

Rejestracja promieniowania kosmicznego za
pomocą licznika Geigera-Müllera i LED -
wizualizacja
Zespołowy Projekt Studencki - gr. 8

Anna Przybysz
Jacek Koniarski

opiekun: dr hab. Marcin Konecki, prof. UW

10 września 2020

Spis treści

1	Wstęp	3
1.1	Licznik Geigera-Müllera	3
1.2	Detekcja mionów	3
2	Praca z danymi	3
2.1	Wykorzystane biblioteki	4
2.2	Ustawienia domyślne	5
2.3	Kod	5
3	Wykresy	8
4	Podsumowanie	10

1 Wstęp

Celem projektu studenckiego jest wizualizacja danych, pochodzących z układu wykonanego przez poprzednie grupy. Specjalnie zaprojektowane płyty wraz z systemem diod i licznikami Geigera-Müllera zostały podłączone do Raspberry Pi 3B+. Pośredniczył przy tym układ trygujący, który miał za zadanie wykrywać koincydencję pomiędzy warstwami liczników, co z kolei interpretowane było jako detekcja mionu.

1.1 Licznik Geigera-Müllera

Każda z płytek wchodzących w skład układu składa się z ośmiu cylindrycznych liczników G-M. Liczniki te składają się z metalowej tuby wypełnionej gazem oraz układem elektrycznym. W tubie znajduje się metalowy pręcik będący pod napięciem rzędu 300 - 400 V. Pomiedzy pręcikiem a metalową obudową tuby powstaje pole elektryczne. W momencie, w którym do tuby wpada cząstka promieniowania jonizującego, dochodzi do zjonizowania obecnego gazu i w rezultacie wytrącenia wolnego elektronu. Jest on następnie przyspieszany w polu elektrycznym, a po drodze zderza się z innymi atomami gazu i powoduje wyładowanie lawinowe. W konsekwencji w układzie elektrycznym obserwowany jest impuls elektryczny, który interpretowany jest jako detekcja promieniowania.

1.2 Detekcja mionów

Liczniki G-M wykrywają promieniowanie jonizujące, w skład którego wchodzi wtórne promieniowanie kosmiczne. Pierwotne promieniowanie kosmiczne wchodzi w interakcję z atmosferą ziemską, czego efektem jest, między innymi, emisja wysokoenergetycznych mionów. Miony te można obserwować przy powierzchni Ziemi, za pomocą aparatury przypominającej nasz układ.

2 Praca z danymi

Projekt zakłada, że w swojej finalnej formie będzie analizował i wyświetlał dane w czasie rzeczywistym. Nasza grupa skoncentrowała się na danych, które zostały już zebrane i zapisane w pliku dane13.txt (Rys. 1.). Niemniej jednak, oba nasze skrypty wykorzystują generatory, tak aby nie wczytywać całych plików przed wyświetleniem wykresu.

Wszystkie skrypty napisaliśmy w języku Python (wersja 3.7).

```

2019-04-10 15:08:35.920168
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
2019-04-10 15:08:38.189647
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
2019-04-10 15:08:40.718384
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
2019-04-10 15:08:51.090136
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
2019-04-10 15:08:55.849545
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
2019-04-10 15:08:59.020560
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]

```

Rysunek 1: Dane wejściowe z pliku dane13.txt (początek pliku)

2.1 Wykorzystane biblioteki

Podstawową biblioteką Pythonowską, na której się oparliśmy, jest Matplotlib. Ponadto korzystaliśmy z:

- argparse - parsowanie danych wejściowych
- datetime, time - wyznaczanie odstępów czasowych
- NumPy - wyznaczanie częstości wystąpień
- csv, pandas, itertools - przy generatorze całkowitych zliczeń w zadanym odstępie czasu
- Matplotlib, seaborn - wizualizacja wykresów

2.2 Ustawienia domyślne

W celu uruchomienia programu, który wyświetli macierze zapaleń diod oraz wykresy częstotliwości z uwzględnieniem trygera, należy wpisać w terminalu `python3 script.py dane13.txt`.

