USER MANUAL

Plasma Analyser Package for Aditya(PAPA) onboard ADITYA-L1

Version 1.3

January 2025

Task	Authors/Agency	Signature
Prepared by	Chemukula Mathin Yadav, Dhanya M B, R Satheesh Thampi	
Reviewed and approved by	PEER review committee	

Document Control Sheet

01	**Security Classification:** Unclassified	**02**	**Distribution:** Free
03	**Document Type:** User manual	**04**	**Document:** User manual for Plasma Analyser Package for Aditya (PAPA) onboard Aditya-L1
05	**Month and Year of Publication:** January, 2025	**06**	Collation: 1) Pages: 63 2) Figures: 12 3) Tables:10
07	**Title**: USER MANUAL for Plasma Analyser Package for Aditya (PAPA) onboard Aditya-L1		
08	**Prepared by:** PAPA Team		
09	**Reviewed by:** PEER review Team		
10	**Abstract:** This document provides the information about the data, calibration (PAPA) factors, analysis procedures and the caveats about Plasma Analyser Package for Aditya (PAPA) onboard Aditya-L1.		
11	**Keywords:** Aditya-L1, PAPA, SWEEP, SWICAR, level-0, level-1, level-2, CDF.		
12	**Supplementary element:** Nil		
13	**Issue :**1		
14	**Revision:** 3		

CHANGE RECORD

Version		Date	Changed Paragraphs	Remarks
Issue	Revision			
1	0	04/09/2024	New document	
1	1	19/12/2024	Modified based on the suggestions of peer review team.	Changed sections 7-12
1	2	26/12/2024	Modified based on the suggestions of peer review team.	Included Tables 5, 6 and 10, Included section 12, Changed ANNEXURE-1.
1	3	01/01/2025	Modified based on the suggestions of peer review team.	python script to process level-2 CDF files separately provided in Annexure-2, sample SWICAR ion mode plots using level-2 CDF files included in Annexure-2, and updated section 13 (caveats).

CONTENT

1. INTRODUCTION

This document provides the information about the data products, calibration factors, data analysis procedures and the caveats of the Plasma Analyser Package for Aditya (PAPA) payload.

2. Application

This document serves as a reference to the users of the PAPA data.

3. Background

The Plasma Analyser Package for Aditya (PAPA) is a scientific payload onboard Aditya-L1, the first Indian mission to study the Sun from the first Lagrangian point (L1) of the Sun-Earth system. The main scientific objective of the PAPA is to investigate the solar wind, the magnetised plasma (ions and electrons) flow from the Sun. Solar wind consists of charged particles (protons, alpha particles, electrons, and heavier ionized atoms), which are embedded in the magnetic field of the Sun. The average speed of solar wind is ~400 km s^{-1}. Solar wind is responsible for the anti-sunward tails of comets and the shape of the magnetosphere around the planets. The exact mechanism of solar wind formation is not known even today. The composition of solar wind contains an imprint of the composition of solar corona, modified by the processes, which lead to solar wind formation. Accurate measurement of the composition aids in separating the effects of different processes from the original makeup of the corona. PAPA is aimed to measure the composition of solar wind plasma and its energy distribution (in the range from 0.01 - 3 keV for electrons and 0.01 to 25 keV for ions) through continuous observations from the L1 point of the Sun-Earth system.

4. Scientific objectives of PAPA

PAPA is intended to study the following (primary scientific objectives):

- The solar wind electron velocity distribution function using continuous measurement of solar wind electrons in the energy range of 0.01–3 keV.
- The electron temperature anisotropy (difference between temperature parallel and

perpendicular to the local magnetic field direction) and its variability during solar transient events, such as solar flares and coronal mass ejections (CMEs).

- The strahl component of the electrons (solar wind electrons have a dominant low energy core, dominant high energy supra-thermal halo, and a beaming component aligned to the magnetic field called "strahl") and to correlate with the local magnetic field.
- The nature and variability of supra-thermal components during solar transient events.
- The study of solar wind composition, which eventually provides insight into the origin of the solar wind and the particle acceleration mechanism.

In addition, following secondary objectives are also planned:

- The study of pick-up ions and their origin (interstellar or local).
- The radial evolution of electron velocity distribution from the Sun to several solar radii in the heliosphere using multi-spacecraft data.

5. PAPA Overview

5.1. Instrument Details

To cater to the scientific objectives, PAPA consists of two sensors.

(1) Solar Wind Ion Composition Analyser (SWICAR)

(2) Solar Wind Electron Energy Probe (SWEEP)

Both SWICAR and SWEEP are cylindrical electrostatic analysers (Fig. 1). SWEEP measures only electrons. SWICAR has two basic modes of operation – ion mode where ion parameters are measured and electron mode where electron parameters are measured. These two modes in SWICAR are mutually exclusive. The basic measurement involves the detection of ions and electrons over a wide energy range, angular coverage (different directions in space), and mass range (for ions). This necessitates the use of high voltages at various parts of the sensors. The electronics consist of high voltage programmable power supply (HVPPS), front end electronics (FEE), and the PAPA processing unit (PPU). HVPPS generates all the voltages required for the operation of the sensors. The entire sensor operation is controlled by the PPU. PPU not only commands and controls the sensor operations, but also interfaces

with the spacecraft.

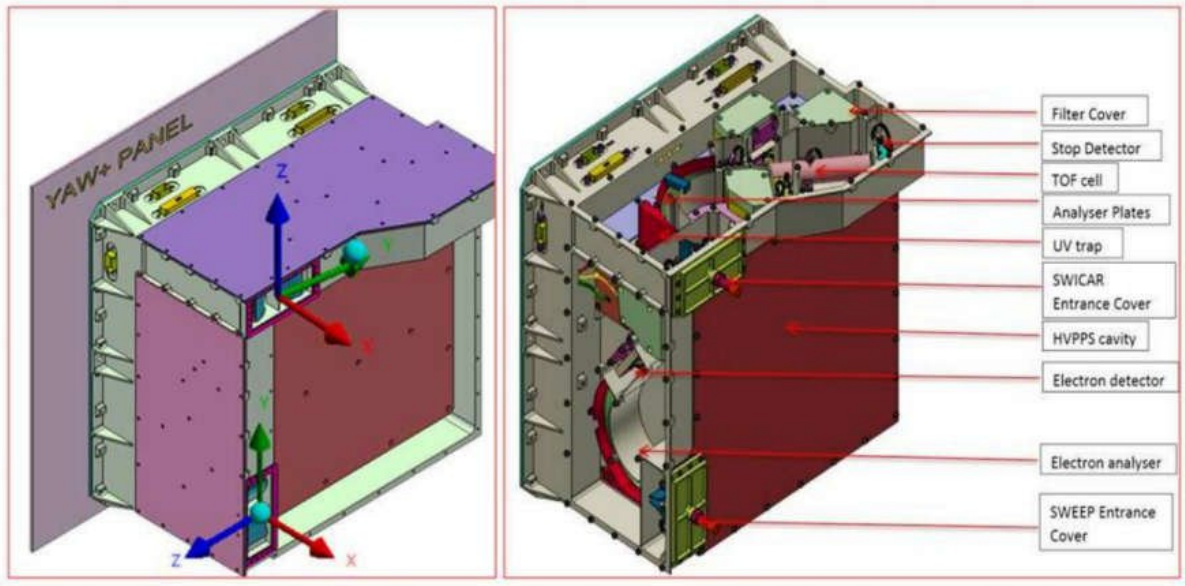

Figure 1: The schematic diagram of PAPA mounted on the spacecraft (left) and the sensors SWICAR and SWEEP are shown with different parts indicated (right). PAPA is mounted on the +Yaw panel of the spacecraft and always looks at the Sun.

5.2. Default Operation Setting Of SWICAR

The default setting of SWICAR ion mode consist of 32 direction bins to cover the total field of view (FOV) of 20°(elevation)× 124° (azimuth), such that one angular pixel size is ~20° (elevation) × 4° (azimuth). The energy range of 10 eV to 25 keV will be covered in 32 steps (energy bins) and the mass range of 1-60 amu is covered in 32 bins of TOF bins. The energy values corresponding to the bins are logarithmically separated. The basic measurement cycle (integration time), which is the time taken to scan in one energy bin and one direction bin is 62.5 ms. Hence one complete measurement of SWICAR ion mode takes 64 s (major cycle).

The default setting of electron mode consists of 32 direction bins to cover the total FOV of 20°(elevation)× 124° (azimuth),such that one angular pixel size is ~20° (elevation) × 4° (azimuth). The energy range of 10 eV to 3 keV will be covered in 16 steps (energy bins). The energy values corresponding to the bins are logarithmically separated. The basic measurement cycle (integration time), which is the time taken to scan in one energy bin and

one direction bin is 62.5 ms. Hence one complete measurement of SWICAR electron mode takes 32 s (major cycle).

In the default operation setting, SWICAR operates in ion mode and electron mode in alternate major cycles. The default binning parameters for SWICAR ion and electron mode are listed in Table 1.

Table 1.PAPA instrument specifications for both the ion and electron mode operations of both SWICAR and SWEEP sensors.

Parameter	SWICAR Ion Mode	SWICAR Electron Mode	SWEEP
Field of View	20° (elevation) × 124° (azimuth)	20° (elevation) × 124° (azimuth)	20° × 124°
Angular pixel resolution	20° × 8° (< 80°) 20°×12° (80° - 124°)	20° × 8° (< 80°) 20°× 12° (80° - 124°)	20° × 8° (< 80°) 20°× 12° (80° - 124°)
Direction bins (default)	32	32	32
Energy Range	10 eV - 25000 eV	10 eV - 3000 eV	10 eV – 3000 eV
Energy resolution (ΔE/E)	4-7.5 %	1.5 – 9.0 %	1.5-9.0 %
Energy bins (default)	32	16	16
Mass range	Amu 1, 2, 3 and 4 separated at all energies. Rest of the masses identifiable as groups.	NA	NA
Mass bins (default)	32	NA	NA
Basic measurement cycle (ms)	62.5	62.5	62.5
Major cycle (s)	64	32	32

5.3. Default Operation Setting of SWEEP

The default setting of SWEEP consists of 32 direction bins to cover the total FOV of

20°(elevation)× 124° (azimuth), such that one angular pixel size is ~20° (elevation) × 4° (azimuth). The energy range of 10 eV to 3 keV will be covered in 16 steps (energy bins). The energy values corresponding to the bins are logarithmically separated. The basic measurement cycle (integration time), which is the time taken to scan in one energy bin and one direction bin which is 62.5 ms. Hence one complete measurement of SWEEP takes 32 s (major cycle). The default binning parameters for SWEEP mode are listed in Table 1.

Apart from default operation setting, operation settings with different combinations of direction, energy and mass bins will be used in the actual operation of PAPA at L1 point, as described in the subsequent section.

5.4. Programmable operation settings

In the science observation phase, settings other than default mode will be required. During the science observations, it will be required to scan over a set of energy bins (programmable) which is a subset of the maximum allowed energy bins. The programmable feature is such any consecutive 32, 16, and 8 energy bins can be selected from the energy table for SWICAR ion mode, any consecutive 16, 8, and 4 energy bins can be selected from the energy table for SWICAR electron mode and any consecutive 16, 8, and 4 energy bins can be selected from the energy table for SWEEP. These parameters (such as energy bin)) are referred to as operating parameters. Raw mode is a non-process mode for time-of-flight data, where the raw TOF values are transmitted to ground without any binning; this mode will be used as part of engineering runs and for any debugging. In addition, a single Energy, single direction mode is also available to be exercised for calibration and special operations. Hence this mode can be termed as 'calibration mode’. There is one more mode called 'fine mode', where the default energy bins are further divided into more finely separated bins. For SWICAR ion mode the default 32 energy bins are divided into 68 energy bins and for SWICAR electron, SWEEP modes the default 16 energy bins are divided into 49 energy bins. In this mode also same programmable energy selection feature is available. The programmable feature is such that any consecutive 32/16/8 energy bins can be selected out of the68 bins from the fine mode energy table for SWICAR ion mode and any consecutive

16/ 8/ 4 energy bins can be selected out of the 49 bins from fine mode energy table for SWICAR electron/SWEEP.

6. PAPA Data Flow and Level Definitions for Science Data Products

6.1. PAPA Data Flow

The data flow from PAPA to the spacecraft, transmission to the ground station, and the various processing carried out at ground station are illustrated in figure 2. On the spacecraft, PPU acquires the data from SWEEP and SWICAR sensors, packetizes the same and transfer the data to the baseband data handling (BDH) unit. From the BDH, data in the space packet format is sent to solid state data recorder (SSR). The data gets transmitted to ground and received at the Indian deep space network (IDSN). The payload data acquisition system (PACQ) at IDSN acquires the data and separates the PAPA data based on the application ID (APID, which is 230d for PAPA). This generates .raw data file for PAPA at the Indian Space Science Data Centre (ISSDC), .raw file is fed to the level-0 processing area. For each raw file, Level-0 software generates a.tar file that consist of payload science data file, OBT to UTC time correlation table, spacecraft ephemeris and attitude information as SPICE kernels (bsp and bc files) and house-keeping (HK) data files. These files will be bundled into a .tar file and received at the PAPA console in payload operation centre (POC) at SPL for further processing by the payload team. The level-0 data files are further processed at the POC to generate the level-1 and level-2 data products.

