A Dual-Bunch Data Analysis Method at the China Spallation Neutron Source

Authors: Gan Lin, Xiaotong Liu, Jiang Wei, Liu Xiaotong

Date: 2024-05-10T00:00:00+00:00

Abstract

China Spallation Neutron Source (CSNS) provides white neutron beams from 0.3 eV to 300 MeV with a total beam intensity up to 10^7 n/s/cm^2, constituting an excellent experimental platform for measuring neutron capture reaction cross sections. In normal operation mode, CSNS accelerates two proton beam bunches separated by 410 ns that strike the target sequentially, thereby generating a neutron beam also composed of two bunches mixed with a 410 ns separation. To avoid mutual interference between the effects of the two bunches, which would compromise the energy precision of neutron capture cross sections, experimental data must be parsed and reconstructed to recover the effects of individual bunches. Existing parsing methods can achieve very fine spectrum unfolding results but are relatively complex and entail a certain usage threshold. This work proposes a simplified dual-bunch spectrum unfolding method that, while ensuring neutron energy precision, is applicable to data with neutron energies below 1.2 MeV, offering a new data processing approach for similar experimental endeavors.

Full Text

A Simplified Method for Unfolding Double-Bunch Data at CSNS

Gan Lin[1], Liu Xiaotong[1], Jiang Wei[2],[3]
[1](School of Microelectronics, Shenzhen Institute of Information Technology, Shenzhen 518172, China)
[2](Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China)
[3](Spallation Neutron Source Science Center, Dongguan 523803, China)

Abstract

Background: The China Spallation Neutron Source (CSNS) provides a white neutron beam with energies ranging from 0.3 eV to 300 MeV and a total beam intensity of up to 10^7 n/s/cm^2, serving as an excellent experimental platform for measuring neutron capture reaction cross sections. During normal operation, the accelerator generates two proton bunches separated by 410 ns that strike the target consecutively, producing a mixed neutron beam composed of two bunches with the same interval. To avoid interference between the two bunches and maintain the energy precision of neutron capture cross sections, experimental data must be analyzed and reconstructed to restore the effects of individual bunches.

Purpose: Existing unfolding methods can produce highly refined results but are relatively complex and have a steep learning curve. This work proposes a simplified double-bunch unfolding method that, while ensuring neutron energy precision, is applicable to data with neutron energies below 1.2 MeV, offering a new data processing approach for similar experimental work.

Methods: This work employed mathematical operations to analyze and reconstruct data using 410 ns as the unit time, processing the data with a channel width of 4100 ns. Additionally, a comparison was made between the impacts of this method and existing methods on the accuracy of neutron incident energy.

Results: The proposed simplified data processing method achieves the same energy resolution as existing methods in the low-to-medium energy range, providing a novel data processing solution for similar experimental work.

Conclusions: The simplified data processing method presented in this study effectively addresses the issue of excessive computational costs when analyzing low-to-medium energy neutron data from CSNS, offering a practical solution for experimental work requiring accurate analysis of neutron capture reactions in this energy range.

Keywords: CSNS; Double-bunch unfolding method; Neutron capture reactions

Introduction

The China Spallation Neutron Source (CSNS), located in Dongguan, Guangdong Province, is China' s first white neutron source experimental platform, providing a powerful research platform for scientists across China and worldwide [1-2]. CSNS generates its neutron beam by accelerating high-energy proton beams that are magnetically deflected to strike a spallation target. The accelerator operates at a frequency of 25 Hz with proton energies of approximately 1.6 GeV. A schematic diagram of the CSNS facility is shown in Figure 1 [Figure 1: see original paper]. A neutron beam is extracted at a 180° angle relative to the proton beam direction, known as the back-streaming white neutron beam line (back_n), which contains neutrons with continuous energies from 0.3 eV to 300

MeV and a maximum neutron flux of 10^7 n/cm^2/s. Two experimental terminals, ES#1 and ES#2, are located 55 m and 76 m from the target, respectively. ES#1 offers higher neutron beam intensity, while ES#2 provides better neutron energy resolution, allowing users to select the appropriate terminal based on their experimental requirements [3-6]. Since beam commissioning began in 2018, CSNS has provided beam support for experiments in fundamental physics, materials science, and other disciplines [7-9].