Dane, które otrzymaliśmy, dotyczyły 3 warstw dla 8 rurek, co przełożyliśmy na macierz 3x8. Jednakże finalnie projekt zakłada więcej rurek, toteż nasz skrypt został przygotowany do łatwego skalowania. Analogicznie, domyślnie wybraną wartością odstępów czasowych wszystkich zliczeń została wybrana 1 minuta, jednak można to modyfikować przy odpaleniu programu.

```
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('input_file', help="data file")
    parser.add_argument('-n', '--nrows', type=int, default=3,
                        help="Number of matrix rows. Default is 3")
    parser.add_argument('-m', '--ncols', type=int, default=8,
                        help="Number of matrix columns. Default is 8")
    parser.add_argument('-t', '--time_threshold', default="00:01:00",
                        help="Time threshold for calculating total. "
                             " Should be in format: HH:MM:SS. Default "
                             " is 00:01:00 (1 min)")
    args = parser.parse_args()
```

Rysunek 2: Domyślne parametry wymiaru macierzy oraz odstępów czasowych.

Chcąc uzyskać wykres, który dla zadanego odstępów czasowych wyświetli: kolejne odstępów (na osi OX) oraz liczbę wszystkich zliczeń (na osi OY), należy wpisać `python3 script_write.py dane13.txt`, a następnie `python3 total.py`.

2.3 Kod

Parsowaniem danych zajmuje się funkcja `read_data`, która zawiera w sobie generator.

```
def read_data(fname):
    with open(fname, 'r') as f:
        date = datetime.datetime.fromisoformat(next(f).rstrip('\n'))
        t = date - date
        m = []
```

```

while True:
    try:
        line = next(f)
    except StopIteration:
        time.sleep(1)
    else:
        line = line.rstrip('\n').strip()
        if line.startswith('[') and line.endswith(']'):
            line = \
                list(map(int,
                        line.lstrip('[').rstrip(']').split(',')))
            m.append(line)
        else:
            yield np.array(m), t
            t = datetime.datetime.fromisoformat(line) - date
            date = datetime.datetime.fromisoformat(line)
            m = []

```

Poniżej prezentujemy funkcje pomocnicze. Funkcja `find_trigger` sprawdza, czy obecnie przetwarzany rekord zawiera tryger i zwraca prawdę lub fałsz w zależności od rezultatu. Funkcja `find_second_layer_index_for_triggered` znajduje indeksy kolumn, w których zarejestrowany został sygnał, w drugiej warstwie liczników. Funkcja zwraca indeksy oraz pomocniczą zmienną logiczną zawierającą informację, czy jakiś indeks został znaleziony.

```

def find_trigger(mat):
    signal_present_in_first_row = False
    signal_present_in_last_row = False
    n, m = mat.shape
    for j in range(m):
        if mat[0, j] == 1:
            signal_present_in_first_row = True
        if mat[-1, j] == 1:
            signal_present_in_last_row = True
    #returns True if there is a signal in both the first and the last row
    return signal_present_in_first_row and signal_present_in_last_row

def find_second_layer_index_for_triggered(mat):
    n, m = mat.shape
    indices = np.zeros(m, dtype=float)
    signal_found = False
    for j in range(m):
        if mat[1, j] == 1:
            indices[j] = 1
            signal_found = True
    return indices, signal_found