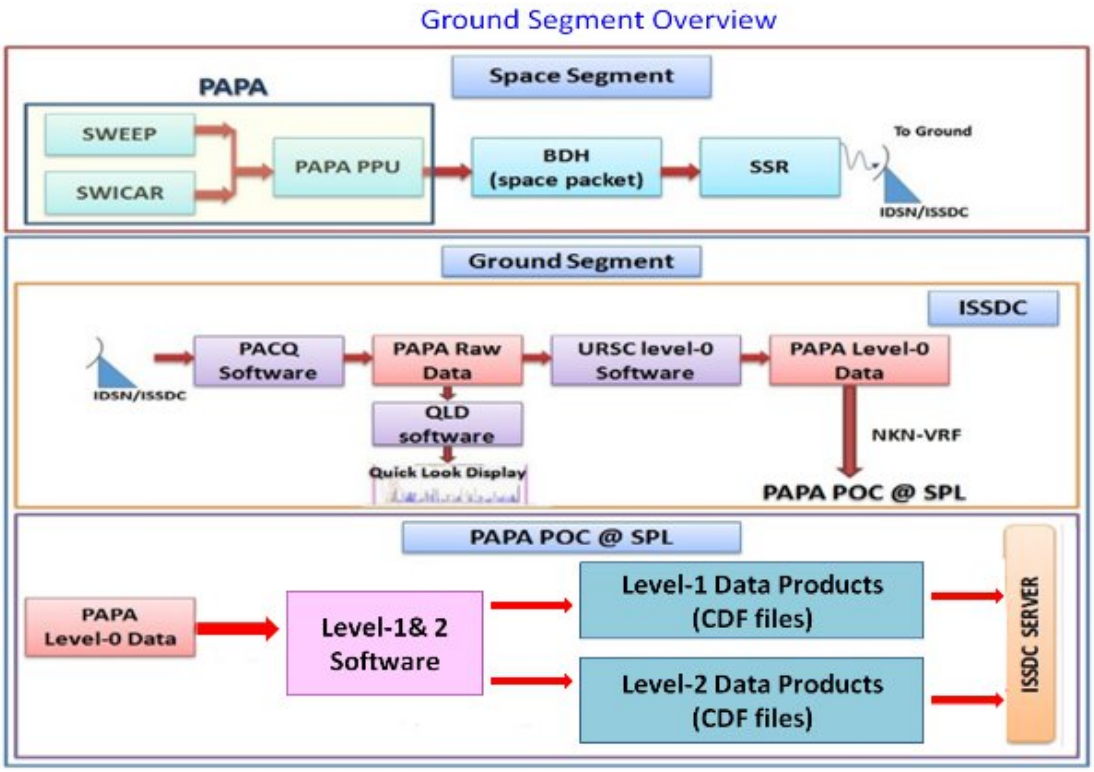

Figure 2: Illustration of data flow Diagram for PAPA

6.2 Level Definitions for PAPA Science Data Products

- **Level-0:** Payload separated, Time-Tagged data with ephemeris, Attitude, TCT (Time Correlation Table for OBT to UTC correlation), and HK data (telemetry data extract from low bit telemetry (LBT)) generated at ISSDC and provided to the POC as a tar file by the ISSDC Team.
- **Level-1:** Sensor separated data, segregated into meaningful observations, time tagged, and error checked data with science parameter as count rates (counts/s). Data products in common data format (CDF). The level-1 data products are generated at PAPA POC and pushed to ISSDC for dissemination to the scientific community.
- **Level-2:** Science parameter in differential number flux units (cm^{-2} s^{-1} sr^{-1} eV^{-1}). Data products are in CDF. The level-2 data products are generated at PAPA POC and pushed to ISSDC for dissemination to the scientific community.

7. Level-1 products and file naming convention

The data from both the electron sensor modes are identical for both SWEEP and SWICAR electron mode but they are aligned perpendicular to each other. In the default mode of operation, the energy range for electron modes is from 10 eV to 3000 eV that are divided in 16 logarithmic bins (Table 2). For each energy bin, the data would be acquired in 32 direction bins (Table 3). Hence, one complete scan will have 16 × 32 counts rates. These count rates are stored in CDF format and made available as Level-1 data products. Subsequently, the data products are disseminated to ISSDC for public release.

Table 2: Energy bins and the corresponding energies for SWEEP and SWICAR electron mode

Energy bin (number)	Energy (eV)
1.	10
2.	14.6
3.	21.4
4.	31.3
5.	45.8
6.	66.9
7.	97.9
8.	143.2
9.	209.5
10.	306.4
11.	448.1
12.	655.5
13.	958.7
14.	1402.3
15.	2051.1
16.	3000

In the default mode of operation, SWICAR ion mode operates in the energy range from 10 eV to 25000 eV sampled in logarithmic 32 bins (Table 4). For each energy bin, the data

would be collected in 32 direction bins. Hence, one complete scan will have 1024 values of count rates (32 energy bins × 32 direction bins). The count rates are further binned to 32 time of flight bins (TOF), so for one complete scan will have 32768 values of TOF binned count rates (32 energy bins x 32 direction bins x 32 TOF bins).The mapping of TOF bins to mass values is provided in Table 5.

Table 3: Direction bins and the corresponding values (angles) in the sensor frame of reference for all the three sensor modes. The angles are measured in the azimuth plane with the middle pixel (centre viewing direction) as reference (zero degree), which has direction bin number 17. Other angles (of particle entry) are measured on either side of middle pixel (positive and negative angles).

Direction bin (number)	Angle (degrees)	Direction bin (number)	Angle (degrees)
1.	64	17.	0
2.	60	18.	-4
3.	56	19.	-8
4.	52	20.	-12
5.	48	21.	-16
6.	44	22.	-20
7.	40	23.	-24
8.	36	24.	-28
9.	32	25.	-32
10.	28	26.	-36
11.	24	27.	-40
12.	20	28.	-44
13.	16	29.	-48
14.	12	30.	-52
15.	8	31.	-56

16.	4	32.	-60

Table 4: Energy bins and their values for SWICAR ion mode

Energy bin (number)	Energy (eV)	Energy bin (number)	Energy (eV)
1.	10	17.	567.3
2.	12.9	18.	730.1
3.	16.6	19.	939.7
4.	21.3	20.	1209.5
5.	27.4	21.	1556.8
6.	35.3	22.	2003.7
7.	45.5	23.	2578.9
8.	58.5	24.	3319.3
9.	75.3	25.	4272.3
10.	96.9	26.	5498.9
11.	124.8	27.	7077.6
12.	160.6	28.	9109.5
13.	206.7	29.	11724.8
14.	266	30.	15091
15.	342.4	31.	19423.6
16.	440.7	32.	25000

Table 5 : TOF bins and their corresponding mass ranges

TOF bin no	Bin start mass (amu)	Bin end mass (amu)
1	1	1
2	2	2
3	3	3

4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20
21	21	21
22	22	22
23	23	23
24	24	24
25	25	25
26	26	30
27	31	35
28	36	40
29	41	45
30	46	50
31	51	55
32	56	60

7.1. File naming Convention for Level-1 data products

The Level-1 data products will be mode corresponding CDF files. The file name convention will be:

<Payload_ID>_<sensor_ID>_<sensor_mode>_<energy_bin_number>_<GRT>_<OBS_ID>_<Le

vel0_tar_file_version>_<Start_time>_<level>_<version>.CDF

<Payload_ID>: PPA

<sensor_ID>: SWR/ SWP (SWR: SWICAR/ SWP: SWEEP)

<sensor_mode>: ion/ele

<energy_bin_number>: Exx (xx can have values 8, 16 and 32 for SWICAR ion mode; 4,8,16 for SWICAR electron mode and SWEEP).

<GRT>: Ground Receive time of the science data

<OBS_ID>:UNP_9999_999999 (UNP: Proposal type, 9999: Proposal cycle id, 999999: Obs id counter).

<Level0_tar_file_version>: <L0_Vm_n>, where m_n can be 1_0 and above

<Start_time>: YYYYDDMMHHMMSS (start time of science data)

<level>: Ly (L1)

<version>: Vz_r (data products version starts with V1_0)

Examples are

PPA_SWR_ion_Exx_<GRT>_<OBS_ID>_L0_V1_0_Start_Time_L1_V1_0.cdf

PPA_SWR_ele_Exx_<GRT>_<OBS_ID>_L0_V1_0_Start_Time_L1_V1_0.cdf

PPA_SWP_ele_Exx_<GRT>_<OBS_ID>_L0_V1_0_Start_Time_L1_V1_0.cdf

PPA_SWP_ele_E16_23350004306005_UNP_9999_999999_L0_V1_1_20231214235958_L1_V1_0.cdf

7.2 Leve-1 data products CDF file content

For SWEEP and SWICAR electron Level-1 products (Table 6), CDF file contains three variables:

1) Epoch, 2) Energies and 3) Count_rates

Here each element of Epoch corresponds to the start time of a scan in co-ordinated

universal time (UTC), Energies corresponds to the energy bins, and Count_rates are the counts recorded per second for a given energy bin and direction bin at a particular time (Count_rates is a 3D array)

Count_rates = counts/sec (Epoch, Energies, directions)

For SWICAR ion Level-1 products (Table 6), CDF file contains four variables 1) Epoch, 2) Energies, 3) Count_rates and 4) TOF_binned_count_rates

Here each element of Epoch corresponds to the start time of a scan in UTC, Energies corresponds to the energy bins, Count_rates are the counts recorded per second in each energy bin and direction bin at a particular time (Count_rates is a 3D array) and TOF_binned_counts are the counts recorded per second for a given energy bin, direction bin and mass bin at a particular time (TOF_binned_counts is a 4D array)

Count_rates = counts/sec (Epoch, Energies, directions)

TOF_binned_counts = counts/sec (Epoch, Energies, directions, mass bins)

Table 6:PAPA Level-1 data products CDF file content

Sl.No	Sensor mode	CDF file content
1	SWEEP electron	1. Epoch (observation time in UTC) 2. Energies (energy values array) 3. Count_rates (counts/sec corresponding to the Epoch, Energies and directions)
2	SWICAR electron	1. Epoch (observation time in UTC) 2. Energies (energy values array) 3. Count_rates (counts/sec corresponding to the Epoch, Energies and directions)
3	SWICAR ion	1. Epoch (observation time in UTC) 2. Energies (energy values array) 3. Count_rates (counts/sec corresponding to the Epoch, Energies and directions)

		4.TOF_binned_count_rates(counts/sec corresponding to the Epoch, Energies, directions and TOF bins)

8. Level-2 products generation and the Level-2 products file naming convention

8.1 Conversion from Level-1 Data to the Level-2:

The Level- 1 output CDF files will be the inputs for generating Level-2 outputs. For generating Level-2 outputs the count rates in each mode are converted to differential number flux (in units of cm^{-2} sr^{-1} eV^{-1} s^{-1}) which is differential in both energy and in direction.

The differential number flux $Df_{i,j}$ (cm^{-2} sr^{-1} eV^{-1} s^{-1}) seen by each angular pixel *j* and energy bin *i* with an energy width of ΔE_i [eV] is

$$Df_{i,j} = \frac{C_{i,j}}{G_{i,j}.E_i}$$

$C_{i,j}$: counts rates for an energy bin (i) and direction bin (j).

$G_{i,j}$: Geometric factor including all the efficiency for each angular pixel j at energy step i

$$G_{i,j} = K_{i,j}.A_i.\Omega_i.\left(\frac{\Delta E_i}{E_i}\right)$$

E_i: Center energy of the energy bin

$\Delta E_i/E_i$: Energy resolution at each energy bin

$K_{i,j}$: Efficiency at each angular pixel j and energy bin i

A_i: The effective area

Ω_i: Effective solid angle at energy bin i

The geometric factor for different energy ranges for ions and electrons, based on the laboratory calibration are provided in table 7. The values provided are for direction bin number 10/11 for SWEEP/SWICAR electron mode and valid for all direction bins in

SWICAR ion mode. For rest of the direction bins for electron measurements, geometric factor relative to this direction bin (relative geometric factor) is used, which is listed in Table 8-9.Usage of relative geometric factor is known practice in plasma analysers.

Table 7: Geometric factor values for all the three modes (SWEEP electron, SWICAR electron and SWICAR ion) for all directions in SWICAR-ion and for electron modes based on direction bin multiplication factors given in Tables 8-9.

Mode	Energy (eV)	Geometric Factor (cm^2sreV/eV)
SWICAR Ion	10 - 567.25	3.72×10^{-6}
	730.1 – 2578.9	5.87×10^{-6}
	3319.3 - 25000	1.0×10^{-8}
SWICAR Electron	14.63 to 97.91	4.54×10^{-5}
	143.21 to 3000	6.75×10^{-5}
SWEEP- Electron	14.63 to 66.94	2.34×10^{-4}
	97.91 to 306.39	6.86×10^{-4}
	448.14 to 3000	8.15×10^{-5}

Table 8:SWEEP direction bin relative geometric factor values

Direction bin	**Relative Geometric Factor**		
	14.63 to 66.94 eV (E_{bin} 2 to E_{bin} 6)	**97.91 to 306.39 eV (E_{bin} 7 to E_{bin} 10)**	**448.14 to 3000 eV (E_{bin} 11 to E_{bin} 16)**
1.	0.469384	0.466023	0.413427
2.	0.576313	0.308683	0.506925
3.	0.66932	0.301506	0.595301
4.	0.751929	0.386198	0.678715
5.	0.825351	0.517365	0.756234

6.	0.888981	0.660903	0.826538
7.	0.940864	0.792773	0.887726
8.	0.978441	0.897616	0.937686
9.	0.998932	0.967428	0.974307
10.	1	1	0.995601
11.	0.980234	0.997716	1
12.	0.939695	0.966098	0.986432
13.	0.880498	0.912497	0.95458
14.	0.807479	0.844617	0.905147
15.	0.728416	0.76921	0.839982
16.	0.654994	0.69069	0.762235
17.	0.568415	0.53478	0.631535
18.	0.533719	0.459824	0.586414
19.	0.508654	0.414279	0.550145
20.	0.513691	0.440577	0.566142
21.	0.540287	0.507604	0.6148
22.	0.581008	0.591744	0.680651
23.	0.629304	0.675784	0.751747
24.	0.679676	0.747754	0.819411
25.	0.727529	0.800402	0.877572
26.	0.769398	0.829611	0.922622
27.	0.802644	0.833827	0.952556

28.	0.825714	0.812773	0.966828
29.	0.838065	0.766625	0.965878
30.	0.839979	0.694981	0.95048
31.	0.832927	0.595709	0.921478
32.	0.819206	0.464115	0.879314

* Energy bin 1 (10 eV) has been removed because of spacecraft charging effects.

