベートネットワークも許可しておく



デスクトップにこのようなアイコンが

Windowsキー+Rをクリックして「ファイル名を指定して実行」を立ち上げる

shell:startup と入力

スタートアップフォルダが立ち上がる所以、そこに config.xlaunc を移動（自動起動させるため）

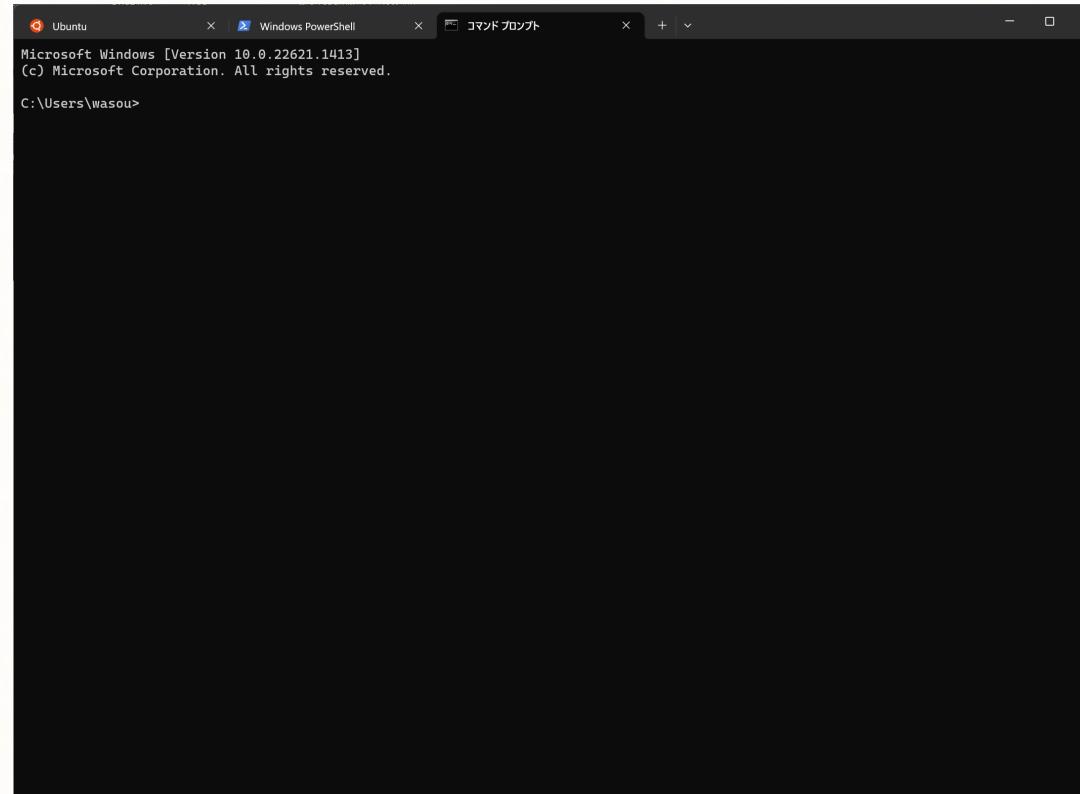
Windows Terminal の準備

Linux の Terminal のように端末として利用可能

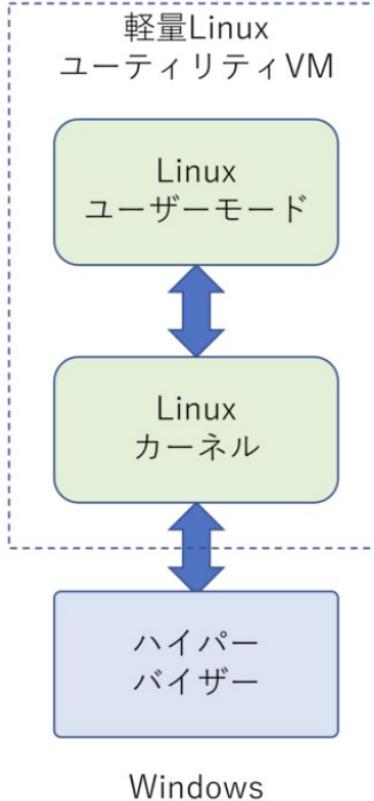
同一端末内で Ubuntu、PowerShell、コマンドプロンプトなどを起動可能（便利）

ターミナルの外観も良い感じ

Microsoft Store からインストール可能



WSL2の準備 (Windows Subsystem for Linux 2)



Windows 10 バージョン 2004 以上 (ビルド 19041 以上) または Windows 11 を実行している必要あり
それ以前 (x64 システムの場合:バージョン 1903 以降、ビルド 18362 以上。ARM64 システムの場合:バージョン 2004 以降、ビルド 19041 以上。) も利用可能だが、インストール方法がやや煩雑なのでここではパス

1. 管理者として PowerShell を開き、以下を実行
`wsl --install`
2. インストールが完了したら、PCを再起動
3. 再起動後、自動でUbuntuが起動する（はず）のでユーザ名とパスワードを設定
4. 正しくインストールできていれば、PowerShell 上から以下のように確認できる

```
PS C:\Users\wasou> wsl -l -v
      NAME      STATE      VERSION
*  Ubuntu    Running     2
```

Tips

WSL上のUbuntuとWindows間でファイルのやり取りが可能

デフォルトでは Ubuntu のホームディレクトリは以下の場所に生成されている

¥¥wsl.localhost¥Ubuntu¥home

また、コマンドが重複するので、設定→操作から コピーを「ctrl+shift+c」ペーストを「ctrl+shift+v」としておくのがおススメ

※ インストールできない場合は「Windows の機能の有効化または無効化」を開き「Linux 用 Windows サブシステム」にチェックが入っているか確認

- 
1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方

WSLに色々インストール

Windows Terminal を起動して WSLを起動

以下の手順でOSアップデート、ソフトウェアインストール例

Ubuntu にインストールする例、

Redhat系にインストールする場合は dnf コマンドを利用

パッケージ名も異なる (libx11-dev → libX11-devel)

アップデート（定期的に）

```
$ sudo apt update      (RedHat系の方は、 sudo dnf update、 sudo yum updateの場合もある)  
$ sudo apt upgrade
```

ROOT, Geant4等々に必要なライブラリのインストール

```
$ sudo apt install cmake g++ libx11-dev libxpm-dev libxft-dev libxext-dev libxmu-dev  
$ sudo apt install gfortran libgif-dev libtiff-dev libjpeg-dev libxml2-dev libfftw3-dev  
$ sudo apt install libgsl-dev cmake-curses-gui xxHash  
$ sudo apt install libxerces-c-dev qtbase5-dev libqt53drender5 libmotif-dev
```

Python関係

```
$ sudo apt install python3 python3-dev pip3
```

Python3.9を導入したい場合

```
$ sudo apt install software-properties-common  
$ sudo apt install python3.9 python3.9-dev pip3.9  
$ sudo pip3.9 install numpy scipy matplotlib jupyter pandas
```

WSLにROOTをインストール

インストール

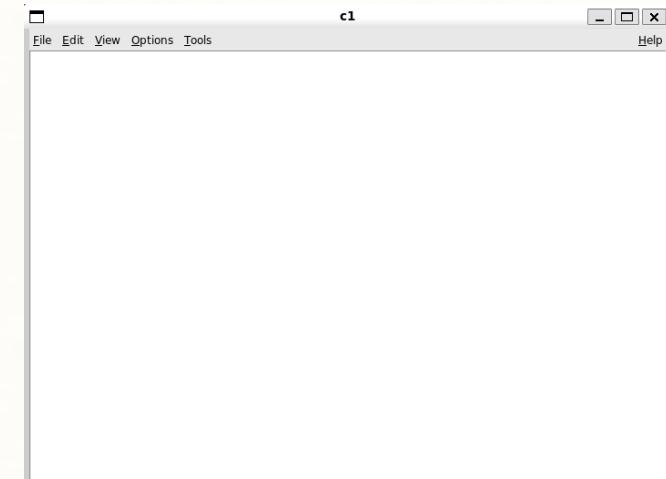
```
$ cd ~/Downloads  
$ wget https://root.cern/download/root_v6.24.08.source.tar.gz; tar zxvf root_v6.24.08.source.tar.gz  
$ mkdir root_v6.24.08_build; cd root_v6.24.08_build  
$ sudo mkdir /usr/local/share/applications/root_v6.24.08  
$ cmake ..//root-6.24.08/  
$ ccmake .  
    CMAKE_INSTALL_PREFIX      /usr/local/share/applications/root_v6.22.08  
    fftw3, fortran, minimal, minuit2, pyroot, roofit あたりがONになっているか確認  
    "c"でコンパイル、"g"でMakefile生成、エラーが出る場合はパッケージ不足している可能性  
$ make -j4 (4-threads でコンパイル、数字は自分のPCのスペックと相談)  
$ sudo make install
```

環境設定

```
~/.bashrc に以下を追記  
export ROOTSYS=/usr/local/share/applications/root_v6.24.08  
source $ROOTSYS/bin/thisroot.sh  
$ source ~/.bashrc で設定を反映
```

起動

```
$ root  
root[0] gROOT->GetVersion() としてインストールしたROOTのバージョンが出れば成功  
root[0] TCanvas *c1 = new TCanvas() として右のような画面が出てきたら成功  
root[0] .q としてROOTを出る
```



WSLにGeant4をインストール

インストール

```
$ cd ~/Downloads  
$ wget https://gitlab.cern.ch/geant4/geant4/-/archive/v10.7.4/geant4-v10.7.4.tar.gz; tar zxvf geant4.10.07.p04.tar.gz  
$ mkdir geant4.10.07.p04_build; cd geant4.10.07.p04_build  
$ sudo mkdir /usr/local/share/applications/geant4.10.07.p04  
$ cmake ..../geant4.10.07.p04/  
$ ccmake .  
    CMAKE_INSTALL_PREFIX      /usr/local/share/applications/geant4.10.07.p04  
    GEANT4_BUILD_MULTITHREADED  ON  
    GEANT4_INSTALL_DATA        ON  
    GEANT4_USE_GDML           ON  
    GEANT4_USE_OPENGL_X11     ON  
    GEANT4_USE_QT              ON  
    GEANT4_USE_RAYTRACER_X11   ON  
    GEANT4_USE_SYSTEM_EXPAT    ON  
    GEANT4_USE_XM              ON  
  
"c"でコンパイル、"g"でMakefile生成、エラーが出る場合はパッケージ不足している可能性  
$ make -j4; sudo make install
```

環境設定

```
~/.bashrc に以下を追記  
export G4INSTALL=/usr/local/share/applications/geant4.10.07.p04  
source $G4INSTALL/bin/geant4.sh  
$ source ~/.bashrc で設定を反映
```

- 
1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方

Linux 環境設定

自分独自の環境がある人はパス

```
$ cd ~
```

```
$ git clone https://github.com/shonagao-nex/RootGeant-Seminar.git
```

リンクを貼る

```
$ ln -s RootGeant-Seminar/environments/bashrc .bashrc
```

inputrc, colorrc, rootlogon.C, rootkinematics.C, rootmacro.C, vim, vimrc も同様

頭に . を付けるのを忘れずに

設定反映

```
$ source .bashrc
```

※ 色設定はダークモードでの利用を想定しているので適宜変更

環境変数設定ファイルコメント

今回は bash を利用（利用シェアが圧倒的に高く、情報量が多いため）。
csh, tcsh, zsh 等々を利用してても良い

```
HISTCONTROL=ignoreboth
export PROMPT_COMMAND="history -a; history -c; history -r; $PROMPT_COMMAND"
shopt -u histappend
export HISTTIMEFORMAT='%F %T '
export HISTSIZE=10000
export HISTFILESIZE=20000
```

```
### ROOT ###
export ROOTSYS=/usr/local/share/applications/root_v6.24.08
source $ROOTSYS/bin/thisroot.sh
### GEANT4 ###
export G4INSTALL=/usr/local/share/applications/geant4.10.7.4
source $G4INSTALL/bin/geant4.sh
```

```
eval `dircolors -b ~/.colorrc`
alias ls='ls -CF --color=auto'
alias cp='cp -i'
alias mv='mv -i'
alias rm='rm -i' } 強く推奨
alias cd='cd -P'
alias vi='vim'
alias root='root -l'
alias clean='rm -f *~'
```

1. はじめに
2. Operating System
3. Windows
4. ROOT, Geant4 導入方法
5. Linux 環境設定
- 6. 鍵認証システム**
7. バージョン管理システム
8. ROOTの使い方
9. モンテカルロ法
10. Geant4の使い方

鍵認証システム

リモートマシンへアクセスする場合、鍵認証システムを利用することを推奨

公開鍵：暗号化専用の鍵 (id_rsa.pub) ホストが製作、クライアントが使う

秘密鍵：復号専用の鍵 (id_rsa) ホストが製作、ホストが死守

lambda から farm へ鍵認証を使ってログインする場合、

クライアント : lambda **サーバー** : farm

鍵の作成および設定方法

```
$ cd ~/.ssh  
$ ssh-keygen -t rsa          (秘密鍵、公開鍵の作成)  
    Enter file in which to save the key (/home/username/.ssh/id_rsa): ← Enter  
    Created directory '/home/username/.ssh'. ← Enter  
    Enter passphrase (empty for no passphrase): ← パスフレーズを入力 (大文字小文字数字等々を織り交ぜて)  
    Enter same passphrase again: ← もう一度パスフレーズを入力  
  
$ chmod 600 id_rsa          (鍵のアクセス権設定)  
$ scp id_rsa.pub farm:~/ssh/ (公開鍵を転送)  
$ cd ~/.ssh  
$ cat id_rsa.pub >> authorized_keys (公開鍵を登録)  
$ chmod 600 authorized_keys (公開鍵のアクセス権設定)
```

使い方

```
$ ssh -Yi .ssh/id_rsa username@farm
```

鍵認証の補足

より便利に使うため、クライアント側で以下の操作を行う

```
$ cd ~/.ssh
```

config ファイルを作成、編集

Host farm

hostname farmのIP Address (サーバー上、 /sbin/ifconfig で確認可能)

identityfile ~/.ssh/id_rsa

user username

```
$ chmod 600 config
```

(configの権限設定)

~/.bashrc に以下を追加

alias farm ‘ssh -Y farm’

```
$ source .bashrc
```

(環境変数を設定)

```
$ eval `ssh-agent`
```

(ssh-agentを起動)

```
$ ssh-add ~/.ssh/id_rsa
```

(ssh-agentに鍵を登録)

```
$ farm
```

(ログイン)

ssh-agent を常に起動したいのであればその旨を環境変数に追記すればよい。

ファイルコピー

Unix系の場合

scp コマンド

```
$ scp コピー元 コピー先
```

```
$ scp nagao@lambda.phys.tohoku.ac.jp:~/test.txt ./
```

(Lambdaホームディレクトリの test.txt を自分のPCの今のディレクトリにコピー)

rsync コマンド

```
$ rsync -auv --progress nagao@lambda.phys.tohoku.ac.jp:~/test.txt ./
```

Windowsの場合

WSLでSCPコマンド等々利用可能だが、専用ソフト（WinSCP）も便利

<https://winscp.net/eng/docs/lang:jp> にアクセスしてダウンロード・インストール



1. ホスト名を入力
2. ユーザー名を入力
3. 保存（ログインの隣？）して
4. ログインをクリック
5. パスワードを入力

1. はじめに
2. Operating System
3. Windows
4. ROOT, Geant4 導入方法
5. Linux 環境設定
6. 鍵認証システム
7. バージョン管理システム
8. ROOTの使い方
9. モンテカルロ法
10. Geant4の使い方

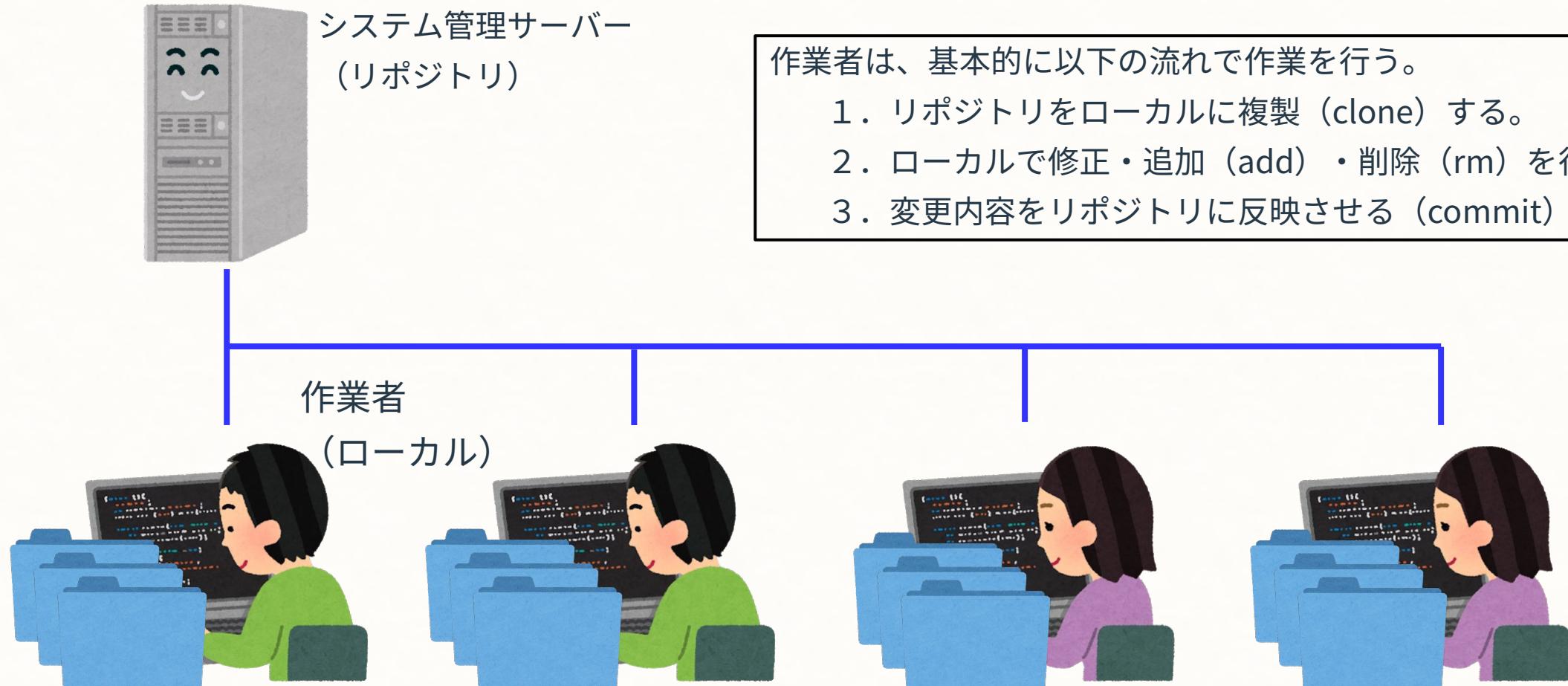
ブランチを切ってローカルリポジトリにコミット
リモートリポジトリにプッシュしてマージする
を理解する