```

Macierze zapaleń zostały napisane jako heatmapy. Wszystkie cztery wykresy zostały włączone jako 4 subploty, które wyświetlają się jednocześnie. Poniżej prezentujemy kod uwzględniający kolejno: macierz ostatnich zapaleń, macierz zliczająca wszystkie zapalenia danej diody, wykres częstotliwości rejestracji sygnału dla n-tej rurki po wyzwoleniu trygera oraz wykres częstotliwości wystąpienia wzorców pionowych dla n-tej rurki.

```
#heat_map = last matrix
fig.add_subplot(221)
heat_map = sns.heatmap(m, xticklabels=x_axis_labels,
                        yticklabels=y_axis_labels, annot=True,
                        cbar=False, linewidths=.5)
plt.title('Last matrix', size=15)
plt.xlabel('column', size=12)
plt.ylabel('row', size=12)
heat_map.set_yticklabels(heat_map.get_yticklabels(), rotation=60)
#count_mat = matrix of total counts
fig.add_subplot(222)
count_mat = sns.heatmap(counts_matrix, xticklabels=x_axis_labels,
                        yticklabels=y_axis_labels, annot=True,
                        cbar=False, linewidths=.5)
plt.xlabel('column', size=12)
plt.ylabel('row', size=12)
plt.title('Matrix - Total Counts', size=15)
count_mat.set_yticklabels(count_mat.get_yticklabels(), rotation=60)
fig.add_subplot(223)
no_col=('1','2','3','4','5','6','7','8')
freq = first_type_histogram_vector/triggers_count
#freq = one_one_one_pattern
x_pos = np.arange(len(no_col))
plt.xlabel('column')
plt.ylabel('frequency')
plt.title('Frequency of nth pipe registering signal after trigger')
plt.bar(x_pos, freq, color='skyblue')
plt.xticks(x_pos, no_col)
fig.add_subplot(224)
no_col = ('1', '2', '3', '4', '5', '6', '7', '8')
freq = (one_one_one_pattern-one_zero_one_pattern)/one_one_one_pattern
#freq = one_zero_one_pattern
x_pos = np.arange(len(no_col))
plt.xlabel('column')
plt.ylabel('frequency')
plt.title('Frequency of correctly registered vertical patterns in nth pipe')
plt.bar(x_pos, freq, color='skyblue')
plt.xticks(x_pos, no_col)
```

Wbudowana w bibliotekę Matplotlib funkcja FuncAnimation została zaadaptowana do wygenerowania wykresu wszystkich zliczeń zapaleń w zadanych odstępach czasu.