Table 9: SWICAR electron direction bin relative geometric factor values

Direction bin	**14.63 to 97.91 eV (E_{bin} 2 to E_{bin} 7)**	**143.21 to 3000 eV(E_{bin} 8 to E_{bin} 16)**
1.	0.809	0.5553
2.	0.8377	0.5869
3.	0.8563	0.5902
4.	0.8716	0.612
5.	0.879	0.6316
6.	0.8914	0.6424
7.	0.9001	0.6567
8.	0.8923	0.6705
9.	0.8724	0.6753
10.	0.8193	0.671
11.	0.7567	0.66
12.	0.672	0.6272
13.	0.6025	0.5927
14.	0.5659	0.5602
15.	0.5257	0.4798
16.	0.5082	0.477
17.	0.4899	0.4778

18.	0.4619	0.4815
19.	0.4842	0.4812
20.	0.6055	0.4917
21.	0.7066	0.4786
22.	0.8045	0.587
23.	0.8548	0.7067
24.	0.9453	0.778
25.	0.9783	0.8685
26.	0.9823	0.9441
27.	0.9915	0.9617
28.	0.9994	0.9954
29.	1	0.9942
30.	0.993	1
31.	0.9855	0.9702
32.	0.9871	0.9551

*Energy bin 1 (10 eV) has been removed because of spacecraft charging effects.

Note: In order to derive the geometric factor matrix for all the energy and corresponding direction bins for the electron modes, the user has to multiply the values given in Table 7 (both SWEEP and SWICAR electron GF values) with the relative geometric factor values given in Tables 8-9 for both SWEEP and SWICAR electron directions. For SWICAR ion for all direction bins the same geometric factor values given in Table 7 is applicable.

8.2 File naming convention for Level- 2 data products

The file name convention of Level-2 products is similar to Level-1 products the change being the products level will be L2 instead of L1:

Example file names:

PPA_SWR_ion_Exx_<GRT>_<OBS_ID>_L0_V1_0_Start_Time_L2_V1_0.cdf

PPA_SWR_ele_Exx_<GRT>_<OBS_ID>_L0_V1_0_Start_Time_L2_V1_0.cdf

PPA_SWP_ele_Exx_<GRT>_<OBS_ID>_L0_V1_0_Start_Time_L2_V1_0.cdf

8.3 Level-2 data products CDF file content

Level-2 products have the science parameter in differential number flux units and the remaining content will be similar to Level-1 products

For SWEEP and SWICAR electron Level-2 products (Table 10), CDF file contains three variables:

1) Epoch, 2) Energies and 3) Differential_number_flux

Here each element of Epoch corresponds to the start time of a scan in UTC, Energies corresponds to the energy bins and differential number fluxes for a given energy and direction at a particular time (differential number fluxes is a 3D array)

Differential_number_flux = differential number flux (epoch, energies, directions)

For SWICAR ion Level-2 products (Table 10), CDF file contains four variables:

1) Epoch, 2) Energies, 3) Differential_number_flux and 4) TOF_binned_Differential_number_flux

Here each element of Epoch corresponds to the start time of a scan in UTC, Energies corresponds to the energies scanned, Differential_number_flux in each energy and direction at a particular time (Differential_number_flux is a 3D array) and TOF_binned_Differential_number_flux in each energy, direction and mass bin at a particular time (TOF_binned_Differential_number_flux is a 4D array)

Differential_number_flux = Differential_number_flux (Epoch, Energies, directions)

TOF_binned_Differential_number_flux = Differential_number_flux (Epoch, Energies, directions, mass bins)

Table 10:PAPA Level-2 data products CDF file content

Sl. No	Sensor mode	CDF file content
1	SWEEP electron	1) Epoch (observation time in UTC) 2) Energies (energy values array)

		3) Differential_number_flux (counts/sec corresponding to the Epoch, Energies and directions)
2	SWICAR electron	1) Epoch (observation time in UTC) 2) Energies (energy values array) 3) Differential_number_flux (counts/sec corresponding to the Epoch, Energies and directions)
3	SWICAR ion	1) Epoch (observation time in UTC) 2) Energies (energy values array) 3) Differential_number_flux (counts/sec corresponding to the Epoch, Energies and directions) 4) TOF_binned_Differential_number_flux(counts/sec corresponding to the Epoch, Energies, directions and TOF bins)

9. PAPA webpage overview of the PRADAN site

In the PRADAN site, there will be a data products table in the PAPA webpage as shown in Figure. 3. The table contains seven columns. The first column corresponds to the serial number of the data products, the second column shows preview image of the CDF file , third column corresponds to the file name of the CDF file (based on the file name which the user can identify the data product levels as well as the mode that is described as in the Section 8), The fourth column corresponds to the start time of the CDF file, the sixth column corresponds to the end time of the data, the sixth column corresponds to the file size, and the seventh column corresponds to the meta info of data file. As part of the preview image, time series of total count rates (count rates summed over in all the energy and direction bins) was generated for level-1 and time series of total differential number flux (differential number flux summed over in all the energy and direction bins)was generated for level-2 products.

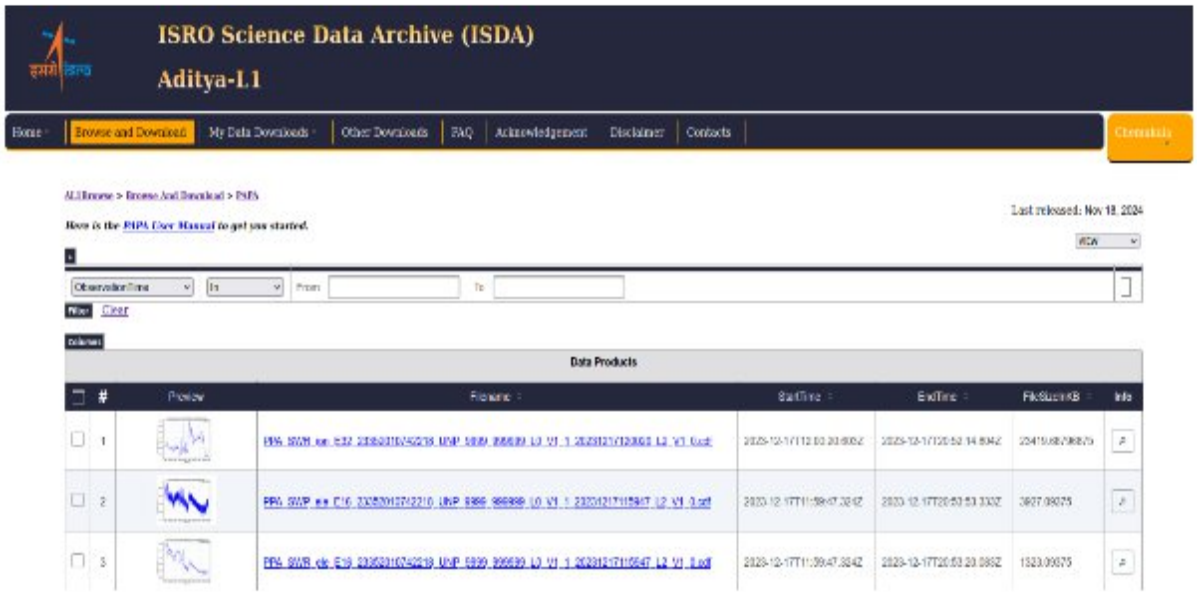

Figure 3: Screenshot of the PAPA page in the PRADAN website.

10. Ancillary Data

The ancillary data is available in the form of SPICE kernels. SPICE stands for Spacecraft, Planet, Instrument, C-matrix, Events and kernels are data files that contain information spacecraft ephemeris, attitude, coordinate systems, geometry etc.The kernel that contains the spacecraft ephemeris information hasan extension '.bsp' and the kernel that contains spacecraft orientation information has an extension '.bc'. These are binary data files. The information on the spacecraft frame is contained in the spacecraft frames kernel, which is a text file with an extension '.tf' and similar information on instrument frames are available in the instrument kernel, which is a text file with extension '.ti'. The required ancillary parameters such as location of observation, altitude, latitude/longitude, spacecraft position, rotation angles etc can be derived from the SPICE kernels using the SPICE toolkit that is freely available for download at the web page of NASA's Navigation and Ancillary Information Facility (NAIF) (https://naif.jpl.nasa.gov/naif/). SPICE Toolkit consist of application program interfaces (APIs) that users can incorporate in their own programs to read the SPICE kernels and derive ancillary parameters. Using the APIs available in the SPICE toolkit to extract the required parameters in the preferred coordinate system, such as J2000, geocentric solar ecliptic (GSE), geocentric solar magnetospheric (GSM) coordinates

etc. Detailed tutorial on the usage of SPICE toolkit as well as kernels are available at the NAIF website(https://naif.jpl.nasa.gov/naif/).

The required SPICE kernels for Aditya-L1 are available in PRADAN website under ‘spice archive’.

11. PAPA CDF data products reading procedure

Common Data Format (CDF) is a self-describing data format for the storage of scalar and multidimensional data in a platform- and discipline-independent way. Scientific data management package (CDF Library) allows application developers to manage these data arrays. Transparent access to the data and the meta-data are through Application Programming Interfaces (APIs). It has built-in support for data compression (gZip, RLE, Huffman) and automatic data decompression, and checksum. It supports large files (> 2G-bytes) and more details can be found at “https://cdf.gsfc.nasa.gov/”.

To read CDF files, the user should install the CDF library. The latest version of the CDF library is found at “https://cdf.gsfc.nasa.gov/html/sw_and_docs.html”. Once the CDF library is installed, depending on the programming language the user has to configure the patch. For Python programming language the user has to install Spacepy package. The user has to use following commands to read the CDF file.

```
from spacepy import pycdf

cdf = pycdf.CDF('/path/to/file.cdf')

print(cdf)
```

More examples on reading and writing to a CDF file can be found at “https://spacepy.github.io/pycdf.html”

For PAPA CDF data products a python code to open, read and generate Total count rates time-series plot, Energy-Time spectrogram plot, Direction-Time spectrogram plot and Energy-Direction spectrogram plot is included in Annexure-1 as well as in the “*Other Downloads*” area of the PRADAN website

12. PAPA Quality flag

The quality of PAPA data is defined based on the packet losses with respect to the level-0 data file received at the POC. A quality flag that has value ranging from 10 to 1, with 10 being the best, has been defined for the purpose. The data with quality flag value 10 have less than 1% packet losses, quality flag value 9 have less than 5% packet losses, quality flag value 8 have less than 10% packet losses, quality flag value 7 have less than 20% packet losses, and quality flag value 6 have greater than or equal to 20% packet losses. Mostly, PAPA data have < 1% packet losses such that the data quality flag can be found to 10 in most of the data products. Also, data losses greater than 20% is highly unlikely and hence the quality factor has been defined up to level6.

13. Important Caveats

1) The energy bin corresponding to 10 eV in SWEEP sensor is contaminated possibly by the photo electrons. So, the data should be used with caution.
2) All the three PAPA modes (SWEEP-electron, SWICAR-electron and SWICAR -ion) performed well till 25th December 2023 and after which the detector degradation started for electron detectors. Quality checks on these data sets shows that the electron data is reliable during the period from 15 to 25th December 2023 only.After06th January 2024, the detector degradation has affected all the data sets significantly any changing the bias voltage the ion data (SWICAR-ion mode) could be able to revive and it is ready for use from 01st April 2024 onwards and is available till date.
3) Both the SWEEP and SWICAR electron data shows reduction in counts due to detector gain degradation and even after bias voltage changes (from 2 kV to 2.4 kV) the trend continued due to which the data as such is not fit for public release. However efforts are ON to re-generate the data sets and if it succeeds then it will be released.
4) The high ion count rates/differential number flux observed in the direction bin number zero (64°) in SWICAR ion mode for a few data sets have to be used with caution.
5) In the SWICAR ion mode, the high count rates/differential number flux observed in the TOF binned counts for the TOF bins 30, 31, and 32 are a consequence of binning and this

data should not be used for any scientific interpretations.

6) The processor upsets occurred multiple times (likely due to solar energetic particle events) resulted in processor hanging and to revive the same power recycling was carried out couple of times and which resulted in discontinuity of PAPA data sets.

Contacts:

1) Dr. SatheeshThampi R, SPL

satheesh_thampi@vssc.gov.in

2) Dr. Dhanya M. B, SPL

mb_dhanya@vssc.gov.in

3) ChemukulaMathinYadav, SPL

mathin_chemukula@vssc.gov.in

References:

1) P. Janardhan, SantoshVadawale, BhasBapat, K. P. Subramanian. Chakrabarty, Prashant Kumar, AveekSarkar, NanditaSrivastava,R. SatheeshThampi, Vipin K. Yadav, M. B. Dhanya, Govind G. Nampoothiri,J. K. Abhishek, Anil Bhardwaj and K. Subhalakshmi (2017), Probing the heliosphere using in situ payloadson-board Aditya-L1, Current Science, 113 (4), 620-624.