GitHubとは、Gitを利用した、開発者を支援するWebサービスです。
自分でサーバーを立ててリポジトリを作成する必要なし。
リポジトリを public にすれば、他人とコードの共有も可能です。
執筆中の論文の管理にも使えます。
開発のレビューなども簡単にできるので非常に便利。是非アカウントを作っておきましょう。

バージョン管理システムとは

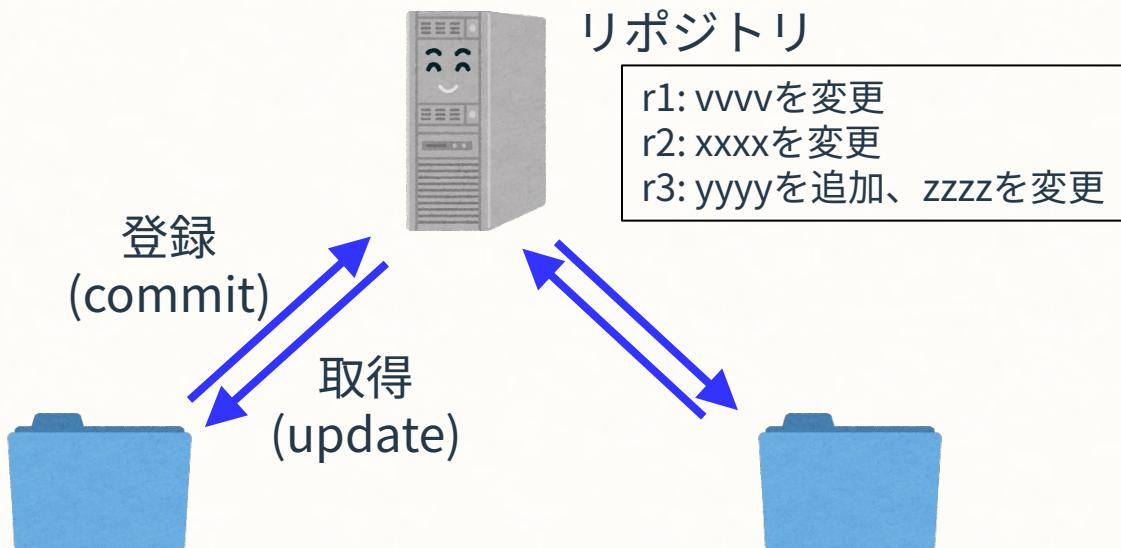
バージョン管理システム（version control system）とは、コードなどの編集履歴を管理、保管するシステム



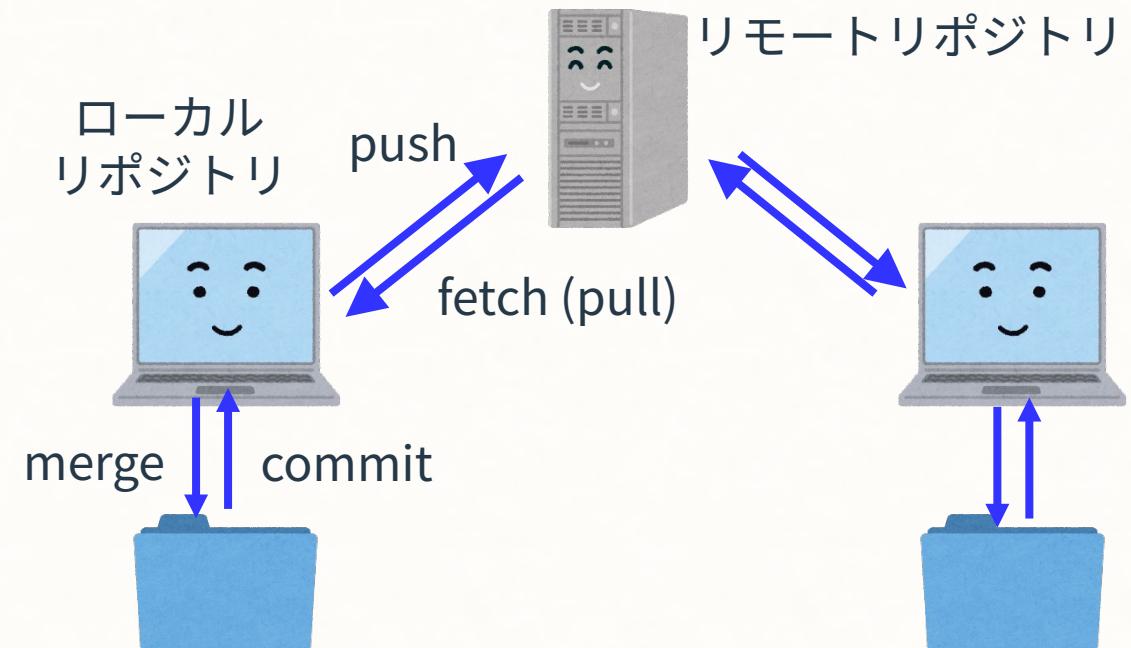
バージョン管理システムの種類

バージョン管理システムは「集中型」「分散型」に分けられる。

集中型：バージョン管理を一か所のリポジトリがリビジョン番号を割り振って行う。CVS、Subversion



分散型：各作業者がリポジトリを保有している（ローカルリポジトリ）。Git、Mercurial



集中型の方が分かりやすいが、作業者は他の作業者の内容がリポジトリに反映されるまで待つ必要がある。近年は分散型の Git が頻繁に使われる。

作業者の流れ (Subversion)

リポジトリ名は vcs。

リポジトリに既にある test.cc を変更、新たに hoge.cc を追加して登録することを考える。

- | | |
|--|---------------|
| 1. svn checkout vcs | リポジトリの内容を複製 |
| 2. 内容を編集、追加 | |
| 3. svn update | 最新の状態にする |
| 4. svn add hoge.cc | ファイルを追加 |
| 5. svn commit test.cc hoge.cc -m “modify test.cc, add hoge.cc” | 作業内容をリポジトリに反映 |

※一度チェックアウトてしまえば、次からは手順1は必要ない。

その他

変更ログを確認したい	svn log
リポジトリとの差を確認したい	svn diff test.cc
現在の状況を確認したい	svn status
変更内容をリポジトリの内容に戻したい	svn revert test.cc

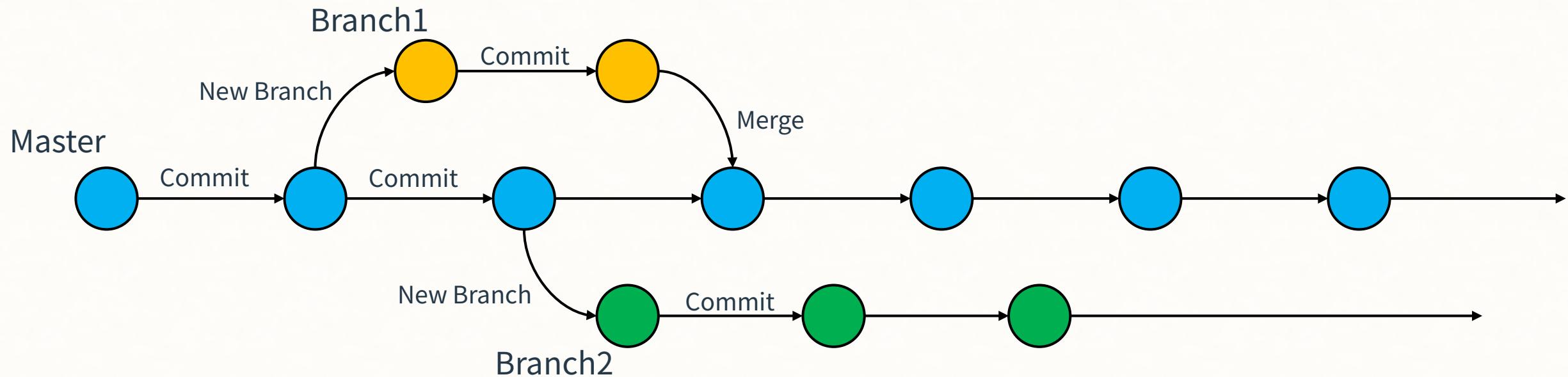
ブランチという考え方（Git）

Git はブランチという機能が充実している。他者の変更内容と競合を避けるため、作業者は、

1. Master の内容を Branch を切って編集作業を進め
 2. ローカルリポジトリに変更内容を逐次 commit
 3. ある程度形になったところで Branch のリモートリポジトリに push し
 4. プルリクエストを作成
 5. 管理者が Master に Branch の内容を merge する。

といったように進めることが多い。Subversionでも似たような機能があるが充実していない。

開発方針によっては、Master のみで進めることもある。



作業者の流れ (Git)

リポジトリ名は vcs。 master ブランチから local ブランチを作成。リポジトリに既にある test.cc を変更、新たに hoge.cc を追加。最終的に master ブランチにマージすることを考える。

- | | |
|--|----------------------|
| 1. git clone vcs | リモートリポジトリの内容を複製 |
| 2. git pull | 最新の状態にする |
| 3. git branch local | local ブランチを作成 |
| 4. git checkout local | local ブランチに移動 |
| 5. 内容を編集、追加 | |
| 6. git add hoge.cc | ファイルを追加 |
| 7. git commit test.cc hoge.cc -m “modify test.cc, add hoge.cc” | 作業内容をローカルリポジトリに反映 |
| 8. git push origin local | ローカルをリモートに反映 |
| 9. git checkout master | master ブランチに移動 |
| 10. git pull origin master | master ブランチを最新の状態にする |
| 11. git merge local | local を master にマージ |
| 12. git push origin master | 内容をリモートに反映 |

※一度クローンてしまえば、次からは手順 1 は必要ない。
※ブランチ機能を使わないのであれば灰色は不要。

Subversion と Git のコマンド

内容	Subversion	Git
リポジトリを作成	svn create	git init
リモートリポジトリの内容を複製	svn checkout	git clone
作業ディレクトリを更新	svn update	git pull
ローカルリポジトリのみ更新		git fetch
ローカルリポジトリの内容を作業ディレクトリに反映		git merge
ファイルの追加（移動、削除）	svn add (mv, rm)	git add (mv, rm)
変更をリポジトリに反映	svn commit	git commit
変更をリモートリポジトリに反映		git push
変更の取消	svn revert	git reset
状態確認	svn status	git status
差分確認	svn diff	git diff
ログの確認	svn log	git log
ブランチの作成、一覧		git branch
ブランチの切替		git checkout
ブランチのマージ	svn merge	git merge

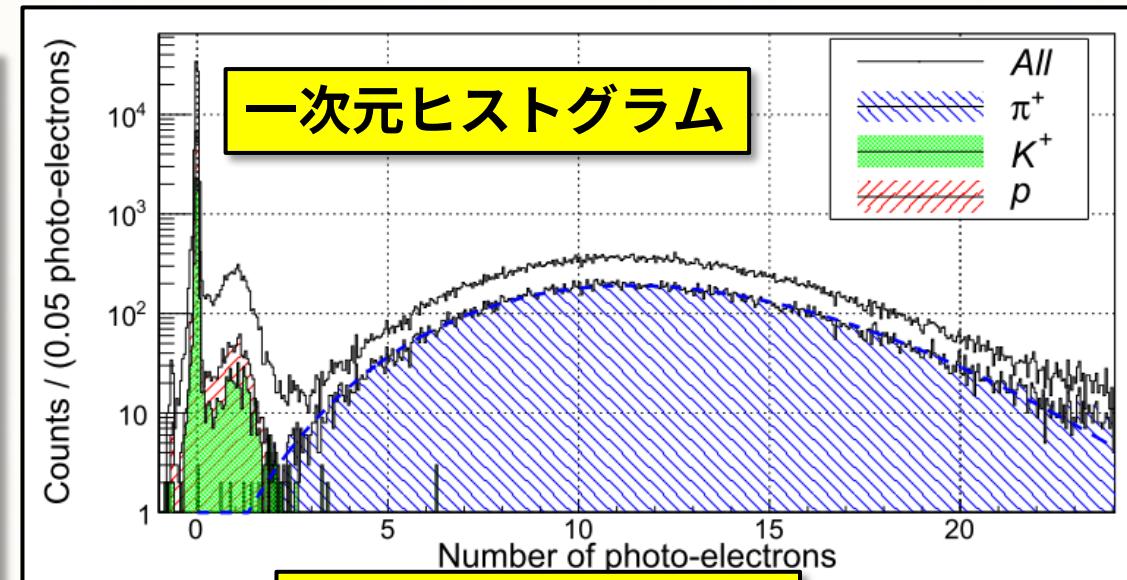
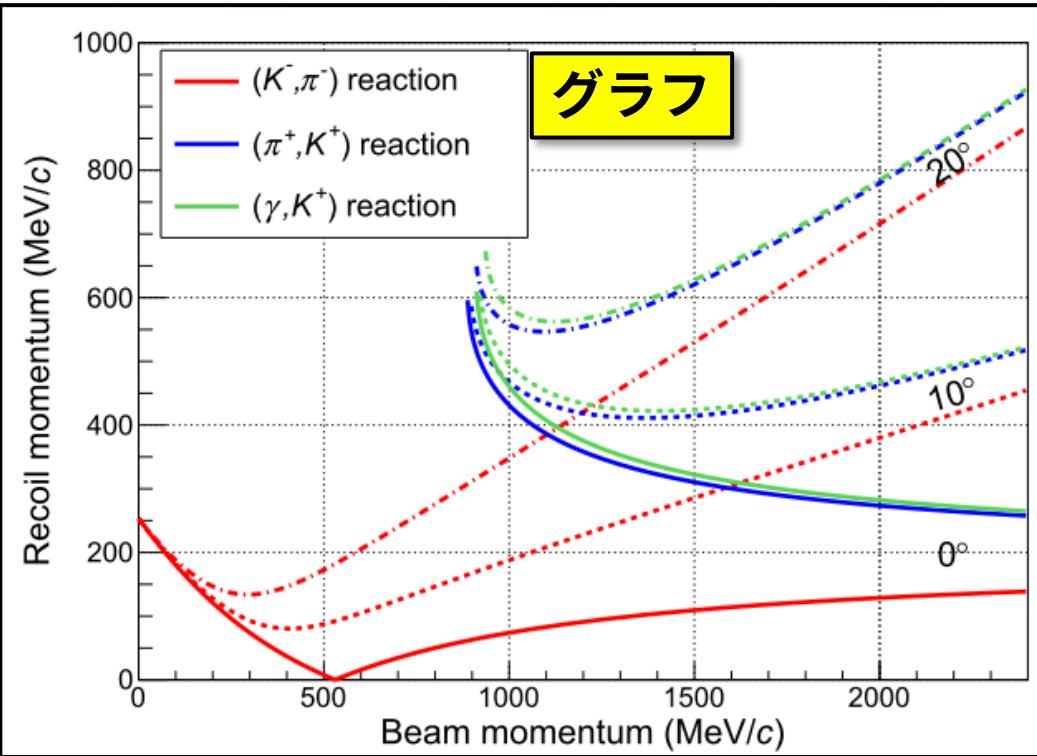
1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

ROOTとは

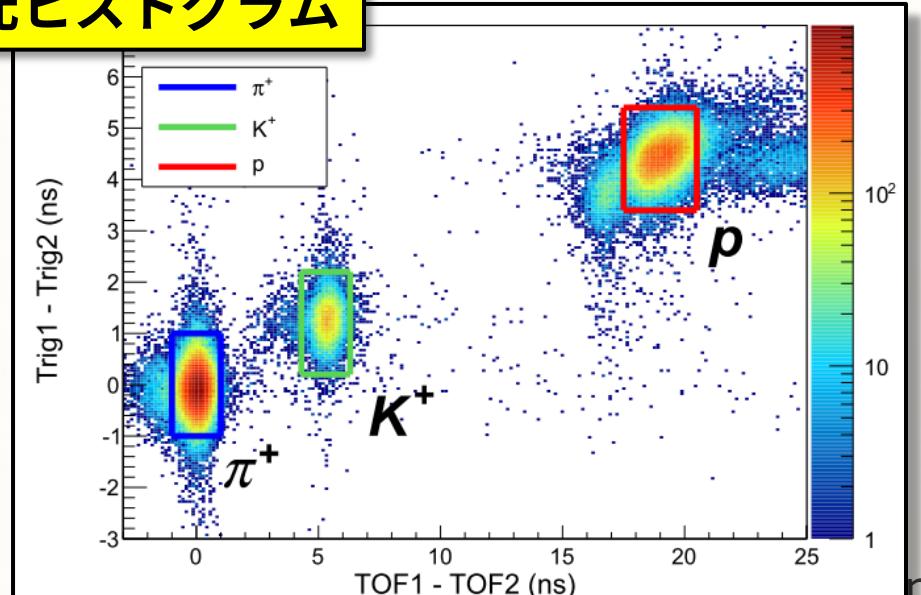
素粒子原子核の解析で利用されるフリープログラム、CERNにおいて開発
データ解析環境・ライブラリを提供
同様なプログラムPAWの後継版
C++ベースで記述できる（PAWはFortranベース）
ホームページ：<https://root.cern.ch/>



