Fission2019-Unified-Analysis-Framework

> - A grobal unified analysis framework to analyze experiment Fission2019 in RIBLL1 in LANZHOU
> - This framework is desiged for Fission2019 data Analysis
>   Contributor: Fenhai Guan(1)
>   (1) gfh16@mails.tsinghua.edu.cn
> - Personal information could be found at the link:
>   http://inspirehep.net/author/profile/Fen.Hai.Guan.1
> - The whole framework is available at the link:
>   https://github.com/gfh16/Fission2019-Unified-Analysis-Framework

目录

Step1. 数据转换

1.1 将原始数据二进制文件转换成.root文件

```
// 运行代码
$ make
$ ./raw2roo.sh listfilename
```

> - **目的:** 将原始数据(二进制文件)转换成.root文件
> - **文件:** /RIBLLVMEDAQ/Raw2ROOT.cpp
> - **说明:** 批量转换
>
> (1) 输入文件(原始文件)都在 /vmedata/文件夹下.
>
> (2) 在 /vmedata/中 添加 listfilename 文件. 将需要格式转换的原始文件名一一写出，每个文件名占一行
>
> (3) 修改 /RIBLLVMEDAQ/Raw2ROOT.cpp文件：修改输出文件的位置 (string rootpath = "./rootdata")
>
> (4) 编译成功后, 执行 ./raw2root.sh listfilename

1.2 将RawRoot数据转换成MapRoot数据

```
// 运行代码
$ make
$ ./ReadRootFile2D listfilename
```

> - **目的:** 第一步得到的.root文件数据以 T103000 等命名，第二步需要利用探测器Map将每个插件对应的探测器还原出来, 以PPAC1_T 等命名
> - **文件:** /RIBLLVMEDAQ/ReadRootFile2D.cpp
> - **说明:**
>
> (1) 修改/RIBLLVMEDAQ/ReadRootFile2D.cpp 文件
>
> (2) 假定输入文件都在 /rootdata/中，在/rootdata/下添加listfilename 文件,将需要转换的文件名称一一列出，每个文件名占一行
>
> (3) 为避免与原始的.root文件混淆，转换后的.root文件需要另起名称，且最好输出到不同的文

> 件夹下
> (4) 执行: 编译成功后, ./ReadRootFile2D listfilename

Step2. 数据质检

2.1 SetBranchAddress方法

- 见 2.2 TTreeReader方法

2.2 TTreeReader方法

```
// 运行代码
$ make
$ ./QC_ReadTree listfilename
```

> - **目的:** 将所有能谱输出为pdf, 便于人眼检查
> - **文件:** QC_BranchAdress.C, QC_ReadTree.C
> - **说明:**
>
> (1) 定义、填充直方图
> (2) 存储所有的Hist，输出pdf到文件, 方便人眼进行质检
> (3) 存储所有的Hist,写入.root文件. 这一步是为了:一旦pdf文件中的谱有问题，马上可以查看.root文件中对应的直方图
> (4) 编译 QC_ReadTree.C 后执行, ./QC_ReadTree listfilename
> (5) 存储、读取 tree 的数据要注意数据类型一致. 比如, 存储数据时使用 Int_t 型, 读取数据时, 也要定义成 Int_t 型; 若定义为 Double_t 型, 则可能出错.

Step3. PPAC数据处理

待定...

Step4. SSD数据处理

4.1 SSD Energy Calibration

> - 硅条能量刻度总结文档

```
// 运行代码
$ root -l ClickToFindPedestals.C                    //手动拟合 pedestal
$ root -l PulserCali_L1_AutoFindPeaksAndFit.C       //自动寻峰
$ root -l PulserCali_L2_AutoFindPeaksAndFit.C       //自动寻峰
$ root -l AlphaCali_CalEnergy.C                     //考虑Mylar膜,计算入射 alpha 的能量
$ root -l AlphaCali_CalEnergyChangingDeadLayer.C    // 计算不同死层下 alpha 的能量
$ root -l AlphaCali_MergeFiles.C                    // 合并 alpha 刻度文件
$ root -l AlphaCali_AutoFindPedestals.C             // 拟合 alpha 刻度文件中的 pedestal
$ root -l AlphaCali_FindPeaks.C                     // 手动拟合 alpha 峰
$ root -l SiEnergyCali.C                            // 以alpha刻度文件中的 pedestal作为cut,拟
$ root -l SiEnergyCali_ModifyDeadLayer.C            // 考虑死层的修正对能量刻度的影响
```

4.1.1 Find Pedestal

> - **目的:** 后续的分析中需要以此作Cut
> - **文件:** ClickToFindPedestals.C
> - **说明:**
>
> (1) Pedestal是探测系统的零点道，是系统没有能量输入情况下，ADC中记录的道址. 理论上，ADC中探测到的所有能量信号都应该在对应的Pedestal以上. 因此Pedestal可以作为ADC能量的Cut值
>
> (2) 写了一个手动选取拟合范围的的程序. 基本操作是: 单击鼠标中间键(滚轮)来取点，单击两次选择拟合范围，最后将拟合结果保存到pdf中，并将拟合参数保存到.dat文件中

```
// 函数 void SetPoints() 用于手动选点
void SetPoints(Int_t event, Int_t x, Int_t y, TObject *selected){}
```