During routine operation, the accelerator employs a double-bunch mode, where two identical proton bunches strike the spallation target with a 410 ns interval. Consequently, the measured experimental data represent the superposition of reaction effects from two bunches separated by 410 ns. The incident neutron energy is calculated based on time-of-flight, with the starting point being the arrival time of the first proton bunch at the target. Neutrons of the same energy from the second bunch correspond to a time-of-flight increased by 410 ns. When measuring neutron capture cross sections, the superposition of the two bunches degrades the precision of the calculated incident neutron energy, particularly for medium-to-high energy neutrons. Therefore, to obtain accurate neutron capture excitation functions, double-bunch data must be unfolded and reconstructed to recover the true effects of individual neutron bunches.

In 2020, Yi Han et al. from the Institute of High Energy Physics, Chinese Academy of Sciences, developed an unfolding program for CSNS double-bunch data [10] that can produce refined neutron reaction cross sections. The program is available for download and compilation at http://code.ihep.ac.cn/yih/csns-back-n-doublebunchunfolder. Users must first convert measured data into histogram-style ROOT files, then integrate the program into their data processing workflow. After 30 to hundreds of iterations, the program finally produces either a one-dimensional time-of-flight spectrum or a two-dimensional time-of-energy spectrum. To ensure that the unfolded data accurately reproduces the true reaction yield, the recommended time channel width should not exceed one-tenth of the delay time, i.e., 41 ns. Traditional neutron capture cross-section measurements use C_6D_6 liquid scintillation detectors with high gamma-ray sensitivity to capture all gamma rays produced in reactions. The data processing then calculates the proportion of neutron capture reactions using weighting factors to obtain the target reaction cross section. Consequently, detectors typically acquire sufficient statistics, allowing for very small time channel widths while maintaining small statistical errors. This approach yields highly detailed excitation functions (curves of neutron capture cross section versus incident energy) but entails enormous computational costs. The program developers therefore recommend that the time span for data unfolding should not exceed 10^5 ns, as longer spans may cause memory errors and program failure. Furthermore, the program was developed for Linux systems and has a relatively complex usage methodology, creating a barrier for users unfamiliar with Linux systems and ROOT programming.

Two scenarios warrant particular consideration: First, when experimental data

have insufficient statistics, using small time channel widths results in very low counts per channel and excessive statistical errors. Second, when measuring low-energy neutron data with time-of-flight in the 10^5 ns to 10^6 ns range, the impact of wide channel widths on energy error becomes negligible, while continuing to use small channel widths introduces unnecessary computational burden. In both cases, larger time channel widths are more appropriate, and a simplified unfolding method can achieve the data analysis objectives while meeting experimental precision requirements. This work proposes a simplified unfolding method for double-bunch data using a 4100 ns time window as an example, providing a new data processing solution for similar neutron capture cross-section measurements.

1. Principle of Double-Bunch Data Unfolding

The neutron beam at back_n can be considered as the superposition of two neutron bunches with identical energy distributions but separated by 410 ns, referred to as bunch A and bunch B. The detector time is set to start at the moment bunch A is produced. Therefore, within detector time 0–410 ns, all data acquired by the detector originate from bunch A' s effects during this period. Within detector time 410–820 ns, the data include both bunch A' s effects in the second time period and bunch B' s effects in the first time period. This pattern continues such that in each subsequent 410 ns interval, the data include bunch A' s effects in that interval plus bunch B' s effects from the previous interval.