作成例

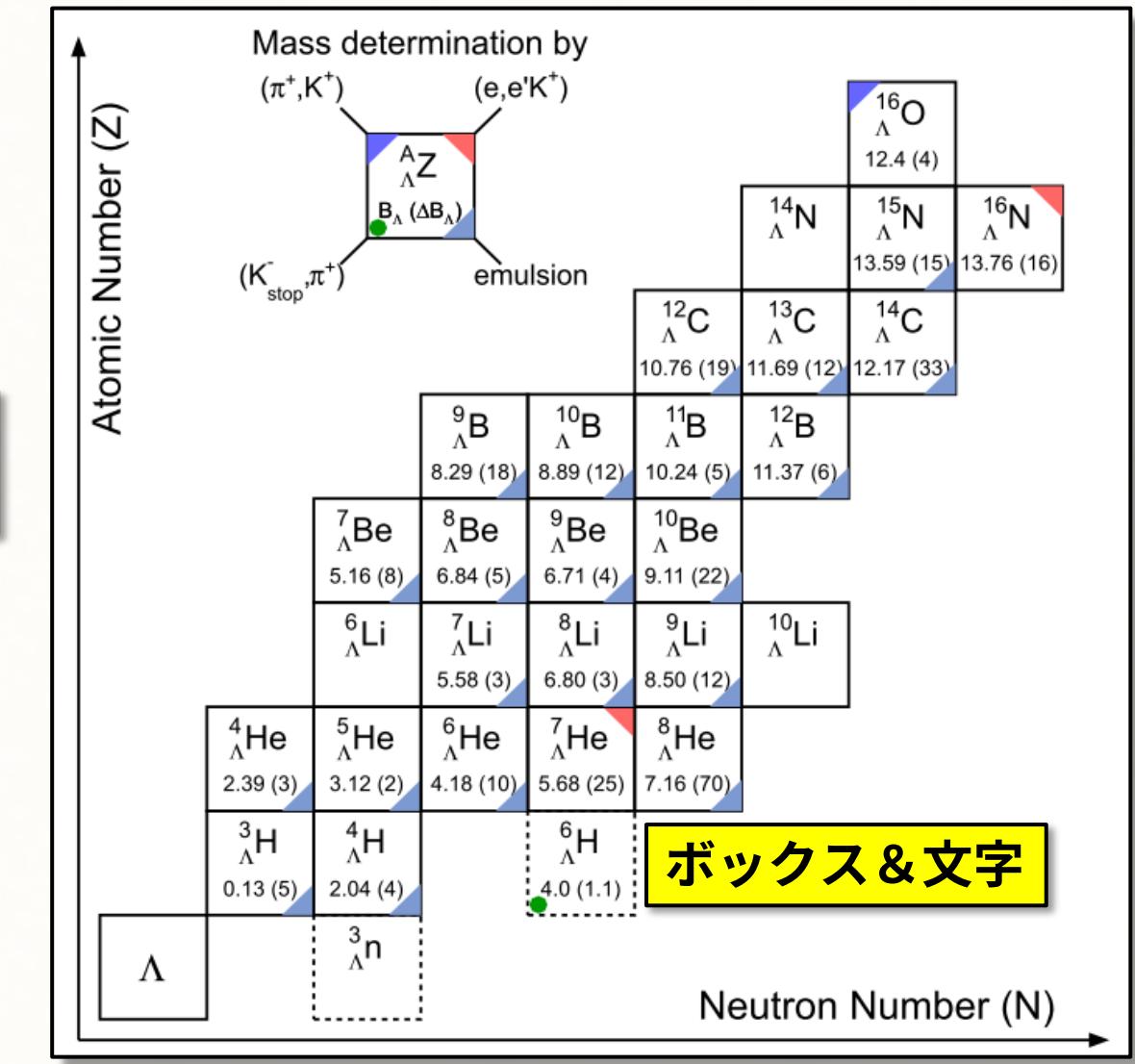
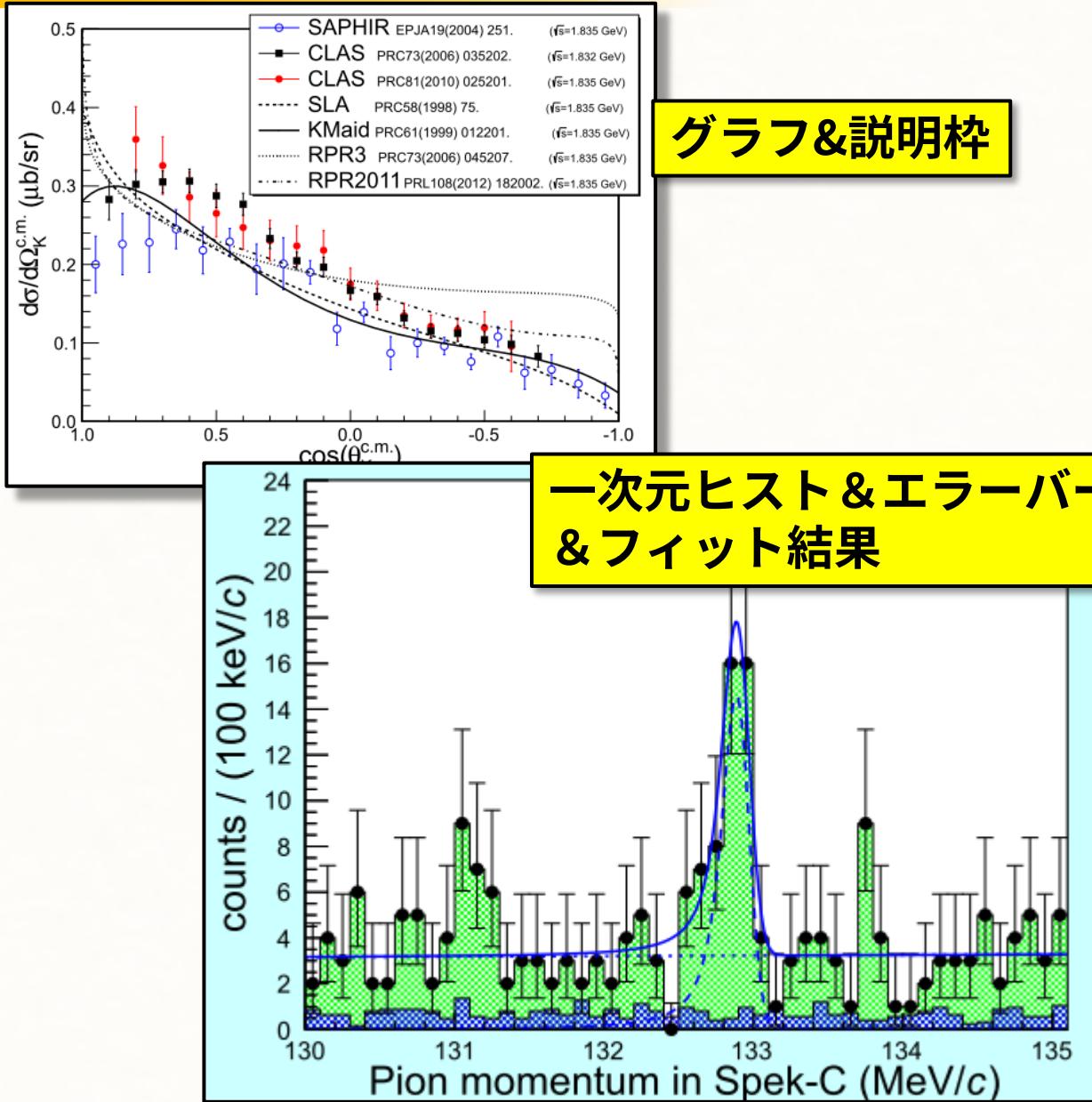


二次元ヒストグラム



演算、乱数計算
テキスト読込
ps, pdf, png, jpg, root, C 出力

作成例



導入方法

インストール方法は、[このページ](#)で解説済み

.bashrc, .cshrc, .zshrc 等々自分のログインシェルでROOTへのパス、環境を通しておく
自分のログインシェルは、

```
$ echo $SHELL
```

で確認可能。ログインシェルは、chsh コマンドで変更可能

bash系

```
export ROOTSYS=/usr/local/cern/root6.24.08  
source $ROOTSYS/bin/thisroot.sh
```

csh系

```
setenv ROOTSYS /usr/local/cern/root6.24.08  
source $ROOTSYS/bin/thisroot.sh
```

設定できていれば、以下のようになるはず

```
$ echo $ROOTSYS  
/usr/local/cern/root6.24.08  
$ echo $PATH  
/usr/local/cern/root6.24.08/bin  
$ echo $LD_LIBRARY_PATH  
/usr/local/cern/root6.24.08/lib
```

ROOTの起動
\$ root
root [0]
となるはず

設定ファイルの準備（オプション）

ROOTの初期設定ファイルは標準で `~/.rootlogon.C` となっている

（自分の作業ディレクトリに `rootlogon.C` がある場合はそちらが優先して読み込まれる）

設定ファイルの例がリポジトリの `environments/rootlogon.C` にある

`ln -s environments/rootlongon.C ~/.rootlogon.C` としてリンクを貼るなどする（任意）

`.rootlogon.C` から `.rootmacro.C`, `.rootkinematics.C` を読み込んでいるのでそれもリンクを貼ること

`.rootlogon.C` の例、基本的には論文や発表で ROOT の画像を掲載することを想定して作る

タイトル、ラベルが小さい、図編集時にグリッド線、統計ボックスが邪魔などを想定している

```
{  
    gROOT->SetStyle("Plain");           ← 画面の背景色を白に（必須）  
    gStyle->SetOptDate(0);              ← 余計な日付表示をOFFに  
    gStyle->SetPadGridX(0);             ← グリッドOFFに  
    gStyle->SetPadGridY(0);  
    gStyle->SetFrameLineWidth(1);        ← 線の太さは最細に  
    gStyle->SetLineWidth(1);  
    gStyle->SetTextFont(42);            ← かっこいいフォントに  
    gStyle->SetGridWidth(1);  
    gStyle->SetNdivisions(505); // tertiary*10000 + secondary*100 + first   ← ラベルの表示桁数  
    gStyle->SetOptStat("ei");          ← 統計情報は Entry と Integral のみ  
    const Int_t NRGBs = 5;              ← 2Dヒストの色設定をかっこよく  
    const Int_t NCont = 99;  
    Double_t stops[NRGBs] = { 0.00, 0.34, 0.61, 0.84, 1.00 };  
    Double_t red[NRGBs] = { 0.00, 0.00, 0.87, 1.00, 0.51 };  
    Double_t green[NRGBs] = { 0.00, 0.81, 1.00, 0.20, 0.00 };  
    Double_t blue[NRGBs] = { 0.51, 1.00, 0.12, 0.00, 0.00 };  
    TColor::CreateGradientColorTable(NRGBs, stops, red, green, blue, NCont);  
    gStyle->SetNumberContours(NCont);  
    gErrorIgnoreLevel = kError;          ← うるさい警告はOFFに  
}
```

1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

まずは触ってみる

計算機

```
root [0] 1+2  
(int) 3  
root [1] 1.+2.1  
(double) 3.1000000      正確に3.1ではないことに注意（浮動小数点計算）  
root [2] cout<<Form("%.30lf",1.+2.1)<<endl  
3.10000000000000088817841970013
```

キャンバス表示

名前、タイトル、X座標、Y座標、幅、高さ

```
root [3] TCanvas *c1 = new TCanvas("c1", "c1", 100, 100, 800, 600);
```

ROOTから出る

```
root[4] .q
```

ROOT のコマンドのひとつ
(TCanvas というクラスが用意されている)
ROOTのクラスはT***で定義されている
グローバル変数はg***



TCanvas : キャンバス
TH1D : double型一次元ヒストグラム
TH2D : double型二次元ヒストグラム
TGraph : グラフ
TF1 : 一次元の関数（フィットに利用）
TTree, TChain : 変数データ管理 (Ntuple)
TLegend : 凡例
TLatex, TText : テキスト
TProfile : 二次元ヒストのプロファイル

覚えておくべきテクニック

cern root tcanvas とブラウザで検索すれば、クラスリファレンスが出てくる

<https://root.cern.ch/doc/master/classTCanvas.html>

Public Member Functions

`TCanvas (Bool_t build=kTRUE)`

Canvas default constructor. More...

`TCanvas (const char *name, const char *title="", Int_t form=1)`

Create a new canvas with a predefined size form. More...

`TCanvas (const char *name, const char *title, Int_t ww, Int_t wh)`

Create a new canvas at a random position n. More...

`TCanvas (const char *name, const char *title, Int_t wtopx, Int_t wtopy, Int_t ww, Int_t wh)`

Create a new canvas. More...

`TCanvas (const char *name, Int_t ww, Int_t wh, Int_t wind)`

Create an embedded canvas, i.e. More...

`virtual ~TCanvas ()`

Canvas destructor. More...

`virtual void Browse (TBrowser *b)`

Browse. More...

`TVirtualPad * cd (Int_t subpadnumber=0)`

Set current canvas & pad. More...

`void Clear (Option_t *option="")`

Remove all primitives from the canvas. More...

`virtual void Cleared (TVirtualPad *pad)`

Emit pad Cleared signal. More...

`void ClearPadSave ()`

`void Close (Option_t *option="")`

Close canvas. More...

が、わざわざブラウザで検索のは面倒

`root [0] TCanvas *c1 = new TCanvas()` ←ここでタブキー

とすることで TCanvas クラスの定義の方法が出てくる

TCanvas クラスで用意されている関数は、

`root [1] c1->` ←ここでタブキー

や

`root [1] c1->Get` ←ここでタブキー

とすることで検索することができる。

基本、英語で名前が定義されている。

キャンバスのタイトルを決めたい → SetTitle("タイトル")

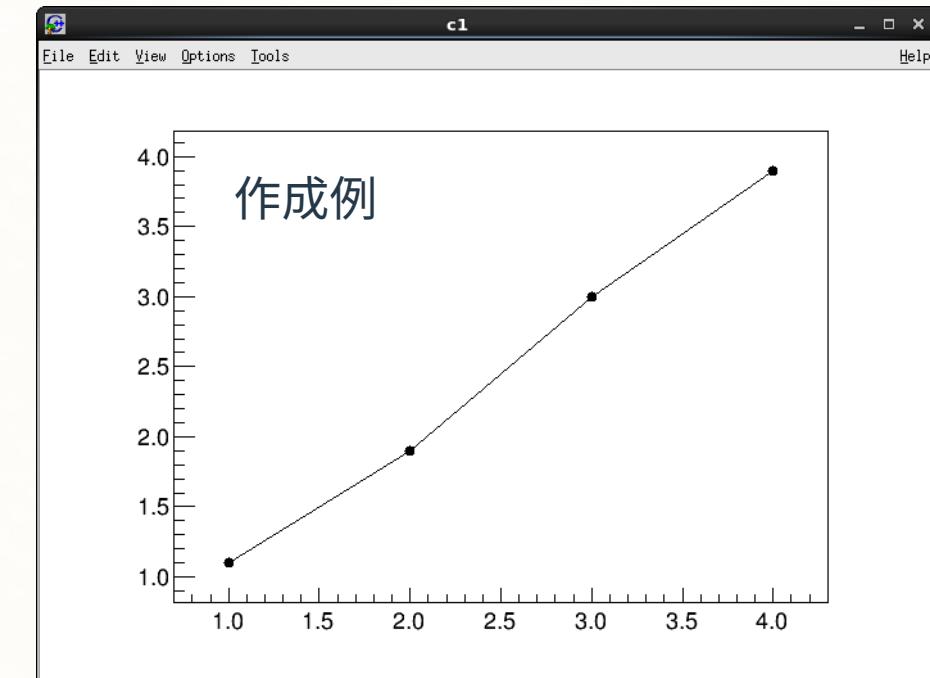
キャンバスのサイズを決めたい → SetWindowSize(x,y)

グラフの手書き

```
root [0] TGraph *g = new TGraph();           ← グラフの定義  
root [1] g->SetPoint(0, 1., 1.1);          ← 点を指定  
root [2] g->SetPoint(1, 2., 1.9);  
root [3] g->SetPoint(2, 3., 3.0);  
root [4] g->SetPoint(3, 4., 3.9);  
root [5] g->SetMarkerStyle(20);             ← マーカー形状の指定  
root [6] TCanvas *c1 = new TCanvas("c1","c1"); ← キャンバスの作成  
root [7] g->Draw("APL");                  ← グラフをキャンバス上に描画 (A:新規、P:点有、L:直線で結ぶ)
```

連続するコマンドは、以下のように{}で閉じることで記述可能

```
root [0] {  
root [1] TGraph *g = new TGraph();  
root [2] g->SetPoint(0,1.,1.1);  
root [3] g->SetPoint(1,2.,1.9);  
root [4] g->SetPoint(2,3.,3.0);  
root [5] g->SetPoint(3,4.,3.9);  
root [6] g->SetMarkerStyle(20);  
root [7] TCanvas *c1 = new TCanvas("c1","c1");  
root [8] g->Draw("APL");  
root [9] }
```



GUI操作

タイトル編集

```
g->SetTitle("title");
```

軸タイトル編集

```
g->GetXaxis()->SetTitle("x-title");
```

ラインの編集

```
g->SetLineColor(2);  
g->SetLineWidth(2);  
g->SetLineStyle(7);
```

点の編集

```
g->SetMarkerColor(2);  
g->SetMarkerSize(2);  
g->SetMarkerStyle(2);
```

対数軸表示

```
gPad->SetLogy(1);
```

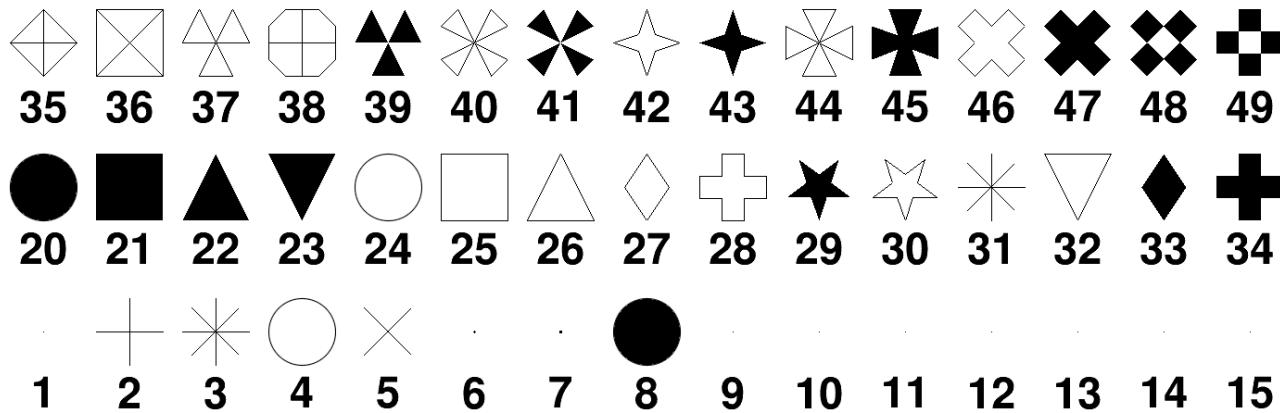
フィット

レンジ指定

グリッド指定

スタイル、色

40	41	42	43	44	45	46	47	48	49
30	31	32	33	34	35	36	37	38	39
20	21	22	23	24	25	26	27	28	29
10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9



1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ**
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

マクロ

いちいちコマンドを打ち込んでいくのは不便
ファイルに処理を記述して（マクロにして）解析する
マクロにも様々な種類があるので、例題を見ながら真似するべし
標準の例題は、\$ROOTSYS/tutorials 以下にある
ココではさらに簡単かつ実用的なマクロを使って学習する
マクロの例はリポジトリの ROOT-example 以下にある