2) R. Satheesh Thampi, J. K. Abhishek, Dersana Sasidharan, Ganesh Varma, Vijay Kumar Sen, Sabooj Ray, M. B. Dhanya, Ullekh Pandey, Shishir Kumar S. Chandra, J. B. Akash, A. N. Aneesh, Tincy M. Wilson, S. Naresh, Neha Naik, Mathin Chemukula Yadav, V. Venkataraman, Rosmy John, R. Manoj, Govind G. Nampoothiri, Pritesh Meshram, Maria George, Vinitha Ramdas, Ginju V. George, Anju M. Pillai, Arjun Dey, Surajit Das, G. Subha Varier, G. Sajitha, Sheeja Mathews, P. Pradeep Kumar, G. R. Nisha, Amarnath Nandi, B. Sundar, R. Sethunadh, A. Rajendra, H. Saleem and A. K. Abdul Samad, Plasma-Analyzer Package for Aditya (PAPA) Onboard the Indian Aditya-L1 Mission, Solar Physics, doi:10.1007/s11207-024-02414-8.

3) For SPICE Toolkit and tutorials: https://naif.jpl.nasa.gov/naif/.

ANNEXURE-1

Python code to read the CDF files (level-1) and generate plots

The sample python code to open and read the CDF file (level-1) to generate time series curve of total count rates, energy-time spectrogram, direction-time spectrogram and energy-direction spectrogram for SWICAR ion mode, electron mode, SWEEP and TOF bin count rates spectra for SWICAR ion mode are provided herewith. Please include the relevant paths (code line no.s: 14, 17 and 20) for generating the plots and if any mode plots are not required then comment the corresponding lines (for SWEEP: code line no 275, for SWICAR electron: code line no 276, for SWICAR ion: code line no 277). The same code is available in PRADAN under 'other downloads'.

The following files are used for generating the plots and plots for SWEEP electron mode are provided here for reference (figures 1 to 4).

SWEEP electron:

PPA_SWP_ele_E16_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L1_V1_0.cdf

SWICAR electron:

PPA_SWR_ele_E16_23352005155669_UNP_9999_999999_L0_V1_1_20231217000043_L1_V1_0.cdf

SWICAR ion:

PPA_SWR_ion_E32_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L1_V1_0.cdf

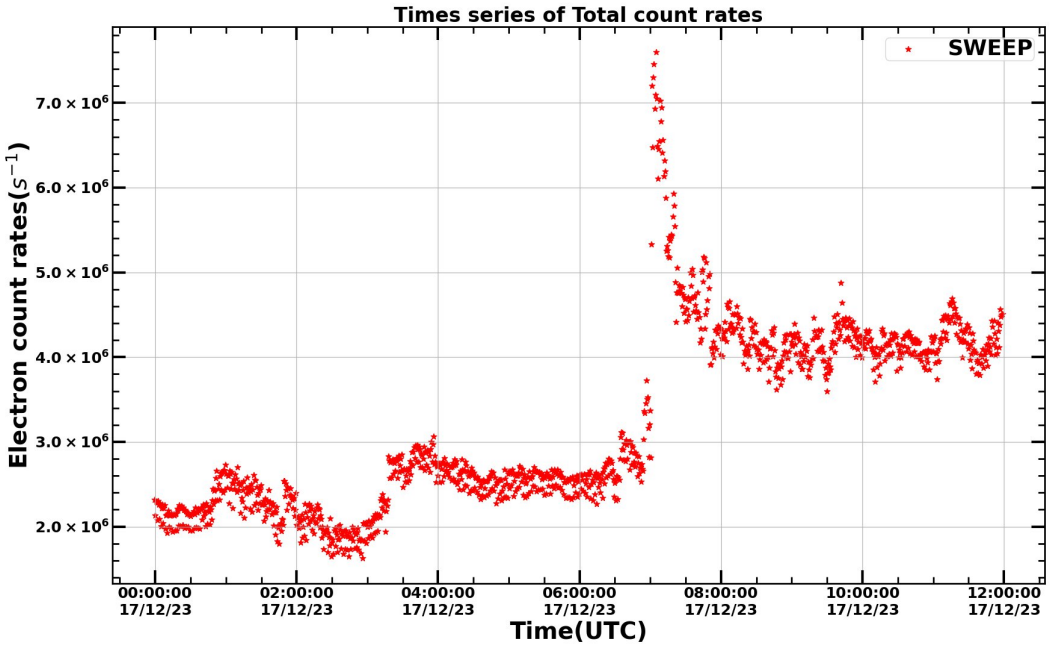

Figure 1: Time series of Total count rates for SWEEP electron mode

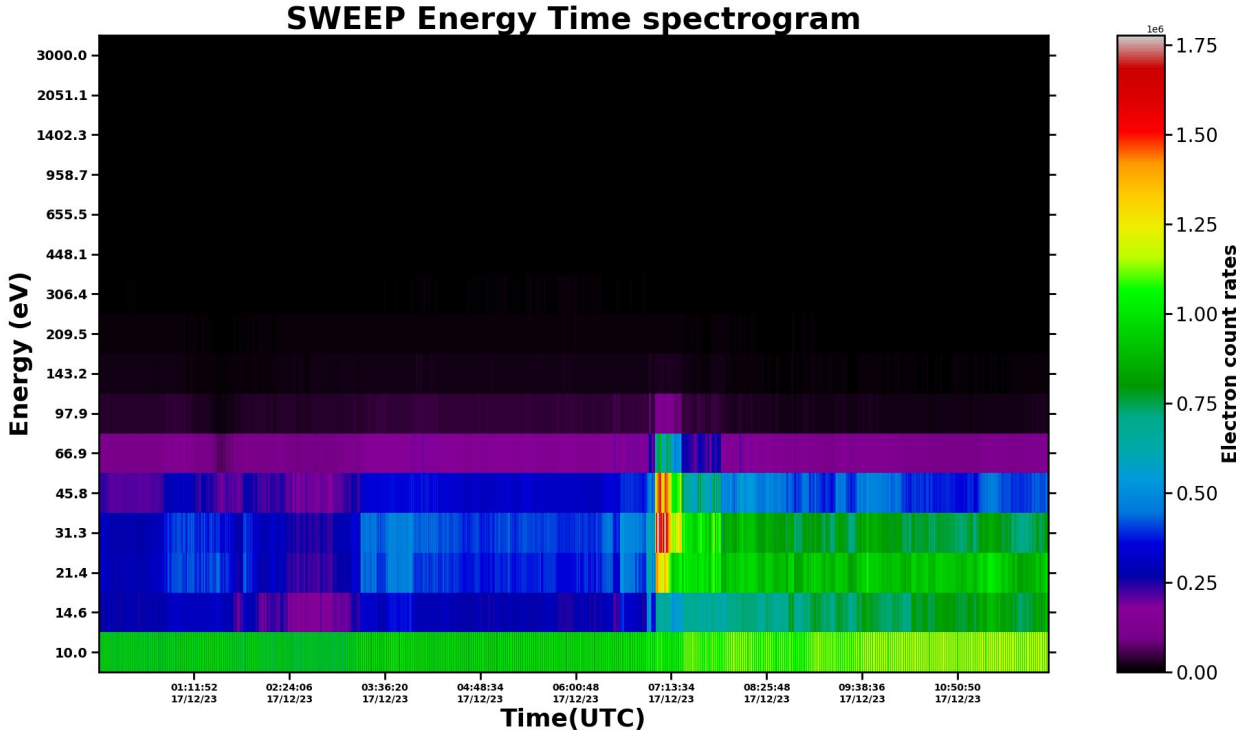

Figure 2: SWEEP electron mode Energy Time spectrogram

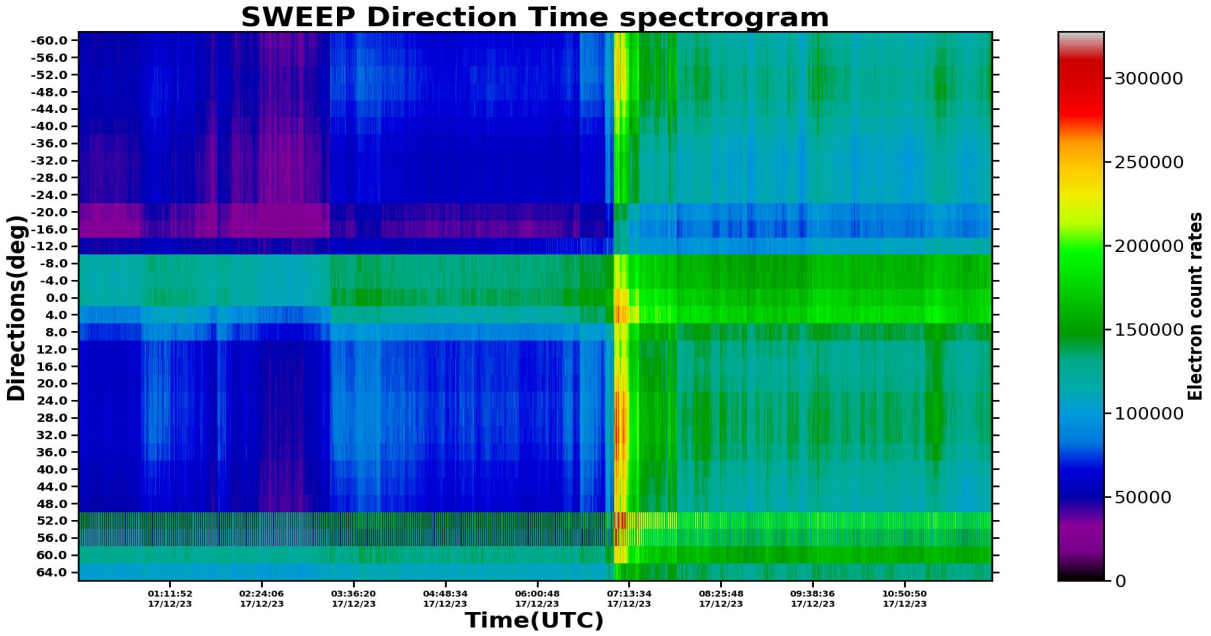

Figure 3: SWEEP electron mode Direction Time spectrogram

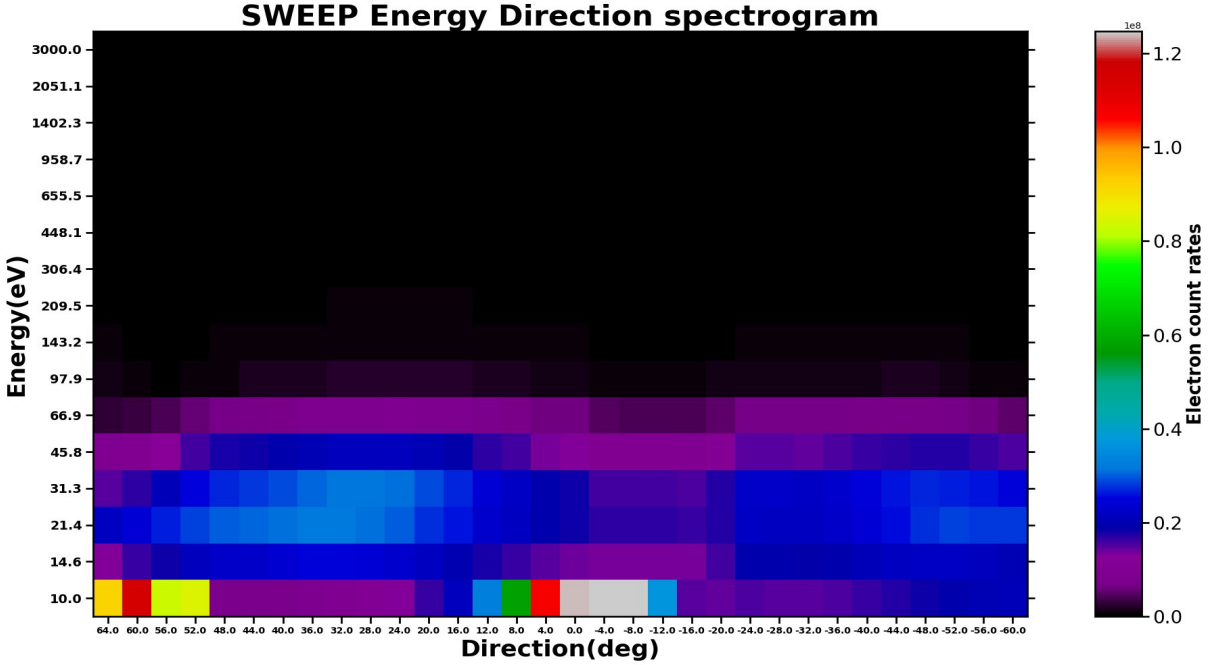

Figure 4: SWEEP electron mode Energy Direction spectrogram

```
# Created by PAPA team on 26-12-2024
import matplotlib.pyplot as plt
from spacepy import pycdf
import numpy as np
from mpl_toolkits.axes_grid1.axes_divider import make_axes_locatable
from matplotlib.ticker import FuncFormatter
import matplotlib.dates as mdates

class  PAPA_data_plots:

   def __init__(self):
      try:
        # include SWEEP electron data path here for SWICAR ion plots and if not required comment the below lines
        self.SWEEP_electron_L1_file_path = 'PPAXXN18P1AL10023209NNNN23352005155669_UNP_9999_999999_V1_1_00/PPA_SWP_ele_E16_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L1_V1_0.cdf'
        self.SWEEP_electron_L1_data = pycdf.CDF(self.SWEEP_electron_L1_file_path)
        # include SWICAR electron data path here for SWICAR ion plots and if not required comment the below lines
        self.SWICAR_electron_L1_file_path ='PPAXXN18P1AL10023209NNNN23352005155669_UNP_9999_999999_V1_1_00/PPA_SWR_ele_E16_23352005155669_UNP_9999_999999_L0_V1_1_20231217000043_L1_V1_0.cdf'
        self.SWICAR_electron_L1_data = pycdf.CDF(self.SWICAR_electron_L1_file_path)
        # include SWICAR ion data path here for SWICAR ion plots and if not required comment the below lines
```

```
        self.SWICAR_ion_L1_file_path =
'PPAXXN18P1AL10023209NNNN23352005155669_UNP_9999_999999_V1_1_00/PPA_SWR
_ion_E32_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L1_V1_0.cdf'
        self.SWICAR_ion_L1_data = pycdf.CDF(self.SWICAR_ion_L1_file_path)
        self.direction_bins = np.linspace(64, -60, 32)
    except Exception as e:
        print(f" Error in the input file paths or input data loading:{str(e)}")
  def SWEEP_data_plots_generation(self):
    # Time series of Total counts plot
    # storing the data to new variables
    self.SWEEP_time_data = self.SWEEP_electron_L1_data['Epoch']
    self.SWEEP_energies_data = np.array(self.SWEEP_electron_L1_data['Energies'])
    self.SWEEP_count_rates = np.array(self.SWEEP_electron_L1_data['Count_rates'])
    # np.sum command does summing over a dimension, here axis=0 is time dimension,
axis=1 is energy dimension, axis=2 is direction dimention
    self.SWEEP_total_count_rates = np.sum(np.sum(self.SWEEP_count_rates, axis=2),
axis=1)  # finding total count rates by summing over all directions and all energies
    self.SWEEP_Energy_Time_count_rates = np.sum(self.SWEEP_count_rates, axis=2)  #
Energy vs time count rates summed over all directions
    self.SWEEP_Direction_Time_count_rates = np.sum(self.SWEEP_count_rates, axis=1)  #
Direction vs time count rates summed over all energies
    self.SWEEP_Energy_Direction_count_rates = np.sum(self.SWEEP_count_rates, axis=0)
# Energy vs Direction count rates summed over the time duration
    # Total count rates time series
    self.total_counts_timeseries_plotting(time_series=self.SWEEP_time_data, title='Times
series of Total count rates',
                                          count_rates=self.SWEEP_total_count_rates , marker_colours='r',
```

```
                yaxis_label="Electron count rates($s^{-1}$)", legend_text='SWEEP')
        plt.savefig('SWEEP_total_count_rates_time_series.png')
        # Energy time spectrogram
        self.Energy_Time_series_spectra(time_data=self.SWEEP_time_data,
matrix=self.SWEEP_Energy_Time_count_rates,
                title="SWEEP Energy Time spectrogram", color_bar_label="Electron
count rates", energy_values=self.SWEEP_energies_data)
        plt.savefig('SWEEP_Energy_Time_series_spectra.png')
        # Direction time spectrogram
        self.Direction_Time_series_spectra(time_data=self.SWEEP_time_data,
matrix=self.SWEEP_Direction_Time_count_rates,
                title="SWEEP Direction Time spectrogram", color_bar_label="Electron
count rates", direction_values=self.direction_bins)
        plt.savefig('SWEEP_Direction_Time_series_spectra.png')
        # Energy direction spectrogram
        self.Energy_Direction_spectra(matrix=self.SWEEP_Energy_Direction_count_rates,
title="SWEEP Energy Direction spectrogram",
                color_bar_label="Electron count
rates",energy_values=self.SWEEP_energies_data, direction_values=self.direction_bins)
        plt.savefig('SWEEP_Energy_Direction_spectra.png')
    def SWICAR_electron_data_plots_generation(self):
        # Time series of Total counts plot
        # storing the data to new variables
        self.SWICAR_electron_time_data = self.SWICAR_electron_L1_data['Epoch']
        self.SWICAR_electron_energies_data =
np.array(self.SWICAR_electron_L1_data['Energies'])
        self.SWICAR_electron_count_rates =
```

```
np.array(self.SWICAR_electron_L1_data['Count_rates'])

    # np.sum command does summing over a dimension, here axis=0 is time dimension, axis=1 is energy dimension, axis=2 is direction dimention

    self.SWICAR_electron_total_count_rates = np.sum(np.sum(self.SWICAR_electron_count_rates, axis=2), axis=1)  # finding total count rates by summing over all directions and all energies

    self.SWICAR_electron_Energy_Time_count_rates = np.sum(self.SWICAR_electron_count_rates, axis=2)  # Energy vs time count rates summed over all directions

    self.SWICAR_electron_Direction_Time_count_rates = np.sum(self.SWICAR_electron_count_rates, axis=1)  # Direction vs time count rates summed over all energies

    self.SWICAR_electron_Energy_Direction_count_rates = np.sum(self.SWICAR_electron_count_rates, axis=0)  # Energy vs Direction count rates summed over the time duration

    # Total count rates time series

    self.total_counts_timeseries_plotting(time_series=self.SWICAR_electron_time_data, title='Times series of Total count rates',
                count_rates=self.SWICAR_electron_total_count_rates , marker_colours='r',
                    yaxis_label="Electron count rates($s^{-1}$)", legend_text='SWICAR Electron')

    plt.savefig('SWICAR_electron_total_count_rates_time_series.png')

    # Energy time spectrogram

    self.Energy_Time_series_spectra(time_data=self.SWICAR_electron_time_data, matrix=self.SWICAR_electron_Energy_Time_count_rates,
                title="SWICAR Electron Energy Time spectrogram", color_bar_label="Electron count rates", energy_values=self.SWICAR_electron_energies_data)
```

```
        plt.savefig('SWICAR_electron_Energy_Time_series_spectra.png')
        # Direction time spectrogram
        self.Direction_Time_series_spectra(time_data=self.SWICAR_electron_time_data,
matrix=self.SWICAR_electron_Direction_Time_count_rates,
                                           title="SWICAR Electron Direction Time spectrogram",
color_bar_label="Electron count rates", direction_values=self.direction_bins)
        plt.savefig('SWICAR_electron_Direction_Time_series_spectra.png')
        # Energy direction spectrogram

self.Energy_Direction_spectra(matrix=self.SWICAR_electron_Energy_Direction_count_rates,
title="SWICAR Electron Energy Direction spectrogram",
                                      color_bar_label="Electron count
rates",energy_values=self.SWICAR_electron_energies_data,
direction_values=self.direction_bins)
        plt.savefig('SWICAR_electron_Energy_Direction_spectra.png')
    def SWICAR_ion_data_plots_generation(self):
        # Time series of Total counts plot
        # storing the data to new variables
        self.SWICAR_ion_time_data = self.SWICAR_ion_L1_data['Epoch']
        self.SWICAR_ion_energies_data = np.array(self.SWICAR_ion_L1_data['Energies'])
        self.SWICAR_ion_count_rates = np.array(self.SWICAR_ion_L1_data['Count_rates'])
        self.SWICAR_ion_TOF_binned_count_rates =
np.array(self.SWICAR_ion_L1_data['TOF_binned_count_rates'])
        # np.sum command does summing over a dimension, here axis=0 is time dimension,
axis=1 is energy dimension, axis=2 is direction dimention
        self.SWICAR_ion_total_count_rates = np.sum(np.sum(self.SWICAR_ion_count_rates,
axis=2), axis=1)  # finding total count rates by summing over all directions and all energies
```

```
        self.SWICAR_ion_Energy_Time_count_rates = np.sum(self.SWICAR_ion_count_rates, axis=2)  # Energy vs time count rates summed over all directions
        self.SWICAR_ion_Direction_Time_count_rates = np.sum(self.SWICAR_ion_count_rates, axis=1)  # Direction vs time count rates summed over all energies
        self.SWICAR_ion_Energy_Direction_count_rates = np.sum(self.SWICAR_ion_count_rates, axis=0)  # Energy vs Direction count rates summed over the time duration
        self.SWICAR_ion_total_TOF_bin_count_rates = np.sum(np.sum(np.sum(self.SWICAR_ion_TOF_binned_count_rates, axis=2), axis=1), axis=0)
        # Total count rates time series
        self.total_counts_timeseries_plotting(time_series=self.SWICAR_ion_time_data, title='Times series of Total count rates',
                                              count_rates=self.SWICAR_ion_total_count_rates , marker_colours='r',
                                              yaxis_label="Ion count rates($s^{-1}$)", legend_text='SWICAR ion')
        plt.savefig('SWICAR_ion_total_count_rates_time_series.png')
        # Energy time spectrogram
        self.Energy_Time_series_spectra(time_data=self.SWICAR_ion_time_data, matrix=self.SWICAR_ion_Energy_Time_count_rates,
                                        title="SWICAR ion Energy Time spectrogram", color_bar_label="ion count rates", energy_values=self.SWICAR_ion_energies_data)
        plt.savefig('SWICAR_ion_Energy_Time_series_spectra.png')
        # Direction time spectrogram
        self.Direction_Time_series_spectra(time_data=self.SWICAR_ion_time_data, matrix=self.SWICAR_ion_Direction_Time_count_rates,