4.1.2 Pulser 线性刻度

> - **目的:** 脉冲线性刻度
> - **文件:**
>
> PulserCali_L1_AutoFindPeaksAndFit.C
>
> PulserCali_L2_AutoFindPeaksAndFit.C
> - **说明:**
>
> (1) 写了一个自动寻峰的函数 *void PulserCali_AutoFindPea()*. 使用ROOT中TSpectrum类中的Search()方法实现自动寻峰.
>
> (2) 将自动寻峰得到的每个峰的Ch作为X值，每个峰对应的输入的pulser的相对幅度作为Y值，画出一系列点
>
> (3) 对这些pulser点进行用 y = a * x +b 进行线性拟合，将拟合参数以及数据点保存到.dat文件中
>
> (4) 将拟合结果保存成pdf，以便检查

4.1.3 α 源刻度

- **目的:** 利用已知能量的 α 刻度
- **文件:**
  AlphaCali_CalEnergy.C
  AlphaCali_MergeFiles.C
  AlphaCali_FindPeaks.C
- **说明:**
  (1) **AlphaCali_CalEnergy.C:** 利用 EnergyLossModule() 函数计算 α 穿过2um镀铝Mylar膜后的能量. 三组分 α 源的三个峰分别来源于239Pu, 241Am, 244Cm, 将三者发射 α 粒子的加权平均能量作为 α 的出射能量
  (2) **AlphaCali_MergeFiles.C:** 合并 α 刻度文件. 使用TChain方法合并刻度文件, 以增加统计量
  (3) **AlphaCali_FindPeaks.C:** 一个手动寻峰的程序 手动选取拟合范围，对三组分α源的三个α峰分别进行了拟合, 并将三个alpha峰的拟合结果保存到.dat文件中
  (4) 同时,对 $3-\alpha$ 峰进行线性拟合, 待后续步骤作为参考. 三个区间叠加拟合
  AlphaCaliHist[SSDNum][CHNum]->Fit(FitPeak1,"R");
  AlphaCaliHist[SSDNum][CHNum]->Fit(FitPeak2,"R+");
  AlphaCaliHist[SSDNum][CHNum]->Fit(FitPeak3,"R+");
  (5) 拟合过程中, 将$3-\alpha$ 拟合结果画出, 如果拟合效果好, 则拟合下一个; 否则, 重新拟合当前的 Ch.

4.1.4 硅条能量线性刻度

- **目的:** pulser 刻度 + α 刻度 $\rightarrow$ 硅条能量线性刻度
- **文件:**
  AlphaCali_AutoFindPedestals.C
  SiEnergyCali.C
  AlphaCali_CalEnergyChangingDeadLayer.C
  SiEnergyCali_ModifyDeadLayer.C
- **说明:**
  (1) **AlphaCali_AutoFindPedestals.C:** 自动寻峰, 拟合 α 刻度文件中的 pedestal, 以背后用.
  (2) **SiEnergyCali.C：** pulser 刻度 + 1个α 能量点 $\rightarrow$ 硅条能量刻度. 同时画出 $3-\alpha$ 拟合结果作为对比.
  (3) **AlphaCali_CalEnergyChangingDeadLayer.C:** 改变死层的厚度, 计算 α 的能量
  (4) **SiEnergyCali_ModifyDeadLayer.C:** 考虑死层修正后, 查看硅条能量刻度的情况是否改善

4.2 CsI Energy Calibration

CsI-能量刻度文档总结
CsI-能量刻度-投影法-DEEFIT教程

- **说明:**

- CsI 能量刻度依赖于硅条的能量刻度, 因此需要硅条能量刻度完成后才能进行 CsI 能量刻度
- CsI 能量刻度可以用两种方法进行: 1.投影法， 2.DEEFIT 方法.
- DEEFIT 是意大利国家核物理研究院(INFN) 为 CHIMERA 探测器开发的一套程序, 专门用于 Si-CsI 的数据分析. 其中包括 CsI 的能量刻度, 粒子鉴别等.

- **DEEFIT - 工作原理简介**
  - 读取已有 $\Delta E - E$ 能谱.
  - 选取特定的核素 (Z, M), 在对应的 $\Delta E - E$ 能谱带上手动选点.
  - 对所有核素的选中的数据点, 用一个多维公式进行拟合.
  - 根据拟合结果, 进行粒子鉴别, 鉴别包括粒子质量 M 与电荷数 Z.
  - 查看粒子鉴别分布情况, 以检验拟合结果的好坏程度.