Using 410 ns as the unit time, let f(n) represent the data produced by a single bunch in the nth unit time interval, and A(n) represent the data acquired by the detector in the corresponding interval. The relationship is:

$$A(1) = f(1)$$
$$A(2) = f(1) + f(2)$$
$$A(n+1) = f(n) + f(n+1)$$

Since the time window is 4100 ns, each data group includes 10 time intervals. Let the starting point of any data group be the beginning of the xth time interval. The sum of data from a single bunch within this time window is $S = f(x) + f(x+1) + f(x+2) + \cdots + f(x+9)$. From the equations above, we know that $f(x) + f(x+1) = A(x+1)$, $f(x+2) + f(x+3) = A(x+3)$, $\cdots$, $f(x+8) + f(x+9) = A(x+9)$. Therefore, the relationship between single-bunch data and detector-acquired data can be rewritten as:

$$S = A(x+1) + A(x+3) + A(x+5) + A(x+7) + A(x+9)$$

Equation (2) shows that by extracting data at intervals and recombining them, we can obtain the single-bunch data within each data group. Additionally, since detectors have different efficiencies for gamma rays of different energies, the energy spectrum for each data group must also be extracted.

In this work, the data unfolding process is divided into four steps: (1) Export the time-energy two-dimensional spectrum stored in ROOT files to text documents;

(2) Split the document into multiple sub-documents using 410 ns time intervals; (3) Extract sub-documents at intervals for each data group and merge them into a new document; (4) Obtain the energy spectrum for each data group.

Furthermore, from equations (1) and (2), the minimum time channel width for this method is 820 ns.

2. Implementation

2.1 Exporting ROOT Data to Text Files

Since ROOT files generated by different data acquisition systems have different branch structures, this section uses our team' s data structure as an example. The branch names for energy and time are "E" and "DeltaT" , with data types Double_t and ULong64_t, respectively. Variables E1 and DT1 of the corresponding data types are defined to extract energy and time data from the ROOT file. The complete data processing code is provided in Appendix 1. The code reads the ROOT file event by event and stores each event' s energy and time information in a text document. Users should modify the corresponding content based on their specific situation. The key program statements are as follows:

```
// t1 is a pointer to the Tree.
// 1) Link variables to data branches:
t1->SetBranchAddress("E", &E1);
t1->SetBranchAddress("DeltaT", &DT1);

// 2) Create text document input.dat to store exported data:
ofstream outfile("input.dat");

// 3) Get total number of events in ROOT file and process event by event:
Long64_t ntot1 = t1->GetEntriesFast(); // Get total number of events
for(i = 0; i < ntot1; i++)
{
    t1->GetEntry(i);
    outfile << DT1 << " " << E1 << endl;
}
```

2.2 Splitting Data Files

The complete code for this section is provided in Appendix 2. The program requires reading the file intervals.txt, which stores time interval values with two numbers per group representing start and end times in ns. The split sub-files are sequentially named 001.dat, 002.dat, etc., with the number of files determined by the number of time groups. To ensure compliance with the data processing principle, the start time must be an integer multiple of 410 ns, and each group' s time interval must be exactly 410 ns.

2.3 Reconstructing Data Files

The complete code for this section is provided in Appendix 3. The program requires reading the file datfile.txt, which stores the names of all sub-files including extensions. The reconstructed data files are sequentially named add1.dat, add2.dat, etc., with the number of files determined by the number of data groups. The program sets an upper limit of 45 data groups; if the number exceeds 45, the upper limit for the group variable in line 34 of the program must be modified.

2.4 Obtaining Energy Spectra

The data generated in Section 2.3 includes two-dimensional time and energy information, where the first column is time and the second column is energy. To obtain the energy spectrum, the program reads all energy values in the file and counts the occurrences of energy values in each channel according to the user-specified channel width. The complete program is provided in Appendix 4. The names of files requiring energy spectrum extraction are stored in the filename.txt file. The energy channel width can be modified in line 30 of the program. After execution, new files with the same name as the source file but with a .txt extension are generated. These new files contain two columns of data: energy in the first column and corresponding counts in the second column. Users can thus obtain energy spectra for individual bunches across multiple time windows.