                                           title="SWICAR ion Direction Time spectrogram", color_bar_label="ion count rates", direction_values=self.direction_bins)
```

```
        plt.savefig('SWICAR_ion_Direction_Time_series_spectra.png')
        # Energy direction spectrogram
        self.Energy_Direction_spectra(matrix=self.SWICAR_ion_Energy_Direction_count_rates,
title="SWICAR ion Energy Direction spectrogram",
                                      color_bar_label="ion                                    count
rates",energy_values=self.SWICAR_ion_energies_data, direction_values=self.direction_bins)
        plt.savefig('SWICAR_ion_Energy_Direction_spectra.png')
     # TOF spectra
        self.TOF_bin_count_rates_spectra(self.SWICAR_ion_total_TOF_bin_count_rates)
        plt.savefig('SWICAR_ion_TOF_bin_count_rates_spectra.png')

    def dynamic_scientific_formater(self, value, pos): # This function is used to format the y-
axis text for total count time series plots
        if value == 0:
            return "0"
        power = int(np.floor(np.log10(abs(value))))
        coeff = value / (10 ** power)

        # return f"${coeff:.1f} x 10^{power}$"
        return r"$\mathbf{{{:.1f} \times 10^{{{}}}}}$".format(coeff, power)

    # function for time series plotting
    def total_counts_timeseries_plotting(self, time_series, title, count_rates, marker_colours,
yaxis_label, legend_text):
        plt.rcParams["axes.linewidth"] = 2.0
        fig = plt.figure(figsize=(20, 12))
        ax = plt.gca()
```

```
    plt.title(title, fontsize=24, weight='bold')
    plt.tick_params('both', length=16, width=3, which='major', direction='in')
    plt.tick_params('both', length=8, width=2, which='minor', direction='in', right=True, top=True)
    plt.tick_params(left=True, bottom=True, right=True, top=True)
    plt.ylabel(yaxis_label, fontsize=28, weight='bold')
    plt.xlabel('Time(UTC)', fontsize=28, weight='bold')
    plt.tick_params(left=True, bottom=True)
    plt.scatter(time_series, count_rates, s=50, marker='*', color=marker_colours, label=legend_text)
    plt.yticks(fontsize=18, weight='bold')
    plt.xticks(fontsize=18, weight='bold')
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%H:%M:%S" + '\n' + "%d/%m/%y"))
    plt.legend(loc='upper right', bbox_to_anchor=(1, 1), fontsize=10, borderpad=.05, labelspacing=.8,
               prop=dict(weight='bold', size=26))
    # plt.locator_params(axis="x", nbins=5)
    formatter = FuncFormatter(self.dynamic_scientific_formater)
    # ax.yaxis.set_major_locator(plt.MaxNLocator(iformatternteger=True))
    ax.yaxis.set_major_formatter(formatter)
    plt.xticks(rotation=0)
    # plt.yscale('log')
    plt.minorticks_on()
    plt.grid(True)
    # plt.show()
```

```
  def Direction_Time_series_spectra(self, time_data, matrix, title, color_bar_label, direction_values):
      fig, ax = plt.subplots(figsize=(20, 12))
      plt.tick_params('both', length=8, width=2, which='major', direction='out')
      plt.tick_params('both', length=6, width=4, which='minor', direction='out', right=True)
      plt.yticks(fontsize=14, weight='bold')
      plt.xticks(fontsize=10, weight='bold')
      plt.tick_params(left=True, bottom=True, right=True)
      plt.title(title, fontsize=32, weight='bold')
      plt.ylabel('Directions(deg)', fontsize=26, weight='bold')
      plt.xlabel('Time(UTC)', fontsize=26, weight='bold')
      centers = [1, len(matrix), 0, 31]
      dx, = np.diff(centers[:2]) / (matrix.shape[0] - 1)
      dy, = -np.diff(centers[2:]) / (matrix.shape[1] - 1)
      extent = [centers[0] - dx / 2, centers[1] + dx / 2, centers[2] + dy / 2, centers[3] - dy / 2]
      plt.xticks(np.arange(centers[0], centers[1] + dx, dx))
      plt.yticks(np.arange(centers[3], centers[2] + dy, dy))
      plt.yticks(ticks=np.arange(len(direction_values)), labels=np.round(direction_values, 0))
      plt.xticks(np.arange(len(time_data)), [time.strftime("%H:%M:%S" + '\n' + "%d/%m/%y") for time in time_data])
      plt.xticks(rotation=0)
      plt.locator_params(axis='x', nbins=10)
      img = ax.imshow(np.transpose(matrix), origin='lower', cmap='nipy_spectral', extent=extent, vmin=0,
                      vmax=np.max(matrix) + 1,
```

```
                aspect='auto', interpolation='nearest')  #  , interpolation='hanning', np.max(matrix)
        cbar_ax = make_axes_locatable(ax).append_axes(position='right', size='5%', pad=1)
        # Create colorbar
        cbar = fig.colorbar(mappable=img, cax=cbar_ax, label=color_bar_label)
        cbar.ax.locator_params(nbins=8)
        cbar.set_label(label=color_bar_label, weight='bold', fontsize=20)
        cbar.ax.tick_params(labelsize=20, width=2, length=8)
    # plt.show()

    def Energy_Time_series_spectra(self, time_data, matrix, title, color_bar_label, energy_values):
        fig, ax = plt.subplots(figsize=(20, 12))
        plt.tick_params('both', length=8, width=2, which='major', direction='out')
    plt.tick_params('both', length=6, width=4, which='minor', direction='out', right=True)
        plt.yticks(fontsize=14, weight='bold')
        plt.xticks(fontsize=10, weight='bold')
        plt.tick_params(left=True, bottom=True, right=True)
        plt.title(title, fontsize=32, weight='bold')
        plt.ylabel('Energy (eV)', fontsize=26, weight='bold')
        plt.xlabel('Time(UTC)', fontsize=26, weight='bold')
        centers = [1, len(matrix), 0, len(matrix[0])-1]
        dx, = np.diff(centers[:2]) / (matrix.shape[0] - 1)
        dy, = -np.diff(centers[2:]) / (matrix.shape[1] - 1)
        extent = [centers[0] - dx / 2, centers[1] + dx / 2, centers[2] + dy / 2, centers[3] - dy / 2]
        plt.xticks(np.arange(centers[0], centers[1] + dx, dx))
```

```
    plt.yticks(np.arange(centers[3], centers[2] + dy, dy))
    plt.yticks( ticks=np.arange(len(energy_values)), labels=np.round(energy_values,2))
    plt.xticks(np.arange(len(time_data)), [time.strftime("%H:%M:%S" + '\n' + "%d/%m/%y") for time in time_data])
    plt.xticks(rotation=0)
    plt.locator_params(axis='x', nbins=10)
    img = ax.imshow(np.transpose(matrix), origin='lower', cmap='nipy_spectral', extent=extent, vmin=0,
                    vmax=np.max(matrix) + 1,
                    aspect='auto', interpolation='nearest')
    # Create axis for colorbar
    cbar_ax = make_axes_locatable(ax).append_axes(position='right', size='5%', pad=1)

    # Create colorbar
    cbar = fig.colorbar(mappable=img, cax=cbar_ax, label=color_bar_label)
    cbar.ax.locator_params(nbins=8)
    cbar.set_label(label=color_bar_label, weight='bold', fontsize=20)
    cbar.ax.tick_params(labelsize=20, width=2, length=8)
    # plt.show()

  def Energy_Direction_spectra(self, matrix, title, color_bar_label, energy_values, direction_values):
    fig, ax = plt.subplots(figsize=(20, 12))
    plt.tick_params('both', length=8, width=2, which='major', direction='out')
    plt.tick_params('both', length=6, width=4, which='minor', direction='out', right=True)
    plt.yticks(fontsize=14, weight='bold')
```

```
    plt.xticks(fontsize=10, weight='bold')
    plt.tick_params(left=True, bottom=True, right=True)
    plt.title(title, fontsize=32, weight='bold')
    plt.ylabel('Energy(eV)', fontsize=26, weight='bold')
    plt.xlabel('Direction(deg)', fontsize=26, weight='bold')
    centers = [0, len(matrix[0])-1, 0, len(matrix)-1]
    dx, = np.diff(centers[:2]) / (matrix.shape[1] - 1)
    dy, = -np.diff(centers[2:]) / (matrix.shape[0] - 1)
    extent = [centers[0] - dx / 2, centers[1] + dx / 2, centers[2] + dy / 2, centers[3] - dy / 2]
    plt.xticks(np.arange(centers[0], centers[1] + dx, dx))
    plt.yticks(np.arange(centers[3], centers[2] + dy, dy))
    plt.yticks(ticks=np.arange(len(energy_values)), labels=np.round(energy_values, 2))
    plt.xticks(ticks=np.arange(len(direction_values)), labels=np.round(direction_values, 0))
    img = ax.imshow(matrix, origin='lower', cmap='nipy_spectral', extent=extent, vmin=0,
vmax=(np.max(matrix)) + 1,
                    aspect='auto', interpolation='nearest')  #  , interpolation='hanning',
np.max(matrix)
    # Create axis for colorbar
    cbar_ax = make_axes_locatable(ax).append_axes(position='right', size='5%', pad=1)
    # Create colorbar
    cbar = fig.colorbar(mappable=img, cax=cbar_ax, label=color_bar_label)
    # cb = plt.colorbar()
    cbar.set_label(label=color_bar_label, weight='bold', fontsize=20)
    cbar.ax.tick_params(labelsize=20, width=2, length=8)
    cbar.ax.locator_params(nbins=8)
    # plt.show()
```

```
  def TOF_bin_count_rates_spectra(self, TOF_counts):


      plt.rcParams["axes.linewidth"] = 2.0
      tof_bins = np.linspace(1, 32, 32)
      fig = plt.figure(figsize=(20, 12))
      # fig=plt.figure(figsize=(15, 10))
      # ax=fig.gca()
      plt.title('TOF binned Count rates spectra', fontsize=28, weight='bold')
      plt.tick_params('y', length=16, width=3, which='major', direction='in')
      plt.tick_params('y', length=8, width=2, which='minor', direction='in', right=True)
      plt.tick_params('x', length=16, width=3, which='major', direction='out')
      plt.tick_params(left=True, bottom=True, right=True)
      plt.ylabel('TOF binned Count rates($s^{-1}$)', fontsize=24, weight='bold')
      plt.xlabel('TOF bins', fontsize=24, weight='bold')
      plt.yticks(fontsize=18, weight='bold')
plt.xticks(fontsize=18, weight='bold')
      plt.xticks(rotation=0)
      plt.xlim(0, 33)
      plt.scatter(tof_bins, TOF_counts, s=300, marker='*', color='r')
      plt.yticks(fontsize=28, weight='bold')
      plt.xticks(fontsize=14, weight='bold')
      plt.locator_params(axis="x", nbins=64)
      plt.yscale('log')
      plt.grid(True)
      # plt.legend()
```

```
    # plt.show()


if __name__ == '__main__':
    all_plots = PAPA_data_plots()
    all_plots.SWEEP_data_plots_generation() # comment this line if SWEEP electron plots are not required
    all_plots.SWICAR_electron_data_plots_generation() # comment this line if SWICAR electron plots are not required
    all_plots.SWICAR_ion_data_plots_generation() # comment this line if SWICAR ion plots are not required
```