マクロの実行方法

- 1) 直接実行、2) ROOTを開いて実行、3) マクロをロードして実行
- 4) ROOTでコンパイルして実行、5) g++でコンパイルして実行

使い勝手×、コーディングの速さ○、デバッグの容易性×、汎用性×：利用せず

ex011_ExeMacro.cc

```
{  
    TGraph *g = new TGraph();           // Definition of TGraph class "g", call empty constructor  
    g->SetPoint(0,1.,1.1);             // Set 1st point (X=1., Y=1.1)  
    g->SetPoint(1,2.,1.9);             // Set 2nd point (X=2., Y=1.9)  
    g->SetPoint(2,3.,3.0);             // Set 3rd point (X=3., Y=3.0)  
    g->SetPoint(3,4.,3.9);             // Set 4th point (X=4., Y=3.9)  
    g->SetMarkerStyle(20);             // Set Marker Style (filled circule)  
    TCanvas *c1 = new TCanvas("c1","c1"); // Definition of TCanvas class "c1", name="c1", title="c1"  
    g->Draw("APL");                  // Draw "g" as an option "A(Draw Axis) & P(Draw Point) & L (Draw Line)"  
}  
    *g を delete する必要はない
```

動かし方 1

farm:> \$ root ex011_ExeMacro.cc

動かし方 2

farm:> \$ root
root [0] .x ex011_ExeMacro.cc

ROOT の専用コマンド

.x : マクロの実行

マクロの実行方法

- 1) 直接実行、 2) ROOTを開いて実行、 3) マクロをロードして実行
- 4) ROOTでコンパイルして実行、 5) g++でコンパイルして実行

使い勝手△、コーディングの速さ○、デバッグの容易性×、汎用性×：そこそこ利用

ex012_ExeMacro2.cc

```
void ex012_ExeMacro2(){ ← ファイル名と関数名が一致している場合に利用可能
    TGraph *g = new TGraph();
    g->SetPoint(0,1.,1.1);
    g->SetPoint(1,2.,1.9);
    g->SetPoint(2,3.,3.0);
    g->SetPoint(3,4.,3.9);
    g->SetMarkerStyle(20);
    TCanvas *c1 = new TCanvas("c1","c1");
    g->Draw("APL");
}
```

動かし方 1

```
farm:> $ root ex012_ExeMacro2.cc
```

動かし方 2

```
farm:> $ root
root [0] .x ex012_ExeMacro2.cc
```

マクロの実行方法

- 1) 直接実行、2) ROOTを開いて実行、3) マクロをロードして実行
- 4) ROOTでコンパイルして実行、5) g++でコンパイルして実行

使い勝手○、コーディングの速さ○、デバッグの容易性×、汎用性△：頻繁に利用

ex013_LoadMacro.cc

```
void ex013_LoadMacroA(){  
    TGraph *g = new TGraph();  
    g->SetPoint(0,1.,1.1);  
    g->SetPoint(1,2.,1.9);  
    g->SetPoint(2,3.,3.0);  
    g->SetPoint(3,4.,3.9);  
    g->SetMarkerStyle(20);  
  
    TCanvas *c1 = new TCanvas("c1","c1");  
    g->Draw("APL");  
}  
  
void ex013_LoadMacroB(double val){  
    TGraph *g = new TGraph();  
    g->SetPoint(0,1.,1.1);  
    g->SetPoint(1,2.,1.9);  
    g->SetPoint(2,3.,val);  
    g->SetPoint(3,4.,3.9);  
    g->SetMarkerStyle(20);  
  
    TCanvas *c1 = new TCanvas("c1","c1");  
    g->Draw("APL");  
}
```

サッと解析したい場合に利用

動かし方

```
farm:> $ root  
root [0] .L example013.cc  
root [1] ex013_LoadMacroA();  
root [2] ex013_LoadMacroB(3.0);
```

ROOT の専用コマンド
.L: マクロのロード

マクロの実行方法

- 1) 直接実行、2) ROOTを開いて実行、3) マクロをロードして実行
- 4) ROOTでコンパイルして実行、5) g++でコンパイルして実行

使い勝手○、コーディングの速さ○、デバッグの容易性×、汎用性△：頻繁に利用

ex014_LoadMacro2.cc

```
void setgraph(TGraph *g, double val){ // Set Graph point function
    g->SetPoint(0,1.,1.1);
    g->SetPoint(1,2.,1.9);
    g->SetPoint(2,3.,val);
    g->SetPoint(3,4.,3.9);
}

void ex014(double val){      // Definition of example function
    TGraph *g = new TGraph();
    setgraph(g,val);        // Call setgraph function
    g->SetMarkerStyle(20);

    TCanvas *c1 = new TCanvas("c1","c1");
    g->Draw("APL");
}
```

前項の拡張

動かし方

```
farm:> $ root
root [0] .L ex014_LoadMacro2.cc
root [1] ex014();
```

ROOT の専用コマンド

.L: マクロのロード

マクロの実行方法

- 1) 直接実行、2) ROOTを開いて実行、3) マクロをロードして実行
- 4) ROOTでコンパイルして実行、5) g++でコンパイルして実行

使い勝手△、コーディングの速さ×、デバッグの容易性△、汎用性△：使わない

ex015_CompROOT.cc

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "TCanvas.h"
#include "TGraph.h"
#include "TAxis.h"

void ex015(){
    TGraph *g = new TGraph();
    g->SetPoint(0,1.,1.1);
    g->SetPoint(1,2.,1.9);
    g->SetPoint(2,3.,3.0);
    g->SetPoint(3,4.,3.9);
    g->SetMarkerStyle(20);
    g->SetTitle("Y v.s X");
    g->GetXaxis()->SetTitle("X (mm)");
    g->GetYaxis()->SetTitle("Y (mm)");

    TCanvas *c1 = new TCanvas("c1","c1");
    g->Draw("APL");
}
```

最初にライブラリ・ヘッダーファイルをインクルードする必要あり

記述はそれなりにC言語・C++である必要がある

\$ root

root [0] .L ex015_CompROOT.cc+ : コンパイル
root [1] ex015()

その他オプションもある

root [] .L ex015_CompROOT.cc++ : 再コンパイル
root [] .L ex015_CompROOT.cc+g : デバック情報出力

マクロの実行方法

- 1) 直接実行、2) ROOTを開いて実行、3) マクロをロードして実行
- 4) ROOTでコンパイルして実行、5) g++でコンパイルして実行

使い勝手○、コーディングの速さ×、デバッグの容易性○、汎用性○：頻繁に利用

ex016_CompGCC.cc

```
#include <iostream>
#include <string>
#include <stdio.h>
#include "TCanvas.h"
#include "TGraph.h"
#include "TAxis.h"
#include "TApplication.h"
```

Cの記述に従ってインクルード

```
int main(int argc, char** argv) {

    TApplication *theApp =
        new TApplication("App", &argc, argv);

    TGraph *g = new TGraph();
    g->SetPoint(0,1.,1.1);

    TCanvas *c1 = new TCanvas("c1","c1");
    c1->Divide(1,1);
    c1->cd(1);
    g->Draw("APL");

    theApp->Run();
    return 0;
}
```

C++なので、メイン文が必要（引数 argc, **argv が必要）

TApplication (TRint) を呼ぶ

動かし方

```
$ g++ -Wall -O2 `root-config --cflags` -o ex016_CompGCC ex016_CompGCC.cc `root-config --libs`  
もしくは、$ make ex016_CompGCC  
$ ./ex016_CompGCC
```

最後に Run() を実行
返し値0

言語、コンパイルは完全にC++に従う

実行方法比較

	書き易さ	実行の速さ	デバッグのし易さ
直接打ち	○	×	×
.x で実行	○	×	△
ロードして (.L) 実行	○	×	△
ROOTでコンパイルして実行	○	○	○
g++でコンパイルして実行	△	○	○

私の場合

直接打ち：簡単な解析

.x, .Lで実行：ちょっとした分布をサクッと描画したい

g++でコンパイル：本格的な解析、時間が必要な解析をしたい！ ROOTを起動することなく結果を保存したい

ROOTでコンパイル：使わない（ROOTでコンパイルするぐらいなら、g++でコンパイルした方が便利）

ex017 と ex018 を動かして比較

GUI操作

特殊文字

#Delta : Δ

#it{#Delta} : Δ

a^{b}_{c} : a^b_c

タイトル

ΔE v.s Position

4.0
3.5
3.0
2.5
2.0
1.5
1.0

対数表示にしたい
グリッドを表示したい

軸タイトル & 単位

ΔE (MeV)

X (mm)

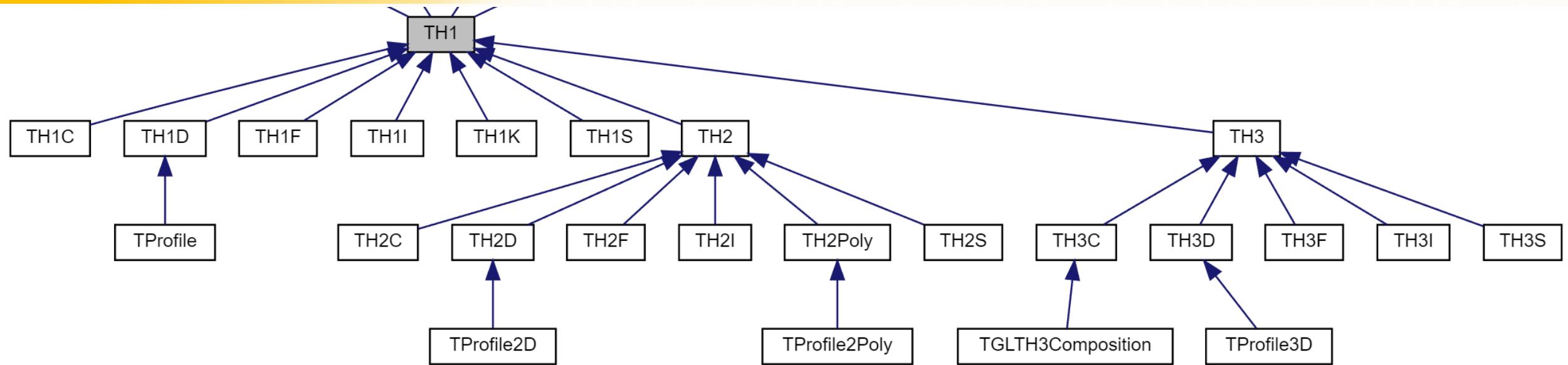
1.0 1.5 2.0 2.5 3.0 3.5 4.0

マーカーの色・形を変えたい

ラインの色を変えたい

1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

ヒストグラムクラス



TH1F : 一次元ヒストグラム float型

TH1D : 一次元ヒストグラム double型

TH2D : 二次元ヒストグラム double型

TH1D TH1D(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)

ヒストグラム

ex021_Hist.cc

```
void ex021(double mean = 10., double wid = 2.){
    double min_x = mean-wid*5;
    double max_x = mean+wid*5;

    TF1 *f = new TF1("f","gaus",min_x,max_x);
    f->SetLineWidth(2);
    f->SetLineColor(4);
    f->SetLineStyle(7);
    f->SetParameters(10,mean,wid);

    TH1D *h1 = new TH1D("h1","Hist",100,min_x,max_x);

    for(int i=0;i<100000;i++){
        double val = gRandom->Gaus(mean,wid);
        h1->Fill(val);
    }

    TCanvas *c1 = new TCanvas("c1","c1",600,500);
    c1->cd(1);
    h1->Draw();
}
```

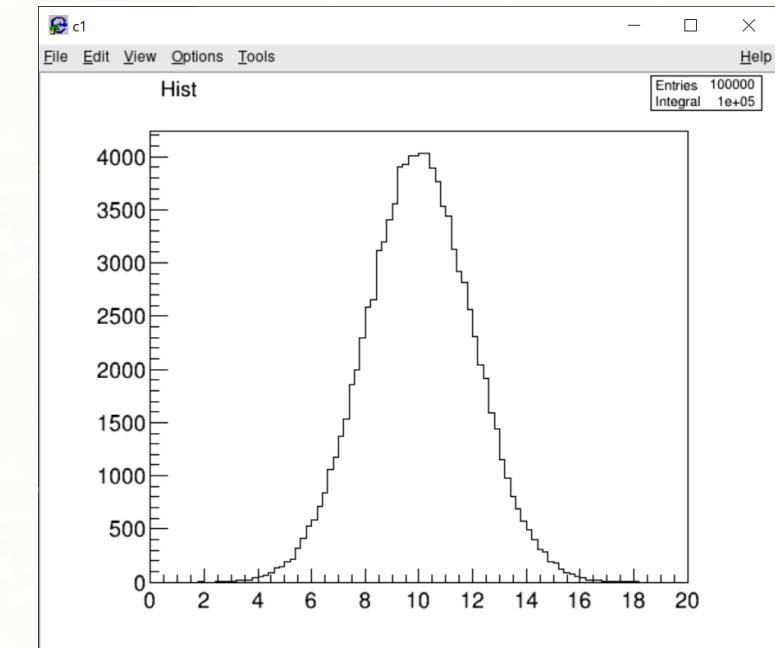
中心値と幅の初期設定
ヒストグラムの範囲設定用の変数

フィット関数の定義 (Gaussian, min_x~max_x)
線幅2
線色青
Dashed Line
関数の初期値設定 (const, mean, sigma)

ヒストグラムの定義 (min_x~max_x を 100 ビンで)

ガウス分布に従ってランダム生成
ヒストグラムにつめる

キャンバスに描画



ヒストグラム設定

ex022_HistSetting.cc

```
void ex022(double mean = 10., double wid = 2.){
    TF1 *f = new TF1("f","gaus",min_x,max_x);
    f->SetParameters(10,mean,wid);
    f->SetNpx(1000);

    TH1D *h1 = new TH1D("h1","Hist022",100,min_x,max_x);
    h1->SetTitle("Hist022");
    h1->GetXaxis()->SetTitle("X (mm)");
    h1->GetXaxis()->CenterTitle();
    h1->GetXaxis()->SetTitleSize(0.06);
    h1->GetXaxis()->SetTitleOffset(1.00);
    h1->GetYaxis()->SetTitle(Form("counts / %.1lf mm", (max_x-min_x)/100.));
    h1->GetYaxis()->CenterTitle();
    h1->GetYaxis()->SetTitleSize(0.06);
    h1->GetYaxis()->SetTitleOffset(1.30);
    h1->SetFillStyle(3002);
    h1->SetFillColor(3);

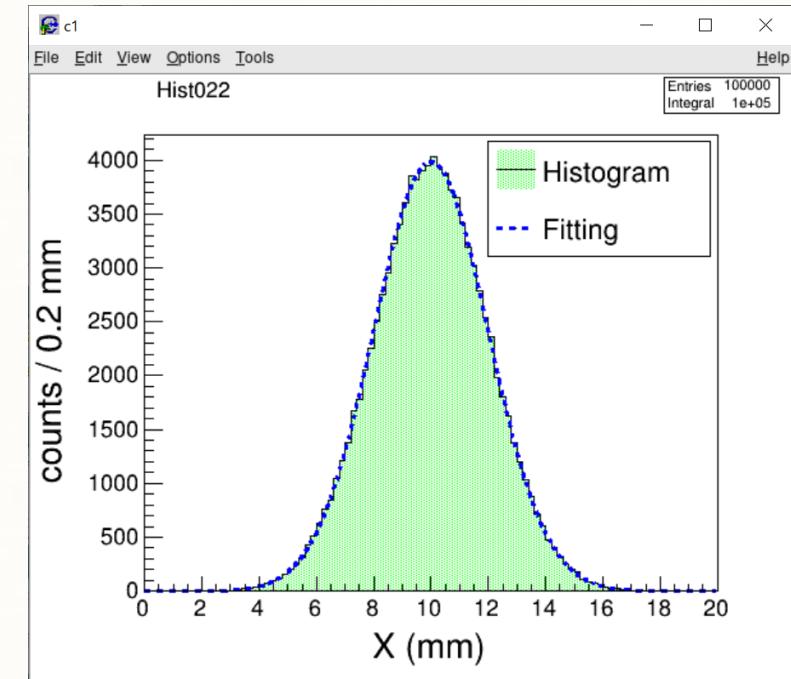
    TCanvas *c1 = new TCanvas("c1","c1",600,500);
    c1->Divide(1,1,1E-5,1E-5);
    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.10);
    h1->Draw();
}
```

フィット関数の描画ポイント数
線の色
線の形式

ヒストグラムのタイトル
X軸タイトル
X軸タイトルを中心に
X軸タイトルのサイズ
X軸タイトルの場所
Y軸タイトル (Formという便利なコマンドがROOTにある)

フィルスタイル
フィルする色

キャンバスを分ける (マージンも小さく設定)
描画のマージン設定



レジェンド

TLegend (Double_t x1, Double_t y1, Double_t x2, Double_t y2, const char *header="", Option_t *option="brNDC")

ex022_HistSetting.cc

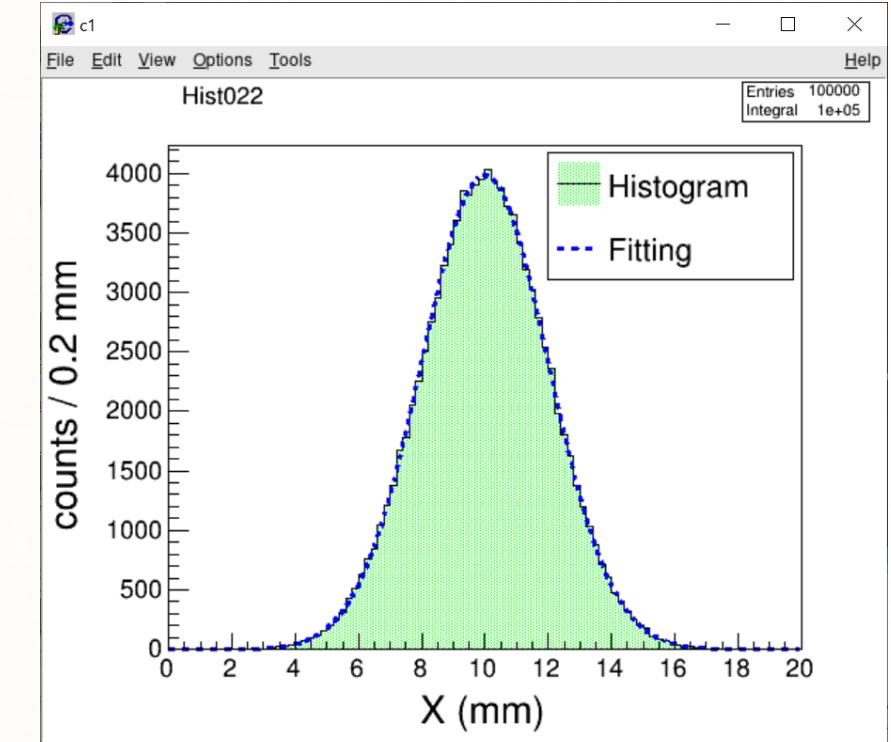
```
TLegend *leg = new TLegend(0.60,0.70,0.89,0.89,"","NDC");
leg->SetFillColor(0);
leg->SetTextSize(0.05);
leg->SetTextFont(42);
leg->SetNColumns(1);
leg->AddEntry(h1,"Histogram","pfl");
leg->AddEntry(f,"Fitting","pl");