2.5 Development Environment

- **Operating System:** Windows 10 Professional Edition
- **ROOT Version:** root_{v6}.30/04
- **C++ Compiler:** Visual Studio 2021

3. Results and Discussion

The incident neutron energy is inversely proportional to the square of the time-of-flight. With constant time channel width, the relative error in neutron energy increases quadratically with increasing incident neutron energy. When measuring neutron capture cross sections at the ES#2 terminal, using a 41 ns time channel width for energy regions above 10 keV can control the relative neutron energy error within 15%. The proposed method primarily targets the energy region below 10 keV, where a 4100 ns time channel width achieves the same error control. Figure 2 [Figure 2: see original paper] shows the neutron energy error curves for channel widths of 4100 ns and 41 ns. The solid line represents the 4100 ns channel width, while the dotted line represents the 41 ns channel width. The two channel widths produce nearly identical relative error curves in their respective energy regions.

To better illustrate the impact of neutron energy error, our team measured

the neutron capture cross section of ^{91}Zr on the back_n beam line, covering a time range from 11,480 ns to 191,880 ns with a time channel width of 4100 ns, yielding 44 data points corresponding to neutron energies from 0.82 keV to 229 keV. The neutron cross section obtained using this method is shown in Figure 3 [Figure 3: see original paper]. The horizontal error bars clearly demonstrate that when the incident neutron energy exceeds 10 keV, the neutron energy error becomes significant, with relative errors reaching approximately 90%. The neutron capture cross section variation also becomes relatively flat, making it difficult to determine whether this trend reflects the actual cross section behavior or results from averaging that obscures the nuclear reaction structure information. As the incident energy decreases, the neutron energy error is reduced to small values, and the measured neutron capture cross section exhibits dramatic fluctuations that well reflect the resonance characteristics of the capture cross section in this energy region.

These results demonstrate that for neutron incident energies above 10 keV with sufficient statistics, the refined unfolding program can yield more accurate capture cross sections that faithfully represent reaction structure information, whereas larger time channel widths would lose this structural information. However, for neutron incident energies below 10 keV, this simplified method can also produce highly accurate excitation function curves.

Notably, the minimum time channel width supported by this method is 820 ns, which can maintain neutron energy resolution within 15% for neutron incident energies up to 1.2 MeV.

Conclusion

The CSNS back_n beam line provides an excellent experimental platform for neutron capture reaction cross-section measurements. However, in normal operation mode, the neutron beam consists of two bunches separated by 410 ns. To obtain accurate incident neutron energies, double-bunch data must be unfolded and reconstructed. Although a refined unfolding method exists, it is relatively complex to use and presents a barrier for users. Moreover, for data requiring larger time channel widths, the refined method is unnecessarily burdensome. This work proposes a highly simplified unfolding method based on fundamental mathematical operations that splits and reconstructs data using larger time channel widths. All programs can be compiled and run directly under Windows systems, making them simple and user-friendly.

When using the minimum time channel width of 820 ns, this method achieves neutron energy resolution comparable to refined unfolding methods for neutron incident energies below 1.2 MeV, while dramatically reducing computational requirements. Additionally, compared to refined methods, larger time channel widths increase statistical counts per channel and reduce statistical errors, offering significant advantages for experiments with insufficient statistics. In summary, for data with sufficient statistics and higher neutron incident ener-

gies, existing unfolding methods are more advantageous. For data with limited statistics and neutron incident energies in the low-to-medium range, the proposed method offers benefits of reduced computational load and operational simplicity.

References

1. Zhang Jie. China Spallation Neutron Source (CSNS) - A Large Science Platform for Multidisciplinary Applications [J]. Bulletin of the Chinese Academy of Sciences, 2006, 21(5): 3. DOI: 10.3969/j.issn.1000-3045.2006.05.017.

2. Wei Jie. Introduction to the China Spallation Neutron Source (CSNS) [J]. Modern Physics Knowledge, 2007, 19(6): 8. DOI: CNKI:SUN:XDWZ.0.2007-06-006.

3. Jing H T, Tang J Y and Tang H Q, et al. Studies of back-streaming white neutrons at CSNS[J]. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2010, 621(1-3): 91-96. DOI: 10.1016/j.nima.2010.06.097.

4. Chen Yanwei. China Spallation Neutron Source (CSNS) [J]. Bulletin of the Chinese Academy of Sciences, 2011, 66(6): 726-728. DOI: 10.3969/j.issn.0368-6396.2014.04.005.