ANNEXURE-2

Python code to read the CDF files (level-2) and generate plots

The code provided in Annexure-1 was configured to process Level-2 CDF data files and provided herewith. The same code is available in PRADAN under 'other downloads'. Sample plots for SWICAR ion mode, generated using level-2 CDF files are shown in Figs.1 to 5.

SWICAR ion file name :

PPA_SWR_ion_E32_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L2_V1_0.cdf

```
# Created by PAPA team on 26-12-2024
import matplotlib.pyplot as plt
from spacepy import pycdf
import numpy as np
from mpl_toolkits.axes_grid1.axes_divider import make_axes_locatable
from matplotlib.ticker import FuncFormatter
import matplotlib.dates as mdates
class  PAPA_data_plots:


   def __init__(self):
      try:
        # include SWEEP electron data path here for SWICAR ion plots and if not required comment the below lines
        self.SWEEP_electron_L2_file_path = 'PPA_SWP_ele_E16_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L2_V1_0.cdf'
        self.SWEEP_electron_L2_data = pycdf.CDF(self.SWEEP_electron_L2_file_path)
        print(self.SWEEP_electron_L2_data)
```

```
        # include SWICAR electron data path here for SWICAR ion plots and if not required comment the below lines
        self.SWICAR_electron_L2_file_path = 'PPA_SWR_ele_E16_23352005155669_UNP_9999_999999_L0_V1_1_20231217000043_L2_V1_0.cdf'
        self.SWICAR_electron_L2_data = pycdf.CDF(self.SWICAR_electron_L2_file_path)
        print(self.SWEEP_electron_L2_data)
        # include SWICAR ion data path here for SWICAR ion plots and if not required comment the below lines
        self.SWICAR_ion_L2_file_path = 'PPA_SWR_ion_E32_23352005155669_UNP_9999_999999_L0_V1_1_20231216235938_L2_V1_0.cdf'
        self.SWICAR_ion_L2_data = pycdf.CDF(self.SWICAR_ion_L2_file_path)
        print(self.SWICAR_ion_L2_data)
        self.direction_bins = np.linspace(64, -60, 32)
      except Exception as e:
        print(f" Error in the input file paths or input data loading:{str(e)}")
  def SWEEP_data_plots_generation(self):
      # Time series of TotalDifferential number flux
      # storing the data to new variables
      self.SWEEP_time_data = self.SWEEP_electron_L2_data['Epoch']
      self.SWEEP_energies_data = np.array(self.SWEEP_electron_L2_data['Energies'])
      self.SWEEP_Differential_number_flux = np.array(self.SWEEP_electron_L2_data['Differential_number_flux'])
      # np.sum command does summing over a dimension, here axis=0 is time dimension, axis=1 is energy dimension, axis=2 is direction dimention
      self.SWEEP_total_Differential_number_flux =
```

```
np.sum(np.sum(self.SWEEP_Differential_number_flux, axis=2), axis=1)  # finding total count rates by summing over all directions and all energies
    self.SWEEP_Energy_Time_Differential_number_flux = np.sum(self.SWEEP_Differential_number_flux, axis=2)  # Energy vs time count rates summed over all directions
    self.SWEEP_Direction_Time_Differential_number_flux = np.sum(self.SWEEP_Differential_number_flux, axis=1)  # Direction vs time count rates summed over all energies
    self.SWEEP_Energy_Direction_Differential_number_flux = np.sum(self.SWEEP_Differential_number_flux, axis=0)  # Energy vs Direction count rates summed over the time duration
    # Total count rates time series
    self.total_counts_timeseries_plotting(time_series=self.SWEEP_time_data, title='Times series of Total Differential number flux',
                                          count_rates=self.SWEEP_total_Differential_number_flux , marker_colours='r',
                                          yaxis_label="Differential number flux($cm^{-2} sr^{-1} eV^{-1} s^{-1}$)", legend_text='SWEEP')
    plt.savefig('SWEEP_total_Differential_number_flux_time_series.png')
    # Energy time spectrogram
    self.Energy_Time_series_spectra(time_data=self.SWEEP_time_data, matrix=self.SWEEP_Energy_Time_Differential_number_flux,
                                    title="SWEEP Energy Time spectrogram", color_bar_label="Electron Differential number flux", energy_values=self.SWEEP_energies_data)
    plt.savefig('SWEEP_Energy_Time_series_spectra.png')
    # Direction time spectrogram
    self.Direction_Time_series_spectra(time_data=self.SWEEP_time_data,
```

```
matrix=self.SWEEP_Direction_Time_Differential_number_flux,
                        title="SWEEP Direction Time spectrogram", color_bar_label="Electron Differential number flux", direction_values=self.direction_bins)
        plt.savefig('SWEEP_Direction_Time_series_spectra.png')
        # Energy direction spectrogram

self.Energy_Direction_spectra(matrix=self.SWEEP_Energy_Direction_Differential_number_flux, title="SWEEP Energy Direction spectrogram",
                        color_bar_label="Electron Differential number flux",energy_values=self.SWEEP_energies_data, direction_values=self.direction_bins)
        plt.savefig('SWEEP_Energy_Direction_spectra.png')
    def SWICAR_electron_data_plots_generation(self):
        # Time series of Total counts plot
        # storing the data to new variables
        self.SWICAR_electron_time_data = self.SWICAR_electron_L2_data['Epoch']
        self.SWICAR_electron_energies_data = np.array(self.SWICAR_electron_L2_data['Energies'])
        self.SWICAR_electron_Differential_number_flux = np.array(self.SWICAR_electron_L2_data['Differential_number_flux'])
        # np.sum command does summing over a dimension, here axis=0 is time dimension, axis=1 is energy dimension, axis=2 is direction dimention
        self.SWICAR_electron_total_Differential_number_flux = np.sum(np.sum(self.SWICAR_electron_Differential_number_flux, axis=2),
                                axis=1)  # finding total count rates by summing over all directions and all energies
        self.SWICAR_electron_Energy_Time_Differential_number_flux = np.sum(self.SWICAR_electron_Differential_number_flux,
```

```
                                    axis=2)  # Energy vs time count rates summed over all
directions
        self.SWICAR_electron_Direction_Time_Differential_number_flux                =
np.sum(self.SWICAR_electron_Differential_number_flux,
                                                axis=1)  # Direction vs time count rates summed over
all energies
        self.SWICAR_electron_Energy_Direction_Differential_number_flux              =
np.sum(self.SWICAR_electron_Differential_number_flux,
                                                 axis=0)  # Energy vs Direction count rates summed
over the time duration
        # Total count rates time series
        self.total_counts_timeseries_plotting(time_series=self.SWICAR_electron_time_data,
                                              title='Times series of Total Differential number flux',

count_rates=self.SWICAR_electron_total_Differential_number_flux, marker_colours='r',
                                              yaxis_label="Differential number flux($cm^{-2} sr^{-1} eV^{-1} s^{-
1}$)",
                                              legend_text='SWICAR electron')
        plt.savefig('SWICAR_electron_total_Differential_number_flux_time_series.png')
        # Energy time spectrogram
        self.Energy_Time_series_spectra(time_data=self.SWICAR_electron_time_data,

matrix=self.SWICAR_electron_Energy_Time_Differential_number_flux,
                                        title="SWICAR electron Energy Time spectrogram",
                                        color_bar_label="Electron Differential number flux",
                                        energy_values=self.SWICAR_electron_energies_data)
        plt.savefig('SWICAR_electron_Energy_Time_series_spectra.png')
```

```
        # Direction time spectrogram
        self.Direction_Time_series_spectra(time_data=self.SWICAR_electron_time_data,

matrix=self.SWICAR_electron_Direction_Time_Differential_number_flux,
                                           title="SWICAR electron Direction Time spectrogram",
                                           color_bar_label="Electron Differential number flux",
                                           direction_values=self.direction_bins)
        plt.savefig('SWICAR_electron_Direction_Time_series_spectra.png')
        # Energy direction spectrogram

self.Energy_Direction_spectra(matrix=self.SWICAR_electron_Energy_Direction_Differential_
number_flux,
                                      title="SWICAR electron Energy Direction spectrogram",
                                      color_bar_label="Electron Differential number flux",
                                      energy_values=self.SWICAR_electron_energies_data,
direction_values=self.direction_bins)
        plt.savefig('SWICAR_electron_Energy_Direction_spectra.png')
    def SWICAR_ion_data_plots_generation(self):
        # Time series of Total counts plot
        # storing the data to new variables
        self.SWICAR_ion_time_data = self.SWICAR_ion_L2_data['Epoch']
        self.SWICAR_ion_energies_data = np.array(self.SWICAR_ion_L2_data['Energies'])
        self.SWICAR_ion_Differential_number_flux                                    =
np.array(self.SWICAR_ion_L2_data['Differential_number_flux'])
        self.SWICAR_ion_TOF_binned_Differential_number_flux                         =
np.array(self.SWICAR_ion_L2_data['TOF_binned_flux'])
        # np.sum command does summing over a dimension, here axis=0 is time dimension,
```

```
axis=1 is energy dimension, axis=2 is direction dimention
        self.SWICAR_ion_total_Differential_number_flux = np.sum(
            np.sum(self.SWICAR_ion_Differential_number_flux, axis=2),
            axis=1)  # finding total count rates by summing over all directions and all energies
        self.SWICAR_ion_Energy_Time_Differential_number_flux = np.sum(
            self.SWICAR_ion_Differential_number_flux,
            axis=2)  # Energy vs time count rates summed over all directions
        self.SWICAR_ion_Direction_Time_Differential_number_flux = np.sum(
            self.SWICAR_ion_Differential_number_flux,
            axis=1)  # Direction vs time count rates summed over all energies
        self.SWICAR_ion_Energy_Direction_Differential_number_flux = np.sum(
            self.SWICAR_ion_Differential_number_flux,
            axis=0)  # Energy vs Direction count rates summed over the time duration
        self.SWICAR_ion_total_TOF_bin_Differential_number_flux = np.sum(
            np.sum(np.sum(self.SWICAR_ion_TOF_binned_Differential_number_flux,       axis=2),
axis=1), axis=0)
        # Total count rates time series
        self.total_counts_timeseries_plotting(time_series=self.SWICAR_ion_time_data,
                                              title='Times series of Total Differential number flux',
                                              count_rates=self.SWICAR_ion_total_Differential_number_flux,
                                              marker_colours='r',
                                              yaxis_label="Differential number flux($cm^{-2} sr^{-1} eV^{-1} s^{-
1}$)",
                                              legend_text='SWICAR ion')
        plt.savefig('SWICAR_ion_total_Differential_number_flux_time_series.png')
        # Energy time spectrogram
```

```
        self.Energy_Time_series_spectra(time_data=self.SWICAR_ion_time_data,
                                        matrix=self.SWICAR_ion_Energy_Time_Differential_number_flux,
                                        title="SWICAR ion Energy Time spectrogram",
                                        color_bar_label="Ion Differential number flux",
                                        energy_values=self.SWICAR_ion_energies_data)
        plt.savefig('SWICAR_ion_Energy_Time_series_spectra.png')
        # Direction time spectrogram
        self.Direction_Time_series_spectra(time_data=self.SWICAR_ion_time_data,
                                           matrix=self.SWICAR_ion_Direction_Time_Differential_number_flux,
                                           title="SWICAR ion Direction Time spectrogram",
                                           color_bar_label="Ion Differential number flux",
                                           direction_values=self.direction_bins)
        plt.savefig('SWICAR_ion_Direction_Time_series_spectra.png')
        # Energy direction spectrogram

self.Energy_Direction_spectra(matrix=self.SWICAR_ion_Energy_Direction_Differential_number_flux,
                              title="SWICAR ion Energy Direction spectrogram",
                              color_bar_label="Ion Differential number flux",
                              energy_values=self.SWICAR_ion_energies_data,
                              direction_values=self.direction_bins)
        plt.savefig('SWICAR_ion_Energy_Direction_spectra.png')
        # TOF spectra

self.TOF_bin_count_rates_spectra(self.SWICAR_ion_total_TOF_bin_Differential_number_flux)
```

```
        plt.savefig('SWICAR_ion_TOF_bin_Differential_number_flux_spectra.png')
    def dynamic_scientific_formater(self, value, pos): # This function is used to format the y-axis text for total count time series plots
        if value == 0:
            return "0"
        power = int(np.floor(np.log10(abs(value))))
        coeff = value / (10 ** power)

        # return f"${coeff:.1f} x 10^{power}$"
        return r"$\mathbf{{{:.1f} \times 10^{{{}}}}}$".format(coeff, power)

    # function for time series plotting
    def total_counts_timeseries_plotting(self, time_series, title, count_rates, marker_colours, yaxis_label, legend_text):
        plt.rcParams["axes.linewidth"] = 2.0
        fig = plt.figure(figsize=(20, 12))
        ax = plt.gca()
        plt.title(title, fontsize=24, weight='bold')
        plt.tick_params('both', length=16, width=3, which='major', direction='in')
        plt.tick_params('both', length=8, width=2, which='minor', direction='in', right=True, top=True)
        plt.tick_params(left=True, bottom=True, right=True, top=True)
        plt.ylabel(yaxis_label, fontsize=28, weight='bold')
        plt.xlabel('Time(UTC)', fontsize=28, weight='bold')
        plt.tick_params(left=True, bottom=True)
        plt.scatter(time_series, count_rates, s=50, marker='*', color=marker_colours, label=legend_text)
```

```
    plt.yticks(fontsize=18, weight='bold')
    plt.xticks(fontsize=18, weight='bold')
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%H:%M:%S" + '\n' + "%d/%m/%y"))
    plt.legend(loc='upper right', bbox_to_anchor=(1, 1), fontsize=10, borderpad=.05, labelspacing=.8,
               prop=dict(weight='bold', size=26))
    # plt.locator_params(axis="x", nbins=5)
    formatter = FuncFormatter(self.dynamic_scientific_formater)
    # ax.yaxis.set_major_locator(plt.MaxNLocator(iformatternteger=True))
    ax.yaxis.set_major_formatter(formatter)
    plt.xticks(rotation=0)
    # plt.yscale('log')
    plt.minorticks_on()
    plt.grid(True)
    # plt.show()


  def Direction_Time_series_spectra(self, time_data, matrix, title, color_bar_label, direction_values):
    fig, ax = plt.subplots(figsize=(20, 12))
    plt.tick_params('both', length=8, width=2, which='major', direction='out')
    plt.tick_params('both', length=6, width=4, which='minor', direction='out', right=True)
    plt.yticks(fontsize=14, weight='bold')
    plt.xticks(fontsize=10, weight='bold')
    plt.tick_params(left=True, bottom=True, right=True)
    plt.title(title, fontsize=32, weight='bold')
```

```
    plt.ylabel('Directions(deg)', fontsize=26, weight='bold')
    plt.xlabel('Time(UTC)', fontsize=26, weight='bold')
    centers = [1, len(matrix), 0, 31]
    dx, = np.diff(centers[:2]) / (matrix.shape[0] - 1)
    dy, = -np.diff(centers[2:]) / (matrix.shape[1] - 1)
    extent = [centers[0] - dx / 2, centers[1] + dx / 2, centers[2] + dy / 2, centers[3] - dy / 2]
    plt.xticks(np.arange(centers[0], centers[1] + dx, dx))
    plt.yticks(np.arange(centers[3], centers[2] + dy, dy))
    plt.yticks(ticks=np.arange(len(direction_values)), labels=np.round(direction_values, 0))
    plt.xticks(np.arange(len(time_data)), [time.strftime("%H:%M:%S" + '\n' + "%d/%m/%y")
for time in time_data])
    plt.xticks(rotation=0)
    plt.locator_params(axis='x', nbins=10)
    img = ax.imshow(np.transpose(matrix), origin='lower', cmap='nipy_spectral',
extent=extent, vmin=0,
                    vmax=np.max(matrix) + 1,
                    aspect='auto', interpolation='nearest')  # , interpolation='hanning',
np.max(matrix)
    cbar_ax = make_axes_locatable(ax).append_axes(position='right', size='5%', pad=1)
    # Create colorbar
    cbar = fig.colorbar(mappable=img, cax=cbar_ax, label=color_bar_label)
    cbar.ax.locator_params(nbins=8)
    cbar.set_label(label=color_bar_label, weight='bold', fontsize=20)
    cbar.ax.tick_params(labelsize=20, width=2, length=8)
    # plt.show()
```

```
    def Energy_Time_series_spectra(self, time_data, matrix, title, color_bar_label, energy_values):
        fig, ax = plt.subplots(figsize=(20, 12))
        plt.tick_params('both', length=8, width=2, which='major', direction='out')
        plt.tick_params('both', length=6, width=4, which='minor', direction='out', right=True)
        plt.yticks(fontsize=14, weight='bold')
        plt.xticks(fontsize=10, weight='bold')
        plt.tick_params(left=True, bottom=True, right=True)
        plt.title(title, fontsize=32, weight='bold')
        plt.ylabel('Energy (eV)', fontsize=26, weight='bold')
        plt.xlabel('Time(UTC)', fontsize=26, weight='bold')
        centers = [1, len(matrix), 0, len(matrix[0])-1]
        dx, = np.diff(centers[:2]) / (matrix.shape[0] - 1)
        dy, = -np.diff(centers[2:]) / (matrix.shape[1] - 1)
        extent = [centers[0] - dx / 2, centers[1] + dx / 2, centers[2] + dy / 2, centers[3] - dy / 2]
        plt.xticks(np.arange(centers[0], centers[1] + dx, dx))
        plt.yticks(np.arange(centers[3], centers[2] + dy, dy))
        plt.yticks( ticks=np.arange(len(energy_values)), labels=np.round(energy_values,2))
        plt.xticks(np.arange(len(time_data)), [time.strftime("%H:%M:%S" + '\n' + "%d/%m/%y") for time in time_data])
        plt.xticks(rotation=0)
        plt.locator_params(axis='x', nbins=10)
        img = ax.imshow(np.transpose(matrix), origin='lower', cmap='nipy_spectral', extent=extent, vmin=0,
                        vmax=np.max(matrix) + 1,
                        aspect='auto', interpolation='nearest')
```

```
    # Create axis for colorbar
    cbar_ax = make_axes_locatable(ax).append_axes(position='right', size='5%', pad=1)

    # Create colorbar
    cbar = fig.colorbar(mappable=img, cax=cbar_ax, label=color_bar_label)
    cbar.ax.locator_params(nbins=8)
    cbar.set_label(label=color_bar_label, weight='bold', fontsize=20)
    cbar.ax.tick_params(labelsize=20, width=2, length=8)
    # plt.show()

  def Energy_Direction_spectra(self, matrix, title, color_bar_label, energy_values, direction_values):
    fig, ax = plt.subplots(figsize=(20, 12))
    plt.tick_params('both', length=8, width=2, which='major', direction='out')
    plt.tick_params('both', length=6, width=4, which='minor', direction='out', right=True)
    plt.yticks(fontsize=14, weight='bold')
    plt.xticks(fontsize=10, weight='bold')
    plt.tick_params(left=True, bottom=True, right=True)
    plt.title(title, fontsize=32, weight='bold')
    plt.ylabel('Energy(eV)', fontsize=26, weight='bold')
    plt.xlabel('Direction(deg)', fontsize=26, weight='bold')
    centers = [0, len(matrix[0])-1, 0, len(matrix)-1]
    dx, = np.diff(centers[:2]) / (matrix.shape[1] - 1)
    dy, = -np.diff(centers[2:]) / (matrix.shape[0] - 1)
    extent = [centers[0] - dx / 2, centers[1] + dx / 2, centers[2] + dy / 2, centers[3] - dy / 2]
    plt.xticks(np.arange(centers[0], centers[1] + dx, dx))
```

```
    plt.yticks(np.arange(centers[3], centers[2] + dy, dy))
    plt.yticks(ticks=np.arange(len(energy_values)), labels=np.round(energy_values, 2))
    plt.xticks(ticks=np.arange(len(direction_values)), labels=np.round(direction_values, 0))
    img = ax.imshow(matrix, origin='lower', cmap='nipy_spectral', extent=extent, vmin=0, vmax=(np.max(matrix)) + 1,
                    aspect='auto', interpolation='nearest')  #  , interpolation='hanning', np.max(matrix)
    # Create axis for colorbar
    cbar_ax = make_axes_locatable(ax).append_axes(position='right', size='5%', pad=1)
    # Create colorbar
    cbar = fig.colorbar(mappable=img, cax=cbar_ax, label=color_bar_label)
    # cb = plt.colorbar()
    cbar.set_label(label=color_bar_label, weight='bold', fontsize=20)
    cbar.ax.tick_params(labelsize=20, width=2, length=8)
    cbar.ax.locator_params(nbins=8)
    # plt.show()

  def TOF_bin_count_rates_spectra(self, TOF_counts):

    plt.rcParams["axes.linewidth"] = 2.0
    tof_bins = np.linspace(1, 32, 32)
    fig = plt.figure(figsize=(20, 12))
    # fig=plt.figure(figsize=(15, 10))
    # ax=fig.gca()
    plt.title('TOF binned Differential flux spectra', fontsize=28, weight='bold')
    plt.tick_params('y', length=16, width=3, which='major', direction='in')
```

```
    plt.tick_params('y', length=8, width=2, which='minor', direction='in', right=True)
    plt.tick_params('x', length=16, width=3, which='major', direction='out')
    plt.tick_params(left=True, bottom=True, right=True)
    plt.ylabel('Differential flux($cm^{-2} sr^{-1} eV^{-1} s^{-1}$)', fontsize=24, weight='bold')
    plt.xlabel('TOF bins', fontsize=24, weight='bold')
    plt.yticks(fontsize=18, weight='bold')
    plt.xticks(fontsize=18, weight='bold')
    plt.xticks(rotation=0)
    plt.xlim(0, 33)
    plt.scatter(tof_bins, TOF_counts, s=300, marker='*', color='r')
    plt.yticks(fontsize=28, weight='bold')
    plt.xticks(fontsize=14, weight='bold')
    plt.locator_params(axis='x', nbins=64)
    plt.yscale('log')
    plt.grid(True)
    # plt.legend()
    # plt.show()


if __name__ == '__main__':
  all_plots = PAPA_data_plots()
  all_plots.SWEEP_data_plots_generation() # comment this line if SWEEP electron plots are not required
  all_plots.SWICAR_electron_data_plots_generation() # comment this line if SWICAR electron plots are not required
  all_plots.SWICAR_ion_data_plots_generation() # comment this line if SWICAR ion plots are not required
```