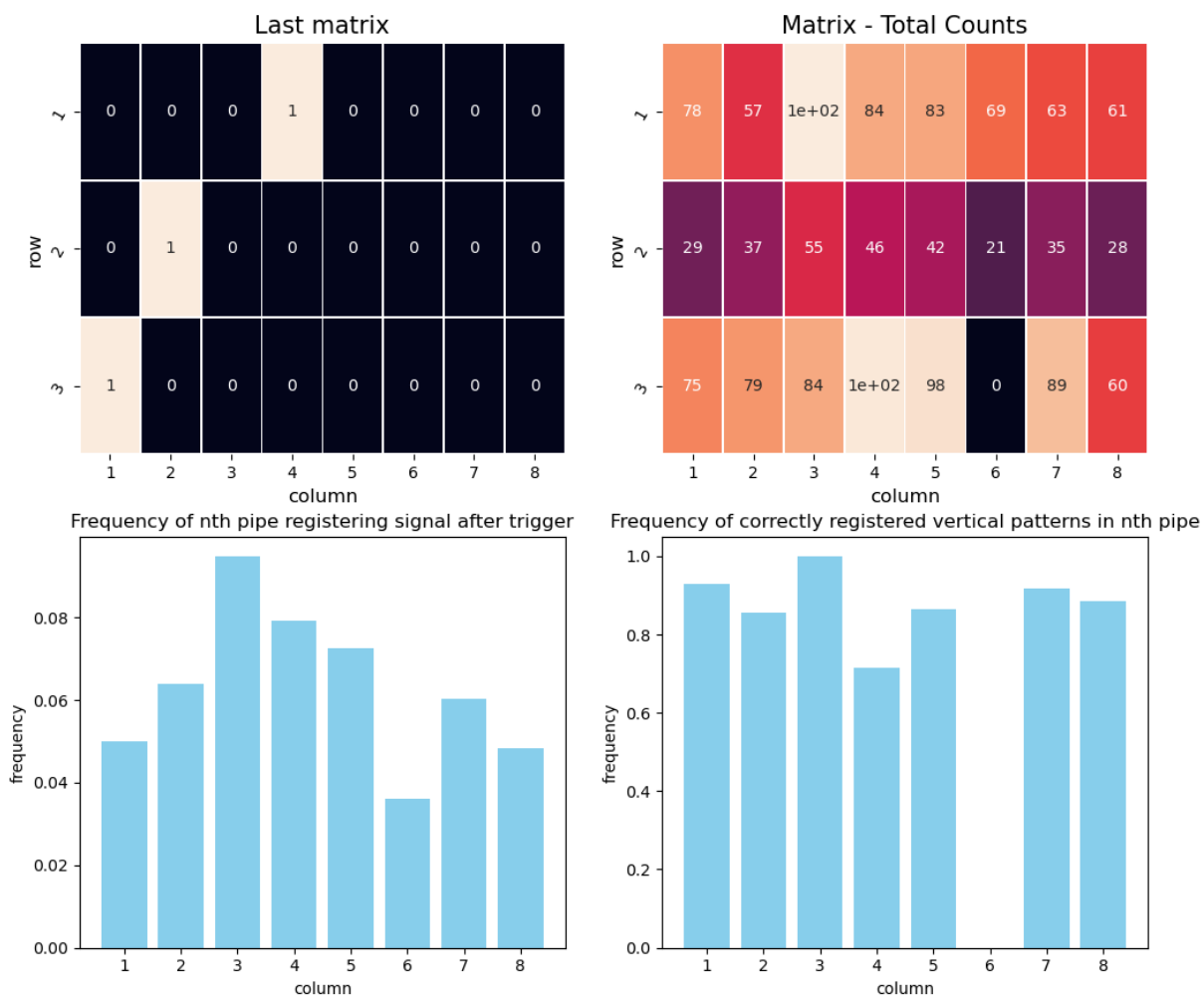
```
def animate(i):
    data = pd.read_csv('data.csv')
    x = data['x_value']
    y = data['total']
    plt.cla()
    plt.plot(x, y, label='Total counts in 1 min')
    plt.legend(loc='upper left')
    plt.tight_layout()

ani = FuncAnimation(plt.gcf(), animate, interval=1000)
```