TCanvas *c1 = new TCanvas("c1","c1",600,500);
c1->Divide(1,1,1E-5,1E-5);
c1->cd(1)->SetMargin(0.15,0.10,0.15,0.10);
h1->Draw();
leg->Draw("same");
```

レジェンドを用意
背景色なし
テキスト設定

列数は1
レジェンド記入、点・フィル・ライン有
点・ラインのみ

レジェンド描画



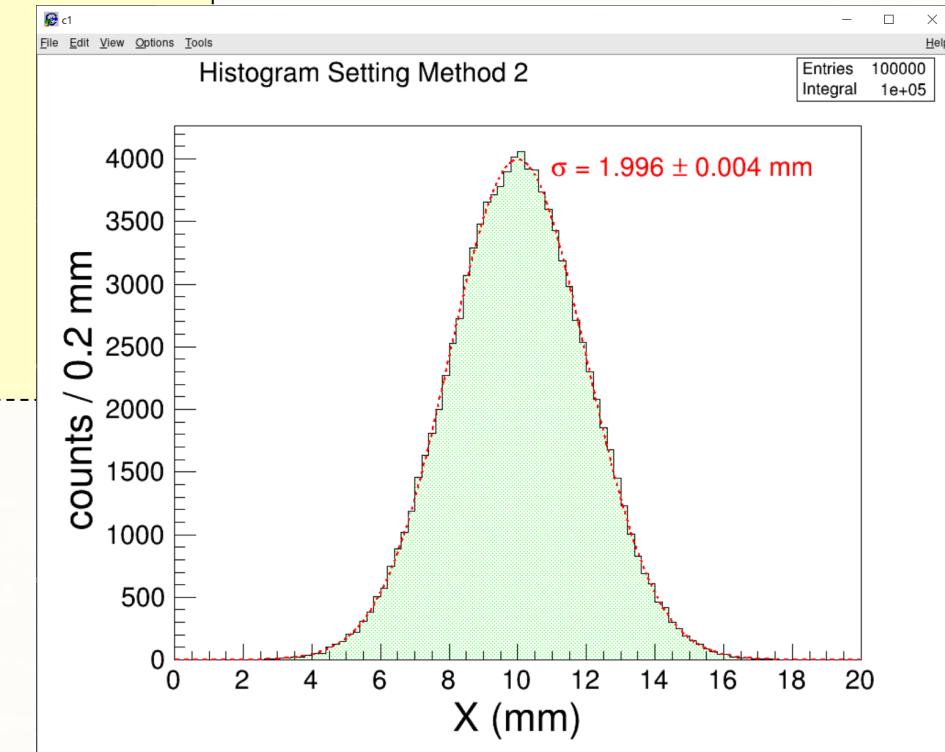
ヒストグラム設定

ex023_HistSetting2.cc

```
void ex023(double mean = 10., double wid = 2.){  
    TF1 *f = new TF1("f","gaus",min_x,max_x);  
    f->SetParameters(10,mean,wid);  
    SetTF1(f,2,2,7);  
  
    TH1D *h1 = new TH1D("h1","h1",100,min_x,max_x); // Set titles with special function  
    SetTH1(h1,"Histogram Setting Method 2","X (mm)",Form("counts / %.1lf mm", (max_x-min_x)/100.),1,3003,3);  
  
    TCanvas *c1 = new TCanvas("c1","c1",1000,800);  
    c1->Divide(1,1,1E-5,1E-5);  
    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.10);  
    h1->Draw();  
    f->Draw("same");  
    t->Draw();  
}
```

詳細は `~/.rootmacro.cc` の `SetTH1` 関数を参照

これだけで設定可能



ヒストグラム計算

```
void ex024(){
    TH1D *h[6];
    for(int i=0;i<6;i++){
        h[i] = new TH1D(Form("h%d",i+1),Form("h%d",i+1),100,-10.,10.);
        SetTH1(h[i],Form("%s",title[i].data()),"X","Counts",1,3003,color[i]);
    }

    h[2]->Add(h[0]);
    h[2]->Add(h[1]);
    h[3] = (TH1D*)h[0]->Clone("h4");
    h[3]->Scale(1./10.);
    h[4]->Multiply(h[0],h[1],1.,1.);
    h[5]->Divide(h[0],h[3],1.,5.);

    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.10);
    h[0]->Draw();
    c1->cd(2)->SetMargin(0.15,0.10,0.15,0.10);
    h[1]->Draw();
    c1->cd(3)->SetMargin(0.15,0.10,0.15,0.10);
    h[2]->Draw("E");
    h[0]->Draw("same");
    h[1]->Draw("same");
    c1->cd(4)->SetMargin(0.15,0.10,0.15,0.10);
    h[3]->Draw("HIST");
    c1->cd(5)->SetMargin(0.15,0.10,0.15,0.10);
    h[4]->Draw("");
    c1->cd(6)->SetMargin(0.15,0.10,0.15,0.10);
    h[5]->Draw("");
}
```

ex024_HistCalc.cc

ヒストグラム配列を用意

ヒストグラム配列の設定

描画オプションはリファレンスで確認可

$$h[2] = h[0] + h[1]$$

$h[3]$ を $h[0]$ のクローンとして用意

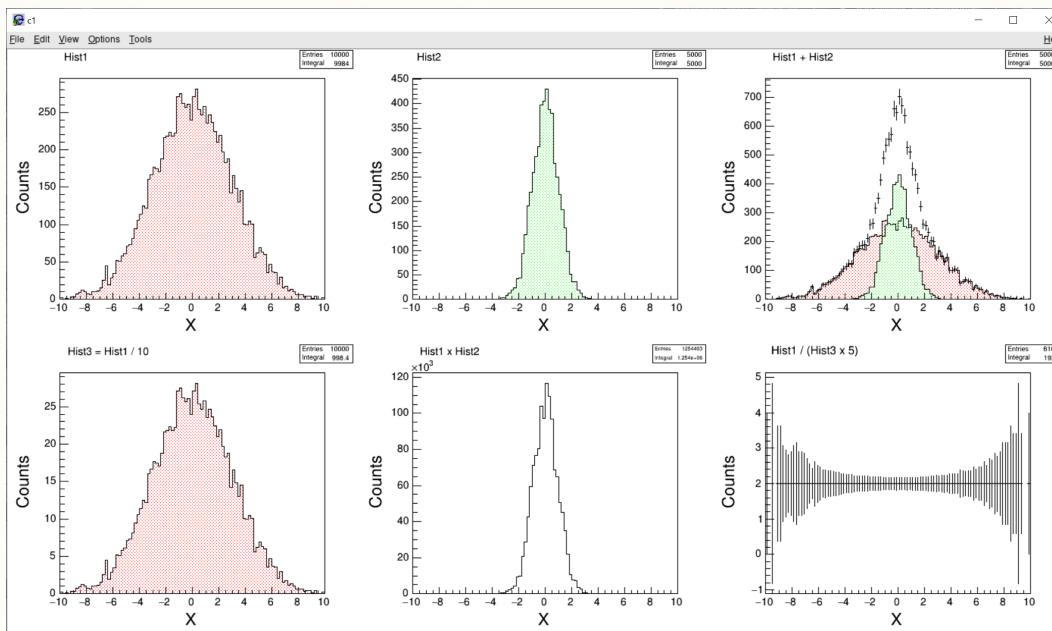
$h[3]$ の高さを 1/10 に

$$h[4] = (1 \times h[0]) \times (1 \times h[1])$$

$$h[5] = (1 \times h[0]) / (5 \times h[3])$$

エラーバー付で描画

ヒストグラムとして描画



2Dヒストグラム

ex025_2DHist.cc

```
void ex025(){
    TH2D *h21 = new TH2D("h21","title",20,-10.,10.,50,0.,20.);
    TH2D *h22, *h23;

    for(int i=0;i<100000;i++){
        double xval = gRandom->Uniform(-9., 9.);
        double yval = gRandom->Landau(4., 1.);
        h21->Fill(xval, yval);
    }
    h22 = (TH2D*)h21->Clone("h22");
    h23 = (TH2D*)h21->Clone("h23");

    SetTH2(h21,"scat option" , "X", "Y",0.0);

    TCanvas *c1 = new TCanvas("c1","c1",1000,800);
    c1->Divide(2,2,1E-5,1E-5);
    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.10);
    h21->Draw("scat");
    c1->cd(2)->SetMargin(0.15,0.10,0.15,0.10);
    h22->Draw("colz");
    c1->cd(3)->SetMargin(0.15,0.10,0.15,0.10);
    h23->Draw("lego2z");
    gPad->SetLogz();
    gPad->SetTheta(30.); gPad->SetPhi(120.);
    c1->cd(4)->SetMargin(0.15,0.10,0.15,0.10);
    h21->ProjectionY()->Draw("");
    h21->ProjectionY()->SetTitle("Y Projection");
}
```

2Dヒストグラムを用意

x軸は -9 ~ 9 まで一様に
y軸は MPV=4, 幅1のランダウ分布
2Dヒストをフィル

ついでにコピーを用意

ヒストグラム設定

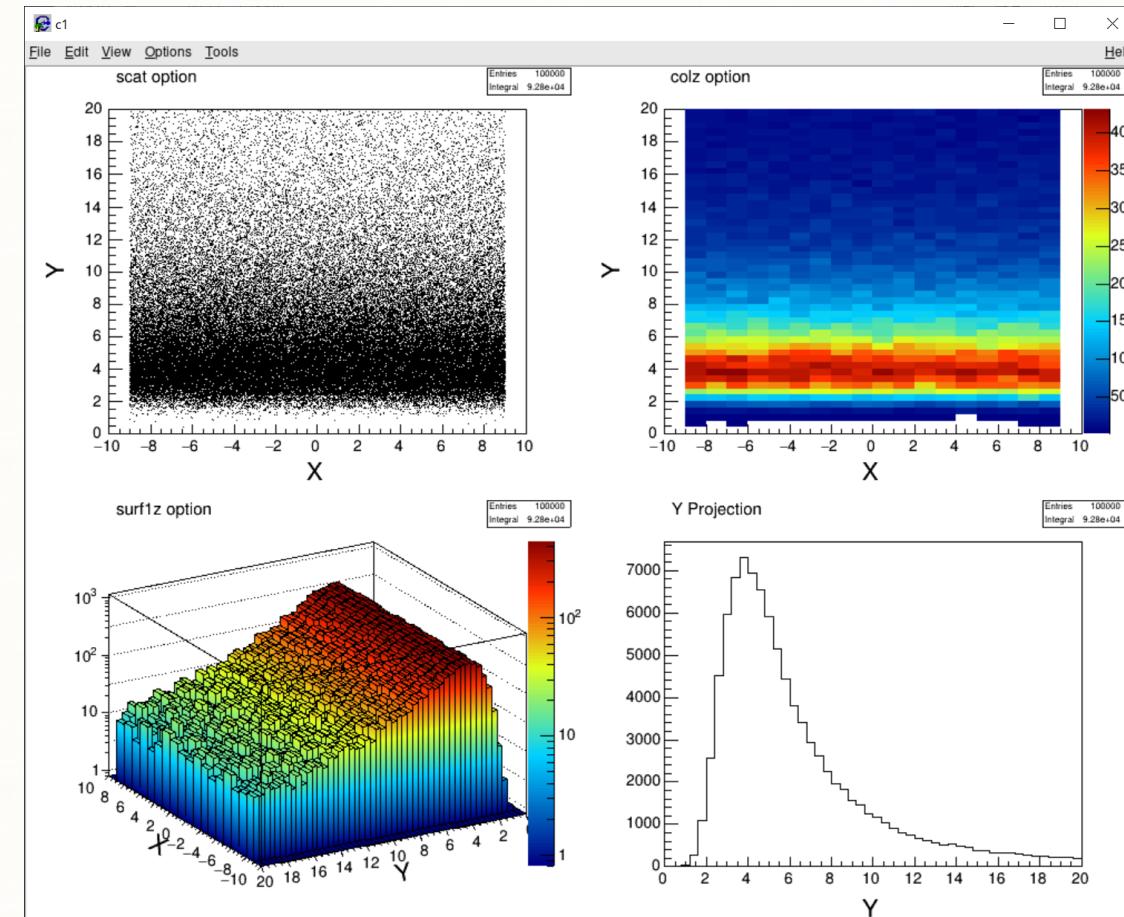
h21 は Scatter Plot で

h22 は カラフルに

h23 は レゴブロックで
Z軸を対数表示
俯瞰角度設定

h21 を Y 方向プロジェクション

描画オプションは [リファレンス](#) で確認可



プロファイル

ex026_Profile.cc

```
void ex026a(){
    TH2D *h2 = new TH2D("h2","title",100,0.,10.,100,-10.,10.);
    TProfile *pr = new TProfile("pr","Profile",200,0.,10,-10.,10.);
    pr->SetLineWidth(4);

    for(int i=0;i<4000;i++){
        double xval = gRandom->Landau(4., 1.5);
        double yval = 5. / sqrt(xval + 1.) + gRandom->Gaus(0., 1.);
        h2->Fill(xval, yval);
        pr->Fill(xval, yval);
    }

    TF1 *f1 = new TF1("f1","[0]/sqrt(x+[1])",0.,10.);
    SetTF1(f1,2,2);
    pr->Fit(f1,"0","",0.,10.);
    double par[2], epar[2];
    f1->GetParameters(&par[0]);
    epar[0] = f1->GetParError(0);
    epar[1] = f1->GetParError(1);

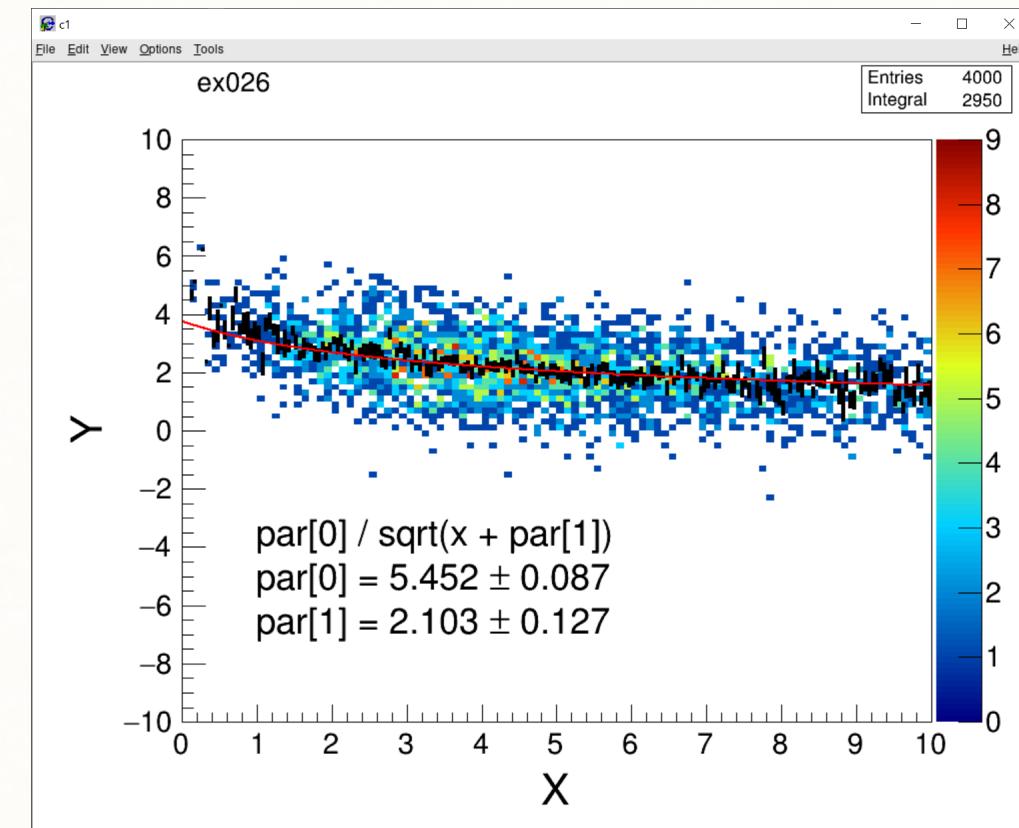
    h2->Draw("colz");
    pr->Draw("same");
    f1->Draw("same");
    TLatex text;
    text.DrawLatex(1.,-4.0,"par[0] / sqrt(x + par[1])");
    text.DrawLatex(1.,-5.5,Form("par[0] = %.3lf #pm %.3lf",par[0],epar[0]));
    text.DrawLatex(1.,-7.0,Form("par[1] = %.3lf #pm %.3lf",par[1],epar[1]));
}
```

2Dヒストグラムを用意
プロファイルを用意

X軸はランダウ分布
Y軸は $1/\sqrt{x}$ で
ヒストにフィル
プロファイルにもフィル

$1/\sqrt{x}$ 関数を用意
フィット
パラメータをゲット
誤差結果をゲット

描画
テキスト描画



フィットスライス

ex026_Profile.cc

```
void ex026b(){
    gErrorIgnoreLevel = kWarning;
    TH2D *h2 = new TH2D("h2","title",100,0.,10.,100,-10.,10.);
    for(int i=0;i<4000;i++){
        double xval = gRandom->Landau(4., 1.5);
        double yval = 5. / sqrt(xval + 1.) + gRandom->Gaus(0., 1.);
        h2->Fill(xval, yval);
    }
    TF1 *f1 = new TF1("f1","gaus",-10.,10.);
    h2->FitSlicesY(f1,1,100,0, "QNRG2");
    TH1D *h1 = (TH1D*)gROOT->FindObject("h2_1");
    h1->SetLineWidth(3);

    TF1 *f2 = new TF1("f2","[0]/sqrt(x+[1])",0.,10.);
    SetTF1(f2,2,2);
    h1->Fit(f2,"0","",0.,10.);
    double par[2], epar[2];
    f2->GetParameters(&par[0]);
    epar[0] = f2->GetParError(0);
    epar[1] = f2->GetParError(1);