5. Zhang L Y, Jing H T, Tang J Y, et al. Design of back-streaming white neutron beam line at CSNS[J]. Applied Radiation and Isotopes: Including Data, Instrumentation and Methods for Use in Agriculture, Industry and Medicine, 2018, 132: 212-221. DOI: 10.1016/j.apradiso.2017.11.013.

6. Tang J, Liu R, Zhang G, et al. Initial years' neutron-induced cross-section measurements at the CSNS Back-n white neutron source. Chinese Physics C, 2021, 45(6): 062001. DOI: 10.1088/1674-1137/abf138.

7. Hu X R, Fan G T, Jiang W, et al. Measurements of the ^{197}Au(n, γ) cross section up to 100 keV at the CSNS Back-n facility[J]. 2021, 32(9): 101. DOI: 10.1007/s41365-021-00931-w.

8. Li X, An Z, Jiang, W, et al. CSNS Back-n Collaboration. Measurement of the natEu (n, γ) cross section up to 500 keV at the CSNS Back-n facility, and the stellar ^{151},^{153}Eu (n, γ) cross section at s-process temperatures. European Physical Journal A, 2022, 58(12): 251. DOI: 10.1140/epja/s10050-022-00887-4.

9. Chen Y, Qiu Y, Li Q, et al. Measurement of the neutron flux of CSNS Back-n ES#1 under small collimators from 0.5 eV to 300 MeV. The European Physical Journal A, 2024, 60(3), 63. DOI: 10.1140/epja/s10050-024-01272-z.

10. Yi H, Wang T F, Li Y, et al. Double-bunch unfolding methods for the Back-n white neutron source at CSNS[J]. Journal of Instrumentation, 2020, 15(03): P03026. DOI: 10.1088/1748-0221/15/03/P03026.

Appendices

Appendix 1: Code for Exporting ROOT Data

```
#include <fstream>
#include <string>

void ana_{extract}(const char *fin1) // Note: Function name must match .C filename, otherwis
{
    Double_t E1; // Energy variable
    Long64_t i;  // Event number variable
    ULong64_t DT1; // Time-of-flight in ns

    TFile* fRun1 = new TFile(fin1);
    TTree* t1 = (TTree*)fRun1->Get("tr"); // Create Tree pointer t1 to get TTree from ROOT f
    t1->SetBranchAddress("E", &E1);
    t1->SetBranchAddress("DeltaT", &DT1);

    Long64_t ntot1 = t1->GetEntriesFast(); // Get total number of events
    ofstream outfile("input.dat"); // Create data storage document

    outfile << "DT" << " " << "E" << endl; // Output DT and E in first line of document

    for(i = 0; i < ntot1; i++) // Process data event by event
    {
        t1->GetEntry(i);
        if(i%(ntot1/10)==0)
            std::cout<<"processing:"<<i*10/(ntot1/10)<<"%"<<std::endl; // Display processing
        outfile << DT1 << " " << E1 << endl; // Write time and energy data to document
    }

    fRun1->Close(); // Close ROOT file, release memory
    outfile.close();
}
```

Appendix 2: Code for Splitting Data Files

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <iomanip>
```

```
// Create data structure variable
struct Data {
    double DT;
    double E;
};

int main()
{
    std::ifstream intervalsFile("intervals.txt"); // Open file storing time intervals
    if (!intervalsFile.is_{open}()) {
        std::cerr << "Failed to open intervals.txt" << std::endl;
        return 1;
    }

    std::vector<std::pair<double, double> > intervals;
    double start, end;

    // Read start and end times
    while (intervalsFile >> start >> end) {
        intervals.push_{back}(std::make_{pair}(start,end));
    }
    intervalsFile.close();

    std::ifstream dataFile("input.dat");
    if (!dataFile.is_{open}()) {
        std::cerr << "Failed to open data.dat" << std::endl;
        return 1;
    }

    std::string title;
    std::getline(dataFile, title);

    std::vector<Data> dataVec;
    double dt, e;
    while (dataFile >> dt >> e) {
        Data data;
        data.DT = dt;
        data.E = e;
        dataVec.push_{back}(data);
    }
    dataFile.close();

    int fileCount = 0;
    for (const auto& interval : intervals) {
        std::vector<Data> filteredData;
```