- **DEEFIT 方法进行 CsI 能量刻度分 5 步:**
  - DEEFIT 方法进行 CsI 能量刻度分 5 步:
  - 1.程序编译
  - 2.数据转换
  - 3.$\Delta E - E$ 能谱拟合
  - 4.计算 CsI 中的能损
  - 5.得到 CsI 能量曲线

4.2.1 程序编译

```
// 在 DEEFIT/ 文件夹下直接 make 即可.
// 如果遇到因为 ROOT 版本而编译不成功, 可进行下面的尝试:
1. ROOT版本问题：在ROOT6版本下，主要问题在于：
   deedict.cxx:163:65: error: ‘DefineBehavior’ was not declared in this scope

2. 在ROOT6中，‘DefineBehavior’等函数定义在命名空间ROOT::Internal中

3. 解决办法：在 deedict.cxx 文件中，加入命名空间：
   using namesapce ROOT::Internal;
```

4.2.2 数据转换

- **目的:** DEEFIT 对读取的 .root 文件格式有要求. 因此需要先进行数据转换.
- **文件:**
- **说明:**

  1.DEEFIT 程序要求 .root 文件存成 Tree 的格式. 共有 3 个 branch:

  h1: histogram name

  numtel: 类型为short, 探测器编号. 按照 CsI 晶体进行标记.

  例如: 我们有4块硅条, 每块硅条有 9 块 CsI 晶体, 应该编号 $0 \sim 35$.

> desilpgf: 类型为float, 硅条中的能能损，以 MeV 为单位, 依赖硅条能量刻度.
> fastpg: 类型为short, CsI 晶体中的能损，以 ADC 道址为单位.

4.2.3 $\Delta E - E$ 能谱拟合

- **目的:**
- **文件:**
- **说明:**

4.2.4 计算 CsI 中的能损

- **目的:**
- **文件:**
- **说明:**

4.2.5 得到 CsI 能量曲线

- **目的:**
- **文件:**
- **说明:**

4.3 Hit Multiplicity && Hit Pattern

- 硅条粒子多重性与 HitPattern 总结文档

```
// 运行代码
$ root -l HitMultiplicity.C
$ root -l HitMultiplicityChangingSigma.C
$ root -l HitPattern.C
```

4.3.1 Hit Multiplicity

- **目的:** 获得单次次事件下,硅条探测到的粒子多重数 $\mathbb{M}$
- **文件:** HitMultiplicity.C
- **说明:**
  1.粒子多重数, 是指每次触发时, 每块硅条探测到的粒子数目, 即点火的条数. 因此$0 \leq \mathbb{M} \leq 16$.
  2.如何判断有效的点火事件?
  (1) 如果 $E_{CH} > (Mean_{pedestal} + \sigma_{pedestal} \times N_{\sigma})$, 则认为是一个有效击中事件, 多重数 $+1$
  3.初步分析, 取 $N_{\sigma} = 5$.
  4.进一步分析,对比不同的 N_{σ} 值对多重数 $\mathbb{M}$ 的影响. 取 $N_{\sigma} = 3, 5, 10, 20, 30, 40$

```
// 判断有效击中事件
if (SSD_E[i][j]>(PedestalMean[i][j]+PedestalSigma[i][j]*SigmaNum))
{
  SSD_E_HitMultiplicity[i]++;
}
```

4.3.2 Hit Pattern

> - **目的:** 查看粒子在硅条中的点火情况
> - **文件:** HitPattern.C
> - **说明:**
>
> 1.先获得粒子多重数 $\mathbb{M}$, 以 $\mathbb{M}$ 作为条件, 查看不同情况下硅条的点火情况. 多加这个条件, 是为以后查看 1-hit，2-hit 事件的点火情况提供方便.
> 2.设定一个多重数的Cut值 M_{Cut}, 当 $\mathbb{M} < M_{Cut}$时,对点火时间进行判选
> 3.对于双面硅条的二维 Hit Pattern, 需要作两重遍历;

```
// 硅条第一层 L1S 的 Hit Pattern 判选条件
if (HitMultiplicity_SSD_L1S[i]<HitMultiplicityCut) {
  for (Int_t j=0; j<CHNum; j++)
  {
    if (ECh_SSD_L1S[i][j]>(PedestalMean_L1S[i][j]+PedestalSigma_L1S[i][j]*SigmaNum)) {
        hist_HitPattern[i]->Fill(j+1, 1.0);
    }
  }
}
```

```
// 硅条第二层的 Hit Pattern 判选条件
if ((HitMultiplicity_SSD_L2F[i]<HitMultiplicityCut)||
   (HitMultiplicity_SSD_L2B[i]<HitMultiplicityCut)) {
   for (Int_t j=0; j<CHNum; j++)
   {
      if ((ECh_SSD_L2F[i][j]>PedestalCut_L2F[i][j])) {
        for (Int_t k=0; k<CHNum; k++)
        {
           if ((ECh_SSD_L2B[i][k]>PedestalCut_L2B[i][k]) {
               hist2D_HitPattern[i]->Fill(j+1, k+1, 1.0);
           }
        }
      }
    }
}
```

4.4 Hit Pixellation

4.5 Particle Identification

Step5. 物理分析