    TCanvas *c1 = new TCanvas("c1","c1",1000,800);
    c1->Divide(1,1,1E-5,1E-5);
    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.10);
    h2->Draw("colz");
    h1->Draw("same");
    f2->Draw("same");
}
```

verbose設定
2Dヒストグラムを用意

X軸はランダウ分布
Y軸は $1/\sqrt{x}$ で
ヒストにフィル

スライス分布のフィット関数を用意
2DヒストをY方向にスライスフィット
スライスフィットの結果をゲット

スライスフィット分布用のフィット関数

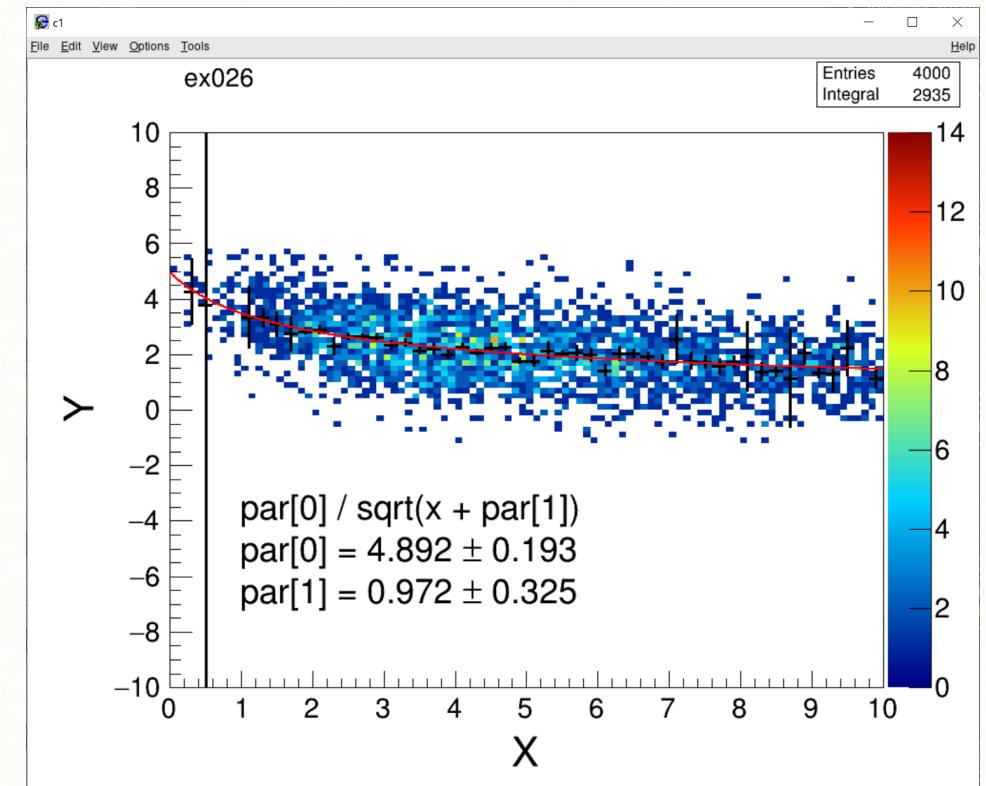
フィット

パラメータをゲット
誤差結果をゲット

描画

FitSlices Options

"Q" means Quiet mode
"N" means do not show the result of the fit
"R" means fit the function in the specified function range
"G2" merge 2 consecutive bins along X
"G3" merge 3 consecutive bins along X
"G4" merge 4 consecutive bins along X
"G5" merge 5 consecutive bins along X
"S" sliding merge:



1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

フィット

ex021_Hist.cc

GUIを使ってフィットする

ex023_HistSetting2.cc

関数を用意してフィットにあてる

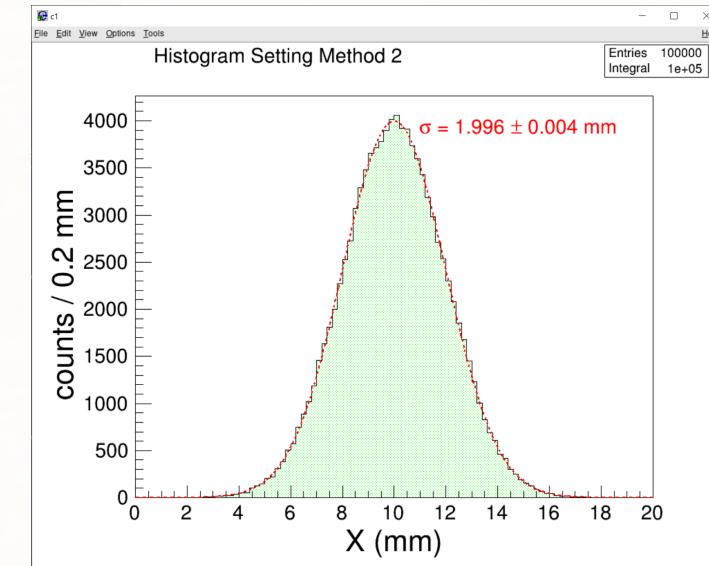
TF1 (const char *name, const char *formula, Double_t xmin, Double_t xmax, Option_t *option)

```
TF1 *f = new TF1("f","gaus",min_x,max_x);
f->SetParameters(10,mean,wid);
SetTF1(f,2,2,7);

double par[3], epar;
h1->Fit(f,"0Q","",min_x,max_x);
f->GetParameters(&par[0]);
epar = f->GetParError(2);
cout<<Form("Constant : %e",par[0])<<endl;
cout<<Form("Mean   : %lf",par[1])<<endl;
cout<<Form("Sigma  : %.3lf",par[2])<<endl;
```

フィット関数の用意
初期値の代入（ピーク高さ、中心値、幅）
関数設定

フィットを実行
フィット結果をゲット
誤差結果をゲット
標準出力する



TF1 Class

標準の関数

```
f = new TF1("f", "gaus", 0, 10)
```

gaus : ガウス分布

gausn : 規格化したガウス分布

expo : 指数関数

landau : ランダウ分布

landau : 規格化したランダウ分布

pol0 ~ 9 : 多項式

cheb0 ~ 9 : チェビシェフ多項式

TMath登録済みの関数

```
f = new TF1("f", "[0]*TMath::Sin([1]*x)", 0., 1.);
```

ACos	Max
ACosH	Min
ASin	Na
ASinH	NaUncertainty
ATan	NextPrime
ATan2	NormQuantile
ATanH	Normalize
Abs	Odd
AreEqualAbs	Permute
AreEqualRel	Pi
Bessel	PiOver2
Bessel0	PiOver4
Bessel1	Poisson
BesselJ0	Poissonl
BesselJ1	Power
BesselK	Prob
BesselK0	Qe
BesselK1	QeUncertainty
BesselY0	Quantiles
BesselY1	QuietNaN
Beta	R
BetaCf	RUncertainty
BetaDist	RadToDeg
BetaDistI	Range
BetaIncomplete	Rgair
Binomial	RootsCubic
BinomialI	Sigma
BreitWigner	SigmaUncertainty
BubbleHigh	Sign
BubbleLow	SignBit
C	SignalingNaN
CUncertainty	Sin
CauchyDist	Sinh
Ccgs	Sq
Ceil	Sqrt
CeilNint	Sqrt2
ChisquareQuantile	StruveH0
Cos	StruveH1
Cosh	StruveL0
DegToRad	StruveL1
DiLog	Student
E	Studentl
Erf	StudentQuantile
ErfInverse	Tan
Erfc	Tanh
ErfcInverse	TwoPi
EulerGamma	Vavilov
Even	Vavilovl
Exp	Voigt

自分で記述

```
f = new TF1("f", "[0]*sin(x)/x", 0, 1);
```

TF1 Class

自分で記述 part2 (PoissonとGausの畠み込み関数)

ex027_PoisGaus.cc

```
void ex027(){
    auto f = new TF1("f",Pois,0.,20.,3);   フィット関数用意、最後の"3"はパラメータ数
    SetTF1(f,2,2,1);
    f->SetParameter(0,6.);
    f->SetParameter(1,1.);
    f->SetParameter(2,100.);

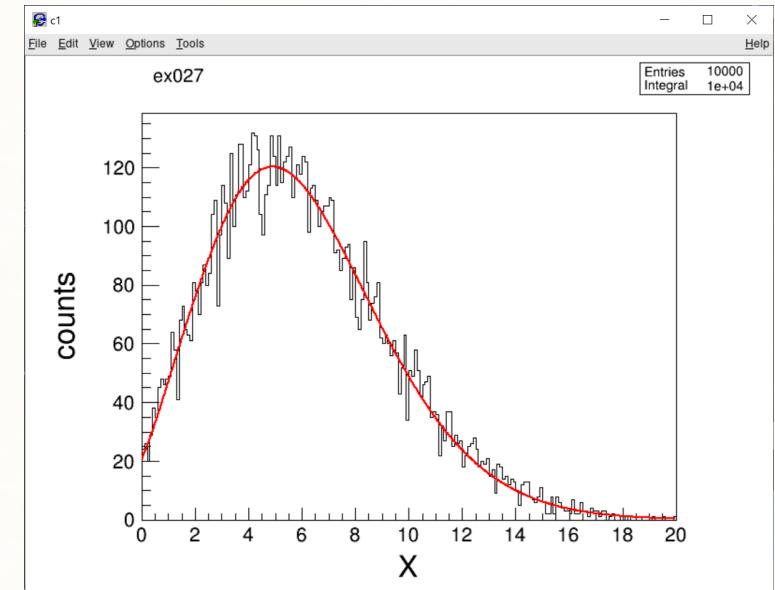
    auto h1 = new TH1D("h1","ex027",200,0.,20.);
    SetTH1(h1,"ex027","X","counts");
    h1->FillRandom("f",10000);           関数形にあわせてランダム生成

    h1->Fit(f,"","",0.,20.);          フィット

    TCanvas *c1 = new TCanvas("c1","c1",800,600);
    c1->Divide(1,1);
    c1->cd(1);
    h1->Draw();
    f->Draw("same");
}
```

~/.rootmacro.C

```
double Pois( double *x, double *par )
{
    // par[0] : lambda, average of #photon
    // par[1] : energy resolution factor
    // par[2] : menseki
    double val = 0;
    for( int np=1; np<500; np++ ){
        double pois;
        double sigma = par[1]*sqrt(np);
        if(np<50){
            pois = par[2]*pow( par[0], np )*exp(-par[0])/TMath::Gamma(np+1);
        }
        else{ // stirling's approximation
            pois = par[2]*pow( par[0]/np, np )*exp(-par[0]+np)/(sqrt(2.*PI)*pow(np,0.5));
        }
        val += pois/(sqrt(2.*PI)*sigma)*exp( -pow(x[0]-np,2)/2./sigma/sigma );
    }
    return val;
}
```



フィットオプション

```
h1->Fit(f, "", "", 0, 1);
```

左から、フィット関数、オプション、グラフィックオプション、範囲最小、範囲最大

“W”	：0ではない bin すべてに重み 1 を付与してフィット	(統計誤差無視)
“WW”	：0カウント bin も含めて重み 1 を付与してフィット	(統計誤差無視)
“I”	：bin の中央値カウントではなく面積を使ってフィット	(分布に対して binning が太い場合)
“L”	：最尤法を利用してフィット	(bin の統計が少なく、Log Likelihood を利用するべき場合)
“WL”	：加重尤度法を利用してフィット	(bin の統計が少なく、かつ各 bin の誤差が Poisson 分布に従わない場合)
“P”	：bin の誤差として Poisson 分布標準偏差を利用しない	(Poisson のカイ二乗検定を利用したい場合)
“Q”	：静穏モード	(うるさい場合)
“V”	：冗長モード	(詳細を知りたい場合)
“S”	：フィット結果を TFitResultPtr に保存する。	
“E”	：Minosを使ったより良い誤差推定	(非対称誤差)
“M”	：TMinuit の IMPROVE を使ったフィット結果の改良	
“R”	：関数レンジを使ってフィット	
“N”	：グラフィックス関数を保存しない、表示しない	
“0”	：フィット結果をプロットしない	
“+”	：フィットした関数リストに新しい関数を追加する	
“B”	：事前フィットの結果の 1 もしくはそれ以上パラメータを固定したい	
“C”	：リニアフィットの場合、カイ二乗を計算しない (時間節約)	
“F”	：多項式の場合、最小化の方法として Minuit を採用する	

1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)**
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

ROOTファイルの書き込み

ex031_Write.cc

```
void ex031(){
    TFile *ofp = new TFile("ex031.root","RECREATE");

    TH1D *h1 = new TH1D("h1","title",100,-10,10);
    SetTH1(h1,"ex031 h1","x","y");

    ofp->mkdir("aaa");
    ofp->cd("aaa");
    TH1D *h2 = new TH1D("h2","title",100,-5,15);
    SetTH1(h2,"ex031 h2","x","y");

    gDirectory->cd("/");
    gDirectory->mkdir("bbb");
    gDirectory->cd("bbb");
    TH1D *h3 = new TH1D("h3","title",100,-5,15);
    SetTH1(h3,"ex031 h3","x","y");

    h1->FillRandom("gaus",10000);
    h2->FillRandom("landau",10000);
    h3->FillRandom("gaus",1000);

    ofp->Write();
    ofp->Close();
}
```

保存用ファイルの新規作成

h1を作成

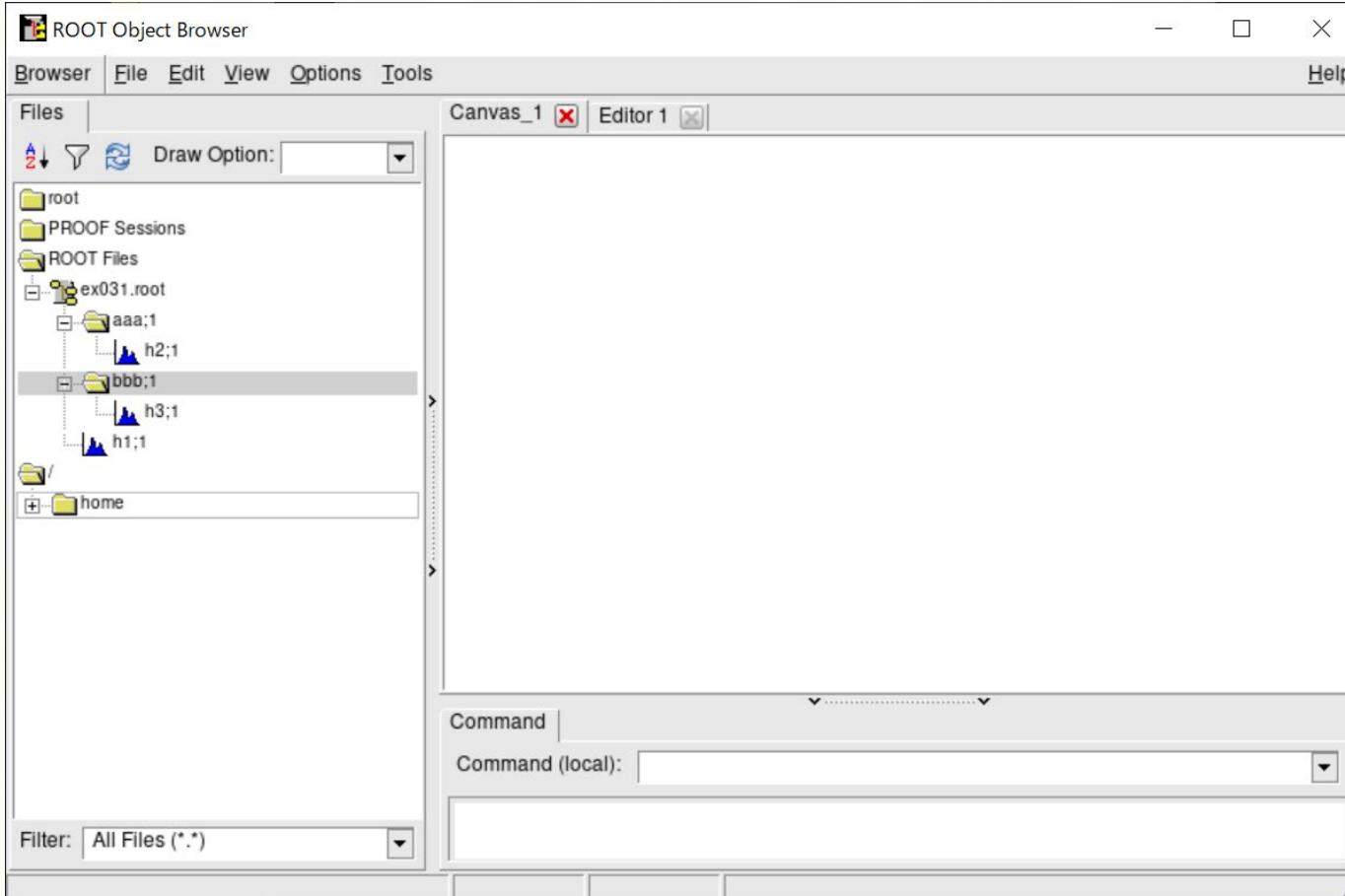
aaa ディレクトリを作成
aaa ディレクトリに移動
h2 を作成

最上位ディレクトリに移動
bbb ディレクトリを作成
bbb ディレクトリに移動
h3 を作成

書き込み
ファイルを閉じる

ファイルの開き方

```
$ root ex031.root  
[0] .ls  
[1] .pwd  
[2] h1->Draw();  
[3] TBrowser b
```



ROOTファイルの読み込み

ex032_Read.cc

```
void ex032(){
    TFile *ifp = new TFile("ex031.root");

    TH1D *h01 = (TH1D*)ifp->Get("h1");
    gDirectory->cd("aaa");
    TH1D *h02 = (TH1D*)gROOT->FindObject("h2");
    gDirectory->cd("../bbb");
    TH1D *h03 = (TH1D*)gROOT->FindObject("h3");