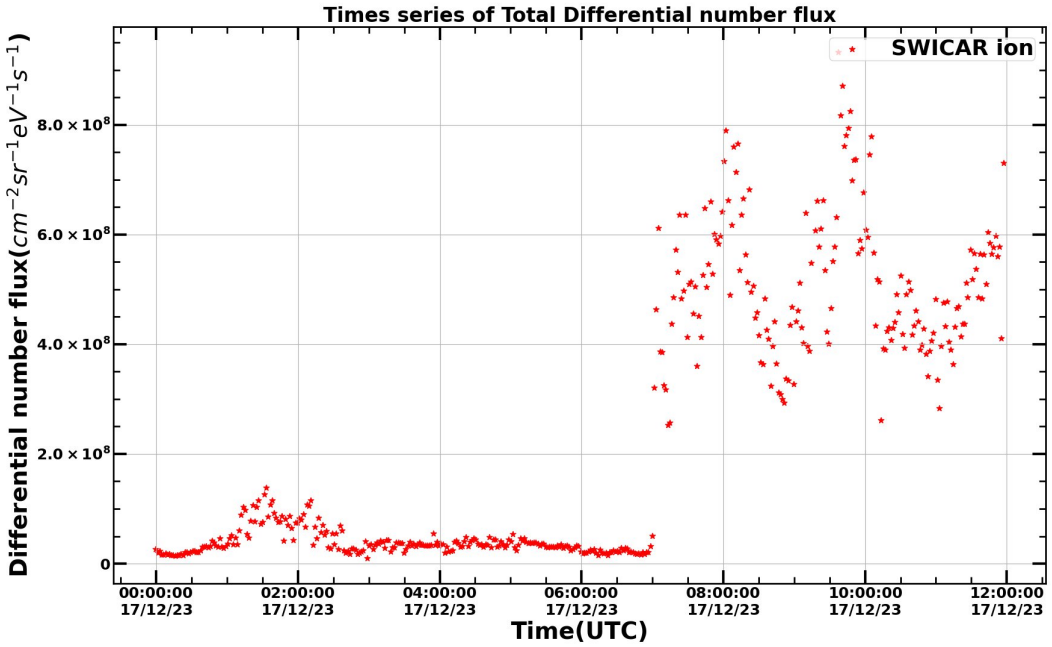

Figure 1: Time series of Total Differential number flux for SWICAR ion mode

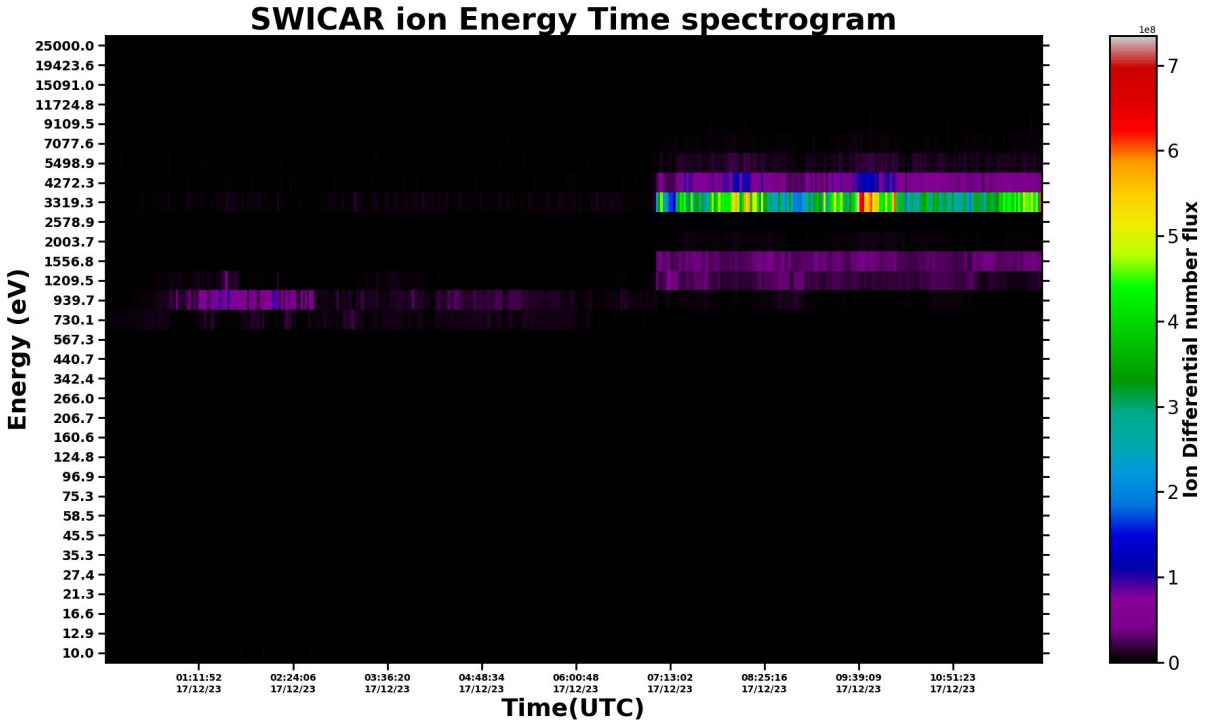

Figure 2: SWICAR ion Energy Time spectrogram

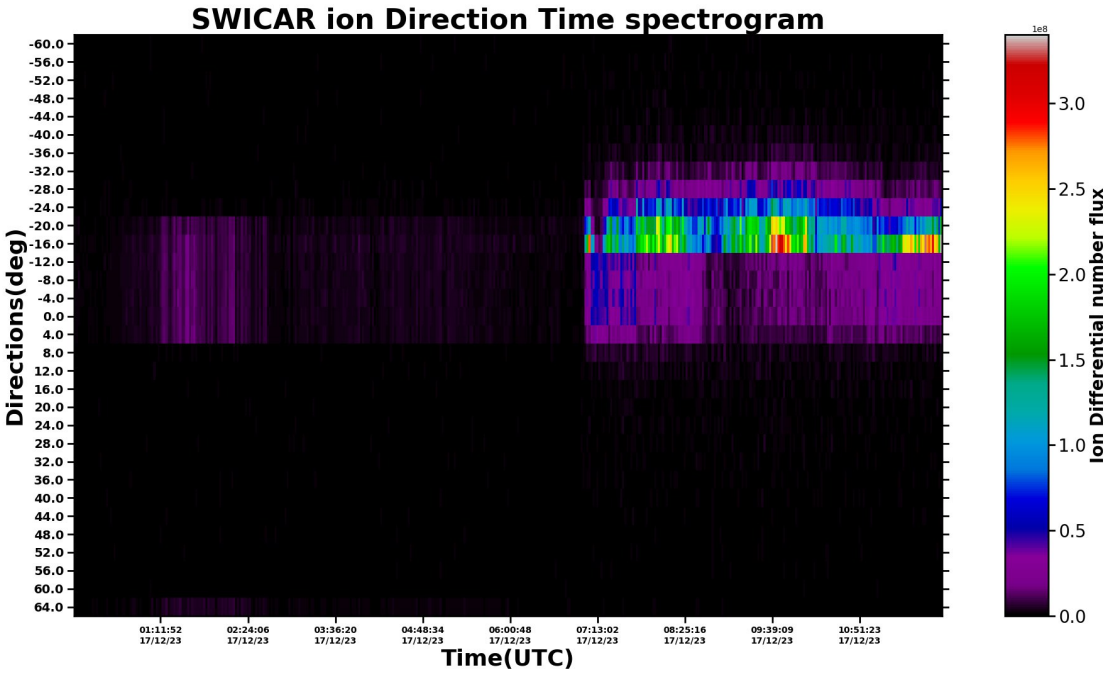

Figure 3: SWICAR ion Direction Time spectrogram

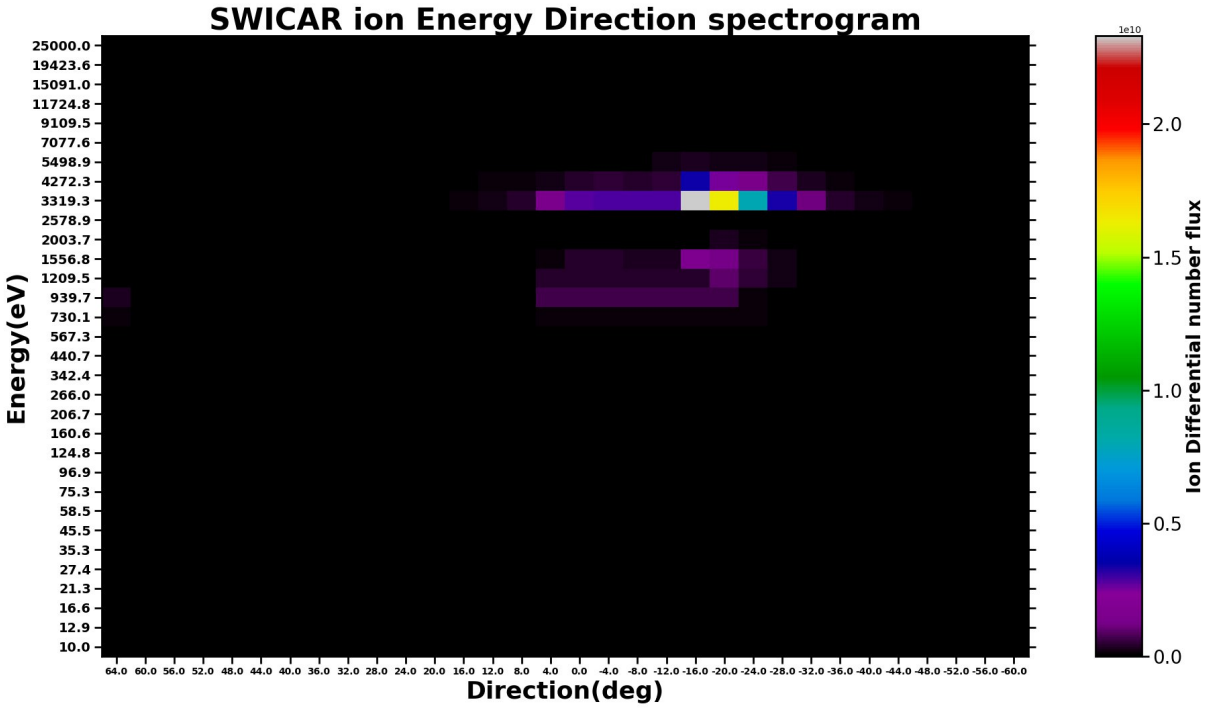

Figure 4: SWICAR ion Energy Direction spectrogram

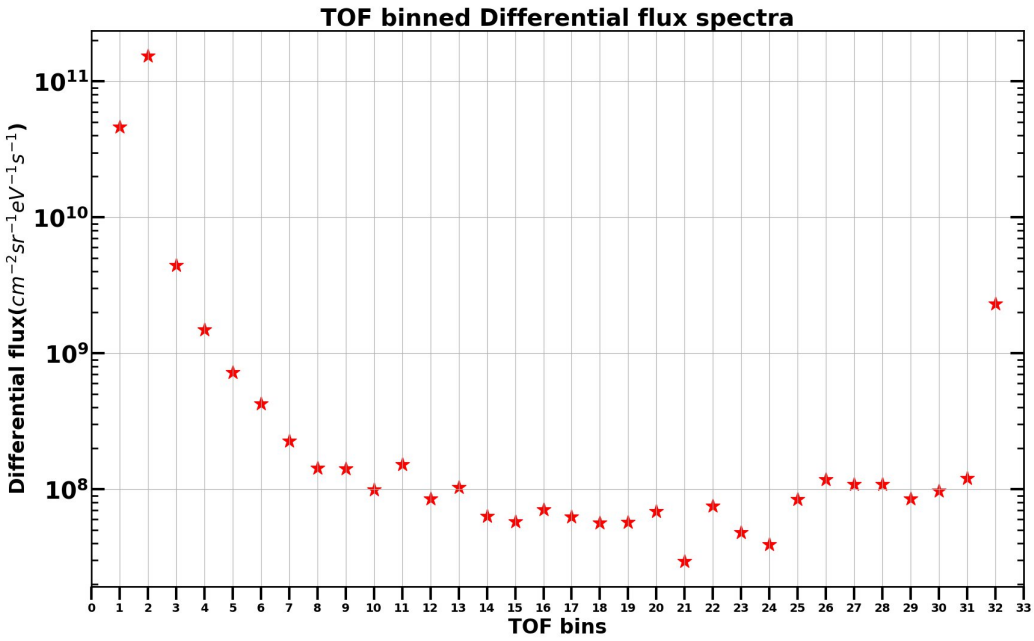

Figure 5: Time series of SWICAR ion TOF bin Differential number flux spectra