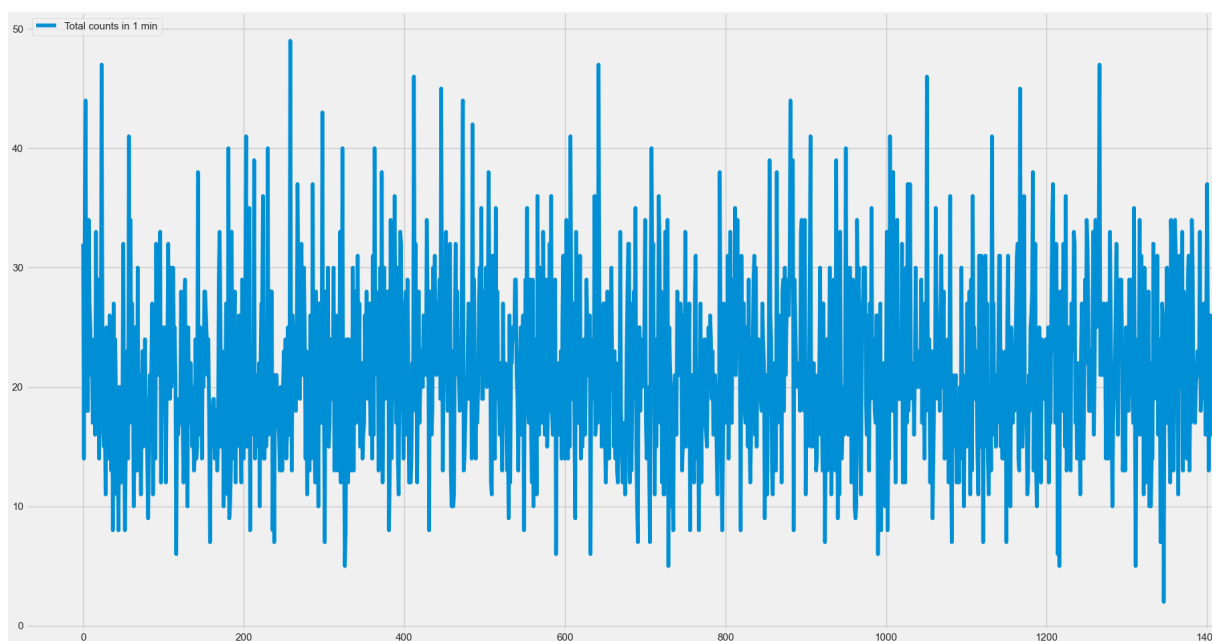
3 Wykresy

Poniżej (Rys. 3., Rys. 4) przedstawiamy rezultat uruchomionego programu do wizualizacji danych z pliku dane13.txt.

Na Rys. 3. widzimy macierz "Last matrix", na której są zapalone wszystkie trzy warstwy (każda warstwa to jeden wiersz macierzy), patrząc na macierz "Matrix - Total Counts" od razu rzuca się w oczy 6 rurka (kolumny macierzy są odpowiednikami rurek), na której dla 3 warstwy ani razu nie zapaliła się dioda, o czym mówi liczba 0. Jest to zgodne z wykresem "Frequency of correctly registered vertical patterns in nth pipe". Podczas gdy histogram wyświetlony obok "Frequency of nth pipe registering signal after trigger" pokazuje dla rurki 6-stej niezerową wartość. Natomiast Rys. 4. przedstawia wykres wszystkich zliczeń dla 1-minutowych progów zliczania.



Rysunek 3: Cztery wykresy dotyczące tego samego odczytu danych, pojawiają się w ramach jednego okienka.



Rysunek 4: Wykres wszystkich zliczeń w odstępach 1 minutowych.

4 Podsumowanie

Realizacja tego projektu przypadła na czas pandemii SARS-cov-2, co niestety spowodowało, że nasza grupa nie miała fizycznego dostępu do układu płyt wraz z licznikami Geigera-Müllera oraz układu trygerującego, które wykonały poprzednie grupy. Nie zawsze sprzyjało też łącze internetowe, na które byliśmy zdani. Niemniej jednak, napisaliśmy skrypt do wizualizacji danych wejściowych, który wyświetla ramkę z 4 wykresami: dwóch macierzy zapaleń (ostatniej i sumującej wszystkie dotychczasowe) oraz dwóch histogramów, z których lewy informuje o tym, która rurka (ze środkowej warstwy) wychwytyuje sygnał po trygerze, natomiast prawy histogram wskazuje na poprawność detekcji wertykalnych mionów (przy założeniu, że "101" to niepoprawny odczyt wynikający z błędu układu). Piątym wykresem, który otwiera się i aktualizuje w odrębnym okienku, jest wykres wszystkich wystąpień sygnałów, które miały miejsce w zadanych odstępach czasowych.

Mamy nadzieję, że nasz wkład pomoże przyszłym grupom w rozbudowaniu wizualizacji dla 8 warstw i różnych konfiguracji zapaleń.

Literatura

- [1] K. Olszowski, T. Rudnicki, W. Żurek, *Rejestracja promieniowania kosmicznego za pomocą liczników Geigera-Mullera*. 2018.
- [2] A. Widomski, K. Krajewska, M. Majczak, *ZPS - rejestrator promieniowania kosmicznego - tryger*. 2019.