    TCanvas *c1 = new TCanvas("c1","c1",800,600);
    c1->Divide(2,2);
    c1->cd(1);
    h01->Draw();
    c1->cd(2);
    h02->Draw();
    c1->cd(3);
    h03->Draw();
}
```

ファイルを開く

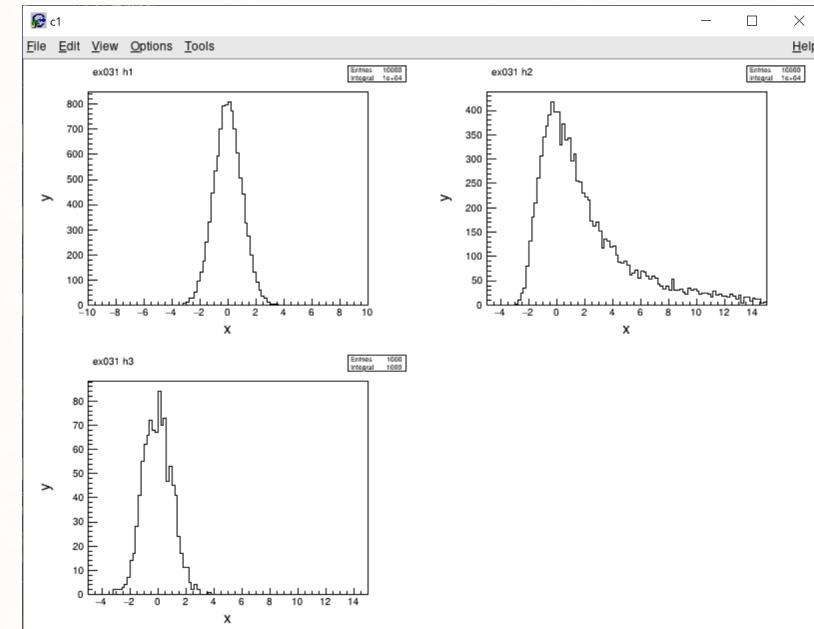
h1 を読み込む

aaa ディレクトリに移動

h2 を読み込む

bbb ディレクトリに移動

h3 を読み込む



1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ ([TGraph](#))
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

グラフ

ex041_Graph.cc

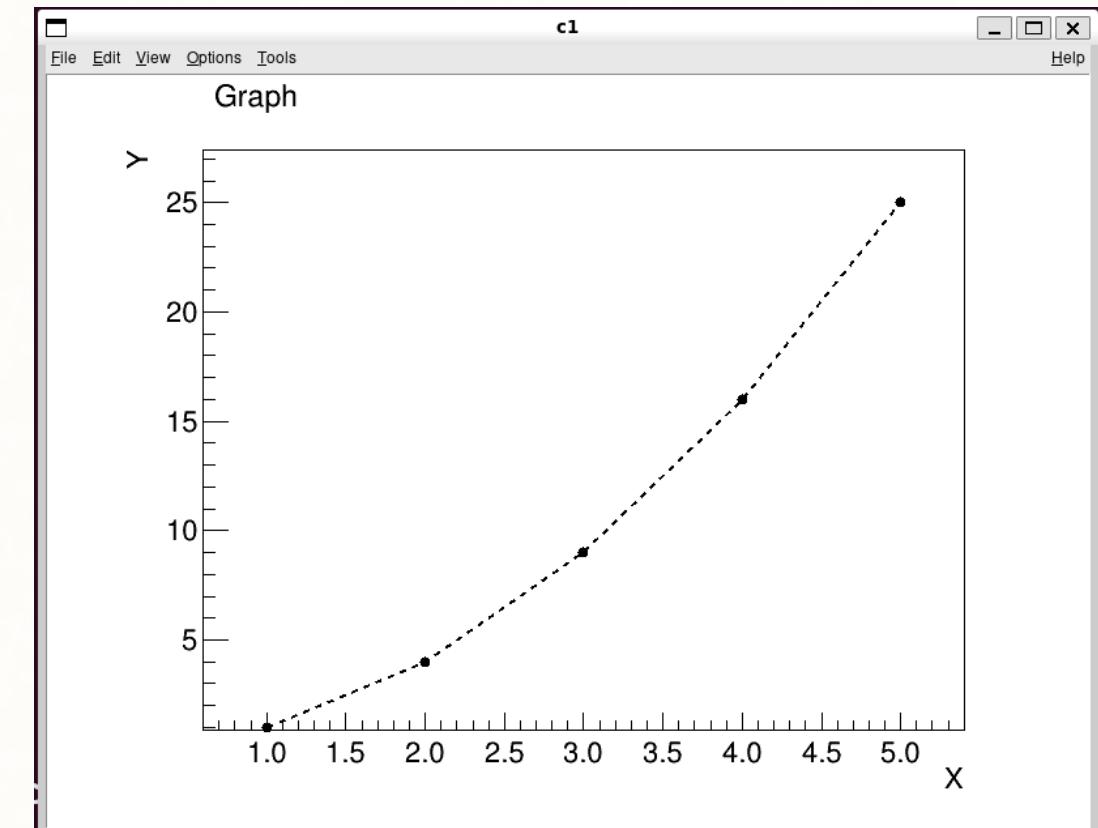
```
void ex041a(){
    double x[5] = {1, 2, 3, 4, 5};
    double y[5] = {1, 4, 9, 16, 25};
    TGraph *g = new TGraph(5, x, y);
    g->SetMarkerSize(1.0);
    g->SetMarkerStyle(20);
    g->SetLineWidth(2);
    g->SetLineColor(1);
    g->SetLineStyle(7);
    g->GetXaxis()->SetTitle("X");
    g->GetYaxis()->SetTitle("Y");
    g->SetTitle("Graph");

    TCanvas *c1 = new TCanvas("c1","c1",800,600);
    c1->cd(1);
    g->Draw("APL");
}
```

変数を用意

グラフを作成
マーカーサイズ指定
マーカースタイル指定
線幅指定
線色指定（黒）
線スタイル（Dashed）
タイトル指定

描画



グラフ

ex041_Graph.cc

```
void ex041b(){
    vector<double> x, y;
    for(int i=0;i<5;i++){
        x.push_back(i);
        y.push_back(i*i);
    }
    TGraph *g = new TGraph(x.size(), &x[0], &y[0]);
    SetGr(g,"Graph","X","Y",1,2,7,1,20,1.);

    g->Draw("APC");
}
```

```
void ex041c(){
    double x[5] = {1, 2, 3, 4, 5};
    double y[5] = {1, 4, 9, 16, 25};
    TGraph *g = new TGraph(5, x, y);
    SetGr(g,"","","","",1,2,7,1,20,1.);

    TH2D *h2 = new TH2D("h2","h2",1000,0,6,1000,0,30);
    SetTH2(h2,"Graph","X","Y",0.0);
    h2->SetStats(kFALSE);

    h2->Draw();
    g->Draw("samePL");
}
```

変数を用意
変数を詰める
グラフを作成
グラフ設定

グラフ描画（カーブで結ぶ）

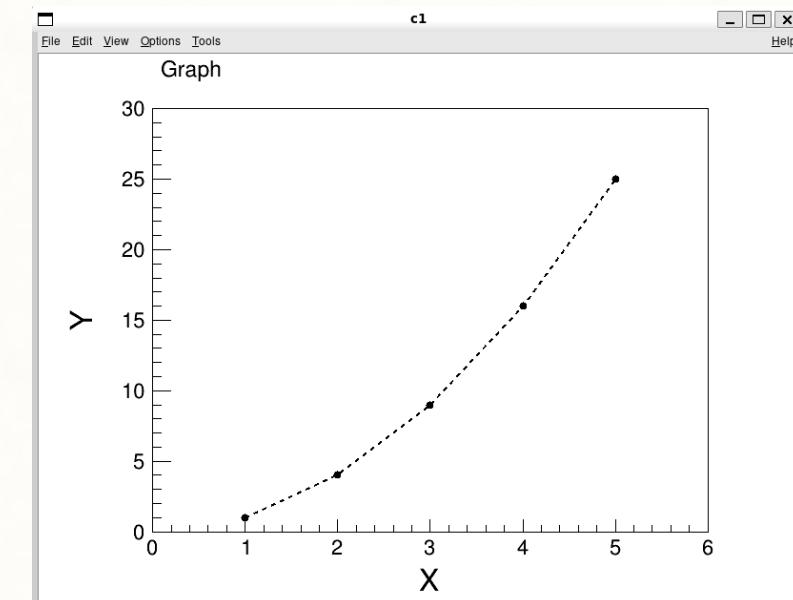
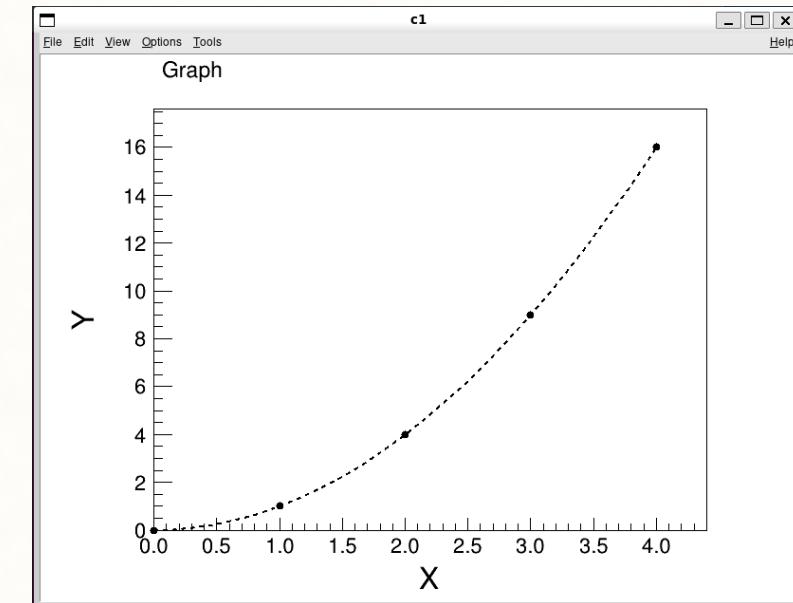
きれいなグラフを作るテク

変数を用意

変数を詰める

空の2Dヒストを用意

ヒスト描画
グラフ描画



グラフ

ex041_Graph.cc

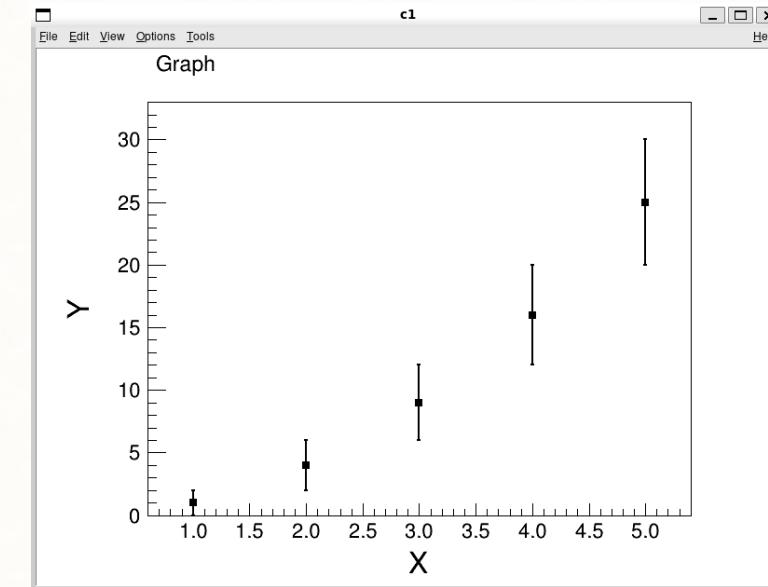
```
void ex041d(){
    double x[5] = {1, 2, 3, 4, 5};
    double y[5] = {1, 4, 9, 16, 25};
    double ey[5] = {1, 2, 3, 4, 5};
    TGraphErrors *g = new TGraphErrors(5, x, y, 0, ey);
    SetGr(g,"Graph","X","Y",1,2,1,1,21,1.);

    TCanvas *c1 = new TCanvas("c1","c1",800,600);
    c1->cd(1);
    g->Draw("AP");
}
```

エラーバー付きグラフ
変数を用意

エラー量を用意
グラフを作成
グラフ設定

グラフ描画



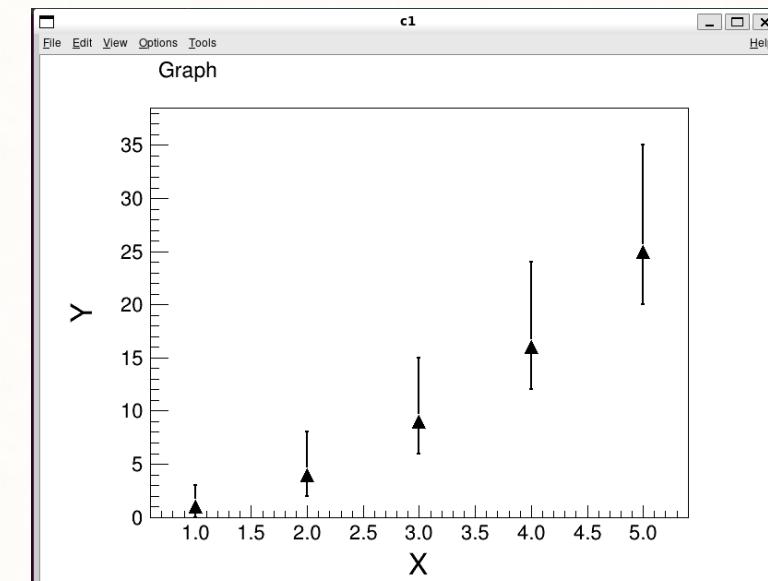
```
void ex041e(){
    double x[5] = {1, 2, 3, 4, 5};
    double y[5] = {1, 4, 9, 16, 25};
    double eyl[5] = {1, 2, 3, 4, 5};
    double eyh[5] = {2, 4, 6, 8, 10};
    auto *g = new TGraphAsymmErrors(5, x, y, 0, 0, eyl, eyh);
    SetGr(g,"Graph","X","Y",1,2,1,1,22,2.);

    TCanvas *c1 = new TCanvas("c1","c1",800,600);
    c1->cd(1);
    g->Draw("AP");
}
```

非対称エラーバー付きグラフ
変数を用意

エラー量を用意
グラフを作成
グラフ設定

グラフ描画



グラフ

ex041_Graph.cc

```
void ex041f(){
    TGraph *g = new TGraph("data.txt","%lg %lg","","");
    SetGr(g,"Graph","X","Y",1,2,1,1,24,2.);

    TCanvas *c1 = new TCanvas("c1","c1",800,600);
    c1->cd(1);
    g->Draw("APL");
}
```

data.txt の中身

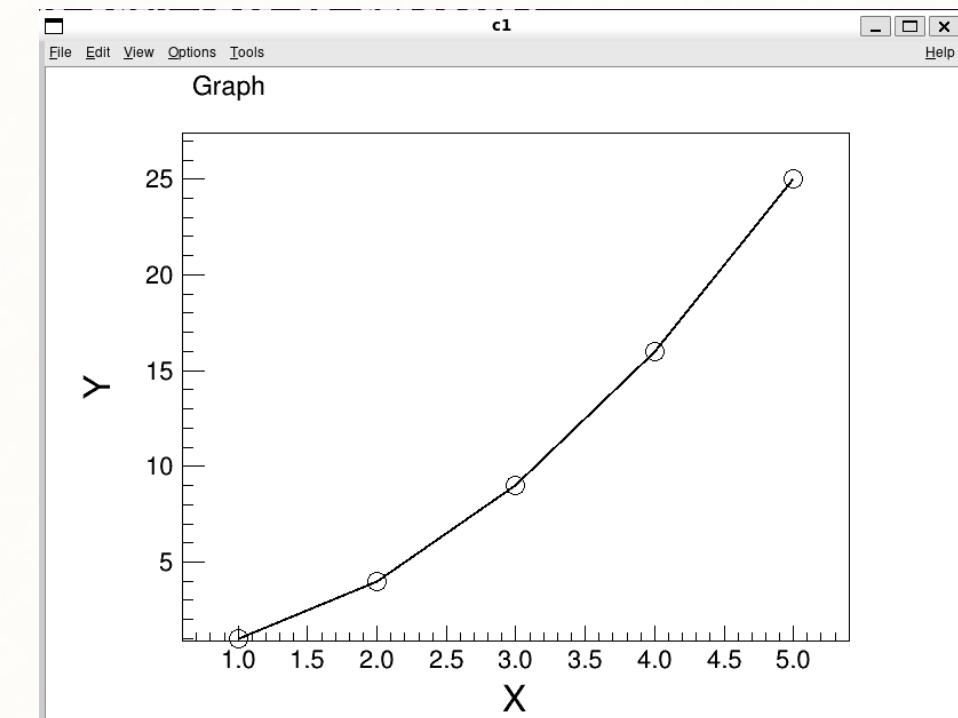
```
1, 1
2, 4
3, 9
4, 16
5, 25
```

ファイルからグラフ生成

グラフを作成、Delimiterは "," を指定
グラフ設定

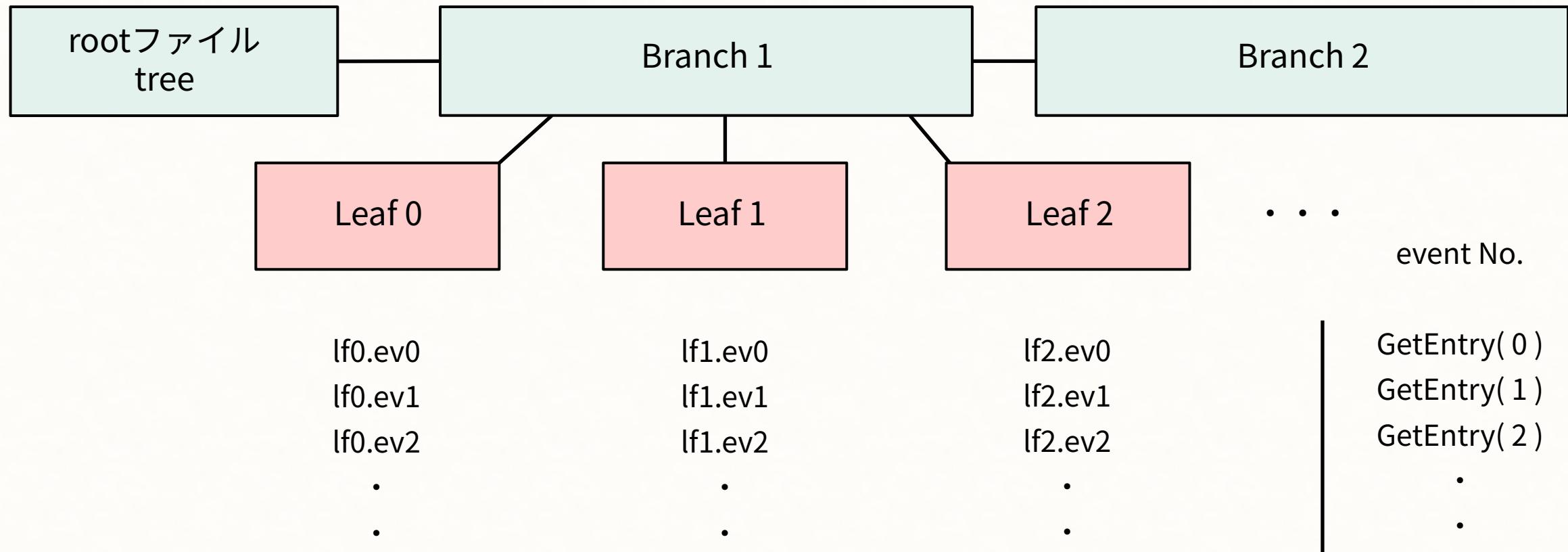
グラフ描画

普通なら、Getline() でループを回すが、、、
csvファイル読み込みなどに使える



1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

Treeという考え方 (Ntuple)



実験では1事象で様々な変数を取得する
変数を事象毎にパッケージ化しておけば、変数間の相関などを確認でき
るようになり、解析が容易
TTree クラスが用意されている

Treeの作り方

ex051_MakeTree.cc