```
        for (const auto& data : dataVec) {
            if (data.DT >= interval.first && data.DT <= interval.second) {
                filteredData.push_{back}(data);
            }
        }

        // Create sub-document storing data for each time interval
        std::stringstream filename;
        filename << std::setfill('0') << std::setw(3) << ++fileCount << ".dat";

        std::ofstream outFile(filename.str());
        if (!outFile.is_{open}()) {
            std::cerr << "Failed to create output file: " << filename.str() << std::endl;
            return 1;
        }

        for (const auto& data : filteredData) {
            outFile << data.DT << " " << (int)(data.E + 0.5) << std::endl; // Round energy t
        }
        outFile.close();
    }

    std::cout << "Data filtering completed successfully." << std::endl;
    return 0;
}
```

Appendix 3: Code for Reconstructing Data Files

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>

// Read filename list
std::vector<std::string> ReadFileNames()
{
    std::ifstream inputFile("datfile.txt");
    std::vector<std::string> filenames;
    std::string filename;

    while (std::getline(inputFile, filename)) {
        filenames.push_{back}(filename);
    }
    return filenames;
}
```

```
// Append source file content to destination file
void AppendFileContent(const std::string& srcFile, const std::string& destFile)
{
    std::ifstream src(srcFile, std::ios::binary);
    std::ofstream dest(destFile, std::ios::binary | std::ios::app);

    if (!src || !dest) {
        std::cerr << "Error opening files!" << std::endl;
        return;
    }
    dest << src.rdbuf();
}

int main()
{
    std::vector<std::string> filenames = ReadFileNames();

    for (int group = 0; group < 45; ++group) // Modify upper limit of group based on number
    {
        std::string destFilename = "add";
        destFilename.append(std::to_{string}(group + 1));

        for (int i = 3 - destFilename.length(); i > 0; --i) {
            destFilename.insert(destFilename.begin(), '0');
        }
        destFilename += ".dat"; // Add file extension

        std::ofstream dest(destFilename, std::ios::binary);

        for (int i = group * 10; i < std::min((group + 1) * 10, static_{cast}<int>(filenames
            if (i % 2 != 0) // Extract sub-files at intervals
                AppendFileContent(filenames[i], destFilename);
        }
    }
    return 0;
}
```

Appendix 4: Code for Obtaining Energy Spectra

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <string>
#include <vector>
```

```cpp
int main()
{
    // Input file stream
    std::ifstream filenameFile("filename.txt");
    if (!filenameFile.is_{open}()) {
        std::cerr << "Failed to open filename.txt" << std::endl;
        return 1;
    }

    // Output file stream
    std::vector<std::string> filenames;
    std::string filename;

    while (std::getline(filenameFile, filename)) {
        filenames.push_{back}(filename);
    }
    filenameFile.close();

    // Count occurrences in each interval
    for (const auto& filename : filenames) {
        std::ifstream file(filename);
        if (!file.is_{open}()) {
            std::cerr << "Failed to open file: " << filename << std::endl;
            continue;
        }

        std::map<int, int> countMap;
        int dt, e;
        int bin = 20; // Energy channel width

        while (file >> dt >> e) {
            int interval = (e / bin) * bin; // After binning
            countMap[interval]++;
        }
        file.close();

        // Create new file
        std::string outputFilename = filename.substr(0, filename.find_{{{last}}_{{of}}}(".")
        std::ofstream outputFile(outputFilename);

        if (!outputFile.is_{open}()) {
            std::cerr << "Failed to create output file: " << outputFilename << std::endl;
            continue;
        }
```

```
        for (const auto& pair : countMap) {
            outputFile << pair.first << " " << pair.second << std::endl;
        }
        outputFile.close();
    }

    std::cout << "Data counting completed successfully." << std::endl;
    return 0;
}
```

Note: Figure translations are in progress. See original paper for figures.