```
typedef struct{ double dval1, dval2; int ival1; } TQ;

void ex051(){
    TFile *ofp = new TFile("tree.root","recreate");
    TTree *tree = new TTree("tree","tree");
    int ival; char cval[100]; float fval; double dval; double darray[2];
    vector<double> vdval;
    TQ tqval;

    tree->Branch("ival" ,&ival , "ival/I" );
    tree->Branch("cval" ,&cval , "cval/C" );
    tree->Branch("fval" ,&fval , "fval/F" );
    tree->Branch("dval" ,&dval , "dval/D" );
    tree->Branch("darray", darray, "darray[2]/D" );
    tree->Branch("vdval" ,&vdval );
    tree->Branch("tqval" ,&tqval , "dval1/D:dval2:ival1/I" );

    string sval[4] = {"Nameko", "Eringi", "Kare", "Masaru"};
    for(int i=0;i<10000;i++){
        ival = (int)gRandom->Gaus(10.,2.);
        strcpy(cval, sval[gRandom->Integer(4)].c_str());
        fval = (float)gRandom->Gaus(0.,10.);
        dval = gRandom->Gaus(0.,10.);
        darray[0] = gRandom->Gaus(0.,1.);
        darray[1] = gRandom->Gaus(0.,10.);
        for(int j=0;j<10;j++){ vdval.push_back( gRandom->Poisson(j+1) ); }
        tqval.dval1 = gRandom->Gaus(0.,10.);
        tqval.dval2 = gRandom->Gaus(10.,1.);
        tqval.ival1 = gRandom->Poisson(4);
        tree->Fill();
        vdval.clear();
    }
    tree->Write(); ofp->Close();
}
```

構造体変数の用意

出力用ファイルの作成
ツリーの作成
変数定義

int型のBranch定義
char型
float型
double型
配列
動的配列
構造体

変数を詰める

treeをファイルする
vectorはクリアしておく

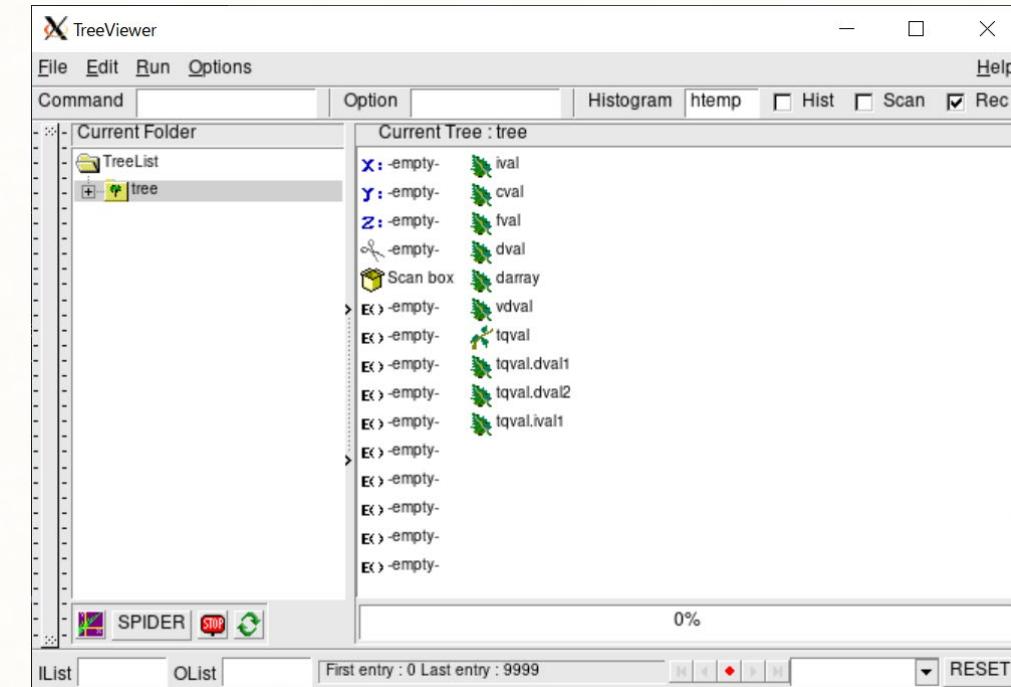
書き込む

Treeの使い方

```
$ root tree.root
```

```
[0] .ls  
[0] tree->Print()  
[0] tree->StartViewer()  
[0] tree->Scan("ival");  
[0] tree->Scan("fval:dval");  
[0] tree->Draw("dval");  
[0] tree->Draw("darray[]");  
[0] tree->Draw("vdval[2]");  
[0] tree->Draw("tqval.dval1:tqval.dval2");  
[0] tree->Draw("tqval.dval1:tqval.dval2", "tqval.dval1<0");  
[0] tree->Draw("tqval.dval1:tqval.dval2", "", "colz");  
[0] tree->GetEntries()
```

ファイルのリストを表示
Branchリストを表示
Branchリストをビュワー表示
ival をリスト表示
fval, cval をリスト表示
dval を描画
darray全配列描画
vdval[2] のみ表示
dval1 をYに、 dval2 をX軸にして2D Plot
dval1<0 のカット条件を追加
colz表示に
イベント数を表示

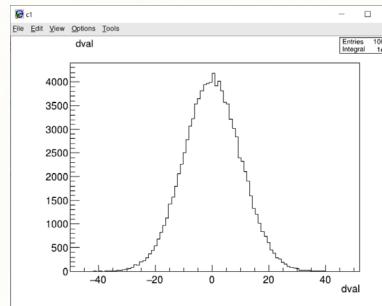


Treeの読み込み方

ex052_ReadTree.cc

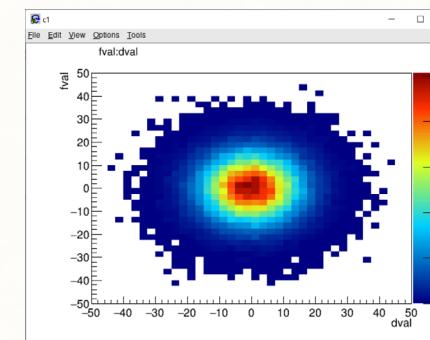
```
void ex052a(){
    TFile *fp = new TFile("tree.root");
    TTree *tree = (TTree*)fp->Get("tree");
    tree->Draw("dval");
}
```

最も単純な方法
ファイルを開く
Treeを読み込む
dval を描画



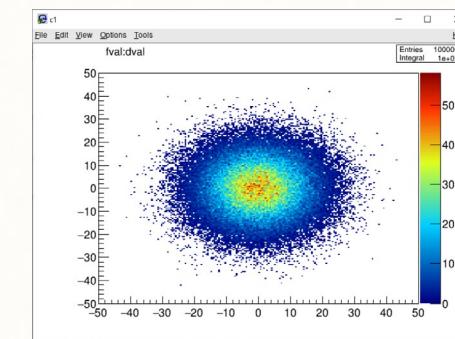
```
void ex052b(){
    TFile *fp = new TFile("tree.root");
    TTree *tree = (TTree*)fp->Get("tree");
    tree->Draw("fval:dval","","colz");
}
```

二次元描画
ファイルを開く
Treeを読み込む
X=dval, Y=fval として描画



```
void ex052c(){
    TFile *fp = new TFile("tree.root");
    TTree *tree = (TTree*)fp->Get("tree");
    tree->Draw("fval:dval>>(200,-50,50,200,-50,50","","colz");
}
```

ヒストグラムに落とし込んで描画
ファイルを開く
Treeを読み込む
X=dval, Y=fval として描画



Treeの読み込み方

ex052_ReadTree.cc

```
void ex052d(){
    TFile *fp = new TFile("tree.root");
    TTree *tree = (TTree*)fp->Get("tree");

    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.07);
    tree->Draw("fval:dval>>h1(200,-50,50,200,-50,50)","","","colz");
    c1->cd(2)->SetMargin(0.15,0.10,0.15,0.07);
    tree->Draw("fval:dval>>h2(50,-50,50,50,-50,50)","","","lego2z");
    c1->cd(3)->SetMargin(0.15,0.10,0.15,0.07);
    tree->Draw("fval:dval>>h3(200,-50,50,200,-50,50)","","","scat");
    c1->cd(4)->SetMargin(0.15,0.10,0.15,0.07);
    tree->Draw("fval:dval>>h4(200,-50,50,200,-50,50)","fval>-10","scat");
}
```

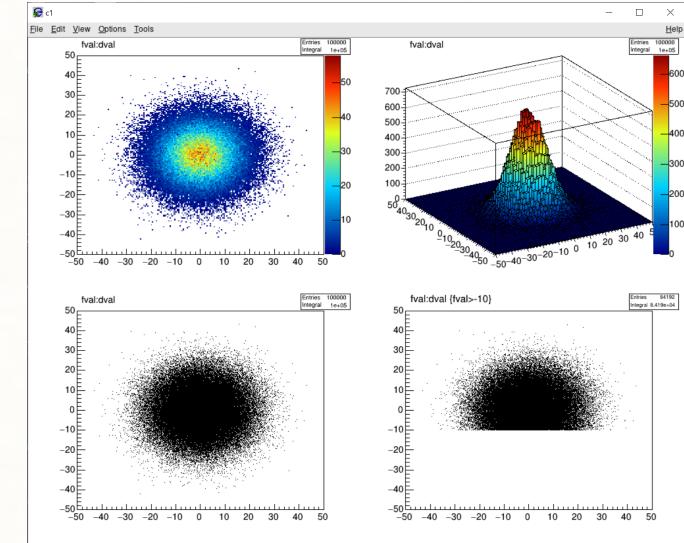
```
void ex052e(){
    TChain *tree = new TChain("tree");
    tree->Add("tree.root");

    TH2D *h = new TH2D("h","h",200,-50,50,200,-50,50);
    tree->Project("h","tqval.dval2:tqval.dval1","tqval.dval1>-20");

    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.07);
    h->Draw("colz");
}
```

ヒストグラムに落とし込んで描画2
ファイルを開く
Treeを読み込む

X=dval, Y=fval として h1 に描画

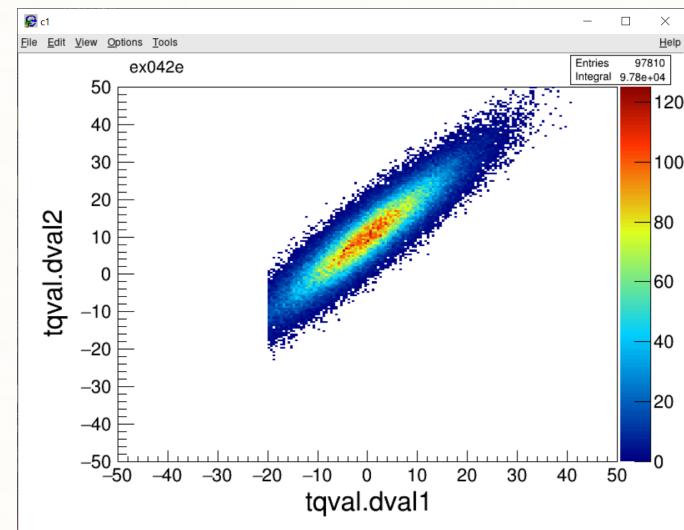


TChainとProject の利用

TChainを定義
treeを追加

ヒストグラムを定義
X=dval1, Y=dval2, カット条件 dval1>-20 として
ヒスト "h" にフィル

hを描画



Treeの読み込み方

ex52_ReadTree.cc

```
typedef struct{ double dval1, dval2; int ival1; } TQ;
void ex052f(){
    TFile *fp = new TFile("tree.root");
    TTree *tree = (TTree*)fp->Get("tree");

    int ival; char cval[100]; float fval; double dval;
    double darray[2]; vector<double> *vdval = 0; TQ tqval;

    tree->SetBranchStatus("*",0);
    tree->SetBranchStatus("ival",1); tree->SetBranchAddress("ival" , &ival );
    tree->SetBranchStatus("cval" ,1); tree->SetBranchAddress("cval" , &cval );
    tree->SetBranchStatus("fval" ,1); tree->SetBranchAddress("fval" , &fval );
    tree->SetBranchStatus("dval" ,1); tree->SetBranchAddress("dval" , &dval );
    tree->SetBranchStatus("darray",1); tree->SetBranchAddress("darray", darray );
    tree->SetBranchStatus("vdval" ,1); tree->SetBranchAddress("vdval" , &vdval );
    tree->SetBranchStatus("tqval" ,1); tree->SetBranchAddress("tqval" , &tqval );

    TH2D *h1 = new TH2D("h1","h1",20,0,20,30,0,30);
    TH2D *h2 = new TH2D("h2","h2",200,-50,50,200,-40,60);

    int ENum = tree->GetEntries();
    for(int n=0;n<ENum;n++){
        tree->GetEntry(n);
        h1->Fill(vdval->at(2), vdval->at(6));
        if( tqval.dval1 > -20 ){ h2->Fill(tqval.dval1, tqval.dval2); }
    }

    c1->cd(1)->SetMargin(0.15,0.10,0.15,0.07);
    h1->Draw("colz"); gPad->SetLogz();
    c1->cd(2)->SetMargin(0.15,0.10,0.15,0.07);
    h2->Draw("colz");
}
```

SetBranchAddressの利用

構造体定義

ファイルを開く
Treeの読み込み

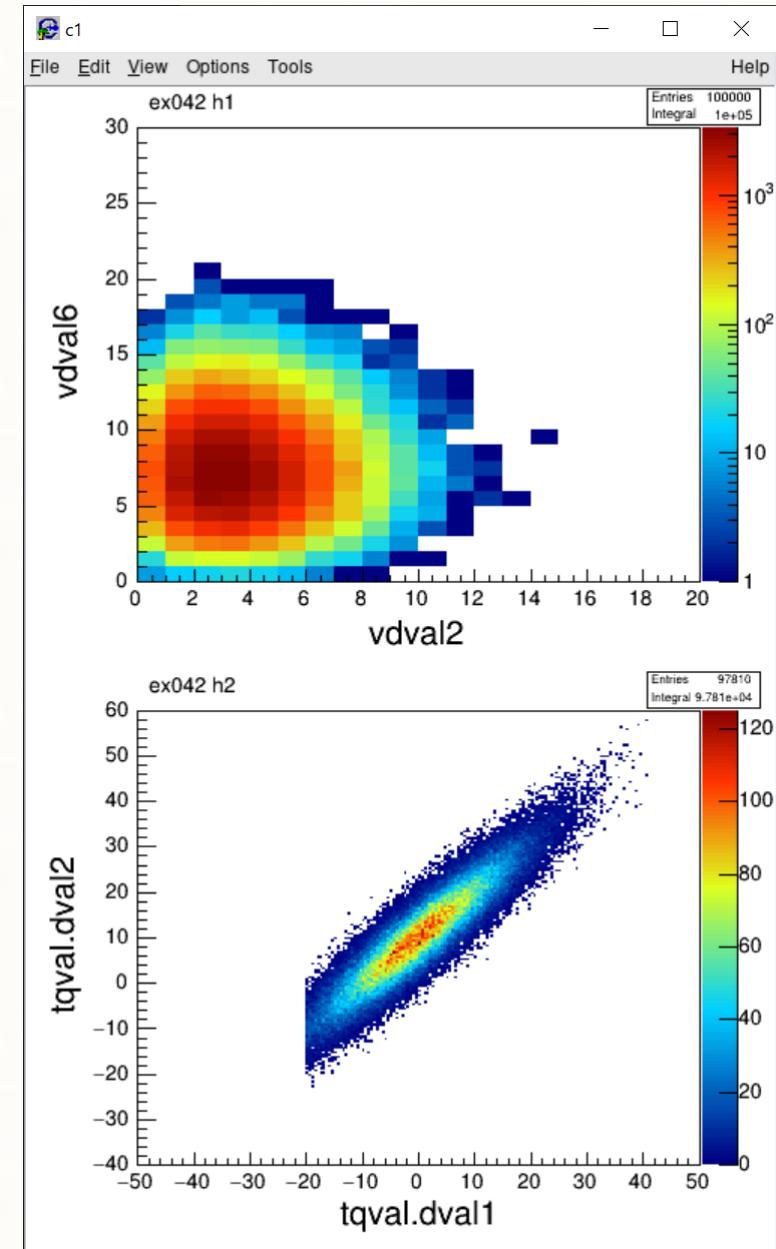
変数定義

全てのブランチを Inactive に
ivalをActive, 変数アドレス ival に Tree ival を登録

配列には & は付かないで注意

tree に入っているイベント総数をゲット
イベント総数でループ
n番目のイベント情報をゲット
フィル
カット条件をかけてフィル

描画



Tree読みの使い分け

	コーディング	汎用性	速度	用途
単純な描画	○	×	×	初期段階の解析 長いコードを書くまでのテスト
ヒストに落とし込んで描画	○	×	×	同上
Project	○	○	×	~10個程度のヒストグラムを作成
SetBranchAddress	×	○	○	複雑なカット条件適用 大量のヒストグラムを作成

TChain + SetBranchAddress + GCCコンパイルは非常に相性が良いので、最終的なデータ解析コードはこれで書く
コーディングに有限の時間がかかるので、解析初期段階では直接コマンドをたたいた方が圧倒的に速い
少しだけ凝った解析をしたいときは Project を使うことが多い

1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 亂数生成 (TRandom)
 - k. 高度なフィット (roofit)

1. はじめに
 2. Operating System
 3. Windows
 4. ROOT, Geant4 導入方法
 5. Linux 環境設定
 6. 鍵認証システム
 7. バージョン管理システム
 8. ROOTの使い方
 9. モンテカルロ法
 10. Geant4の使い方
-
- a. ROOTとは
 - b. 導入方法、初期設定
 - c. 動かし方の基本
 - d. マクロ
 - e. ヒストグラム (TH)
 - f. フィット (TF)
 - g. I/O (TFile)
 - h. グラフ (TGraph)
 - i. ツリー (TTree)
 - j. 乱数生成 (TRandom)
 - k. 高度なフィット (roofit)

