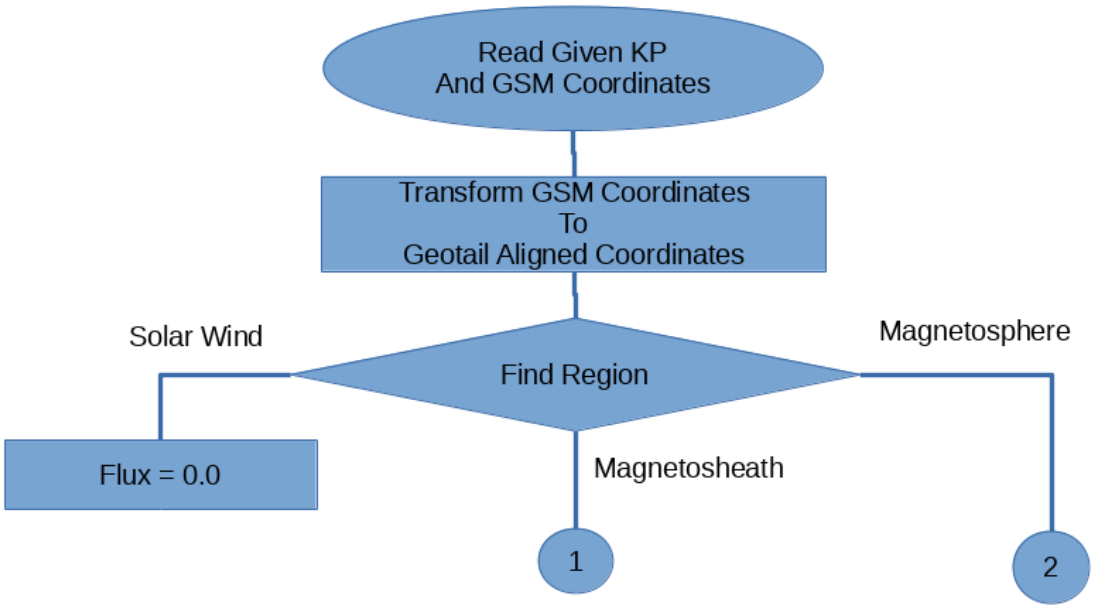
The *crmflx* code estimates an ion flux value at a satellite position of a given magnetic activity *KP* index based on the preset ion flax maps. The code was originally written in a FORTRAN in early 2000s. Since no one was maintaining it, the *MTA* group decided to rewrite it in PYTHON (v.3.6) so that it is easier to maintain in the future.

CRM Flux Computation Steps

The steps to compute Chandra specific ion fluxes are described below. Note that the original code was designed to accommodate various cases, but we shortened the code to speed up the computation.

1. Read a KP value and *GSM* coordinates of the satellite.
2. Convert the *GSM* coordinates to the Geotail aligned coordinates.
3. Find which regions (Solar Wind, Magnetosheath, Magnetosphere) the satellite is located.
4. If the satellite is in the Solar Wind region, the estimated flux is zero.
    * We do not use the flux data in the solar wind region (the outside of the bowshock), and the flux values are set to zero in this script.
5. If the satellite is in the Magonetosheath, go to the Page 2.
6. If the satellite is in the Magnetosphere, go to page 3.

Figure 1a: _CRM_ Flux Computation Step: Beginning



Ion Flux Model Data File contains:

* cell positions in x, y, z between -30 and 30 with an increment of one Earth radius at the center of the Earth as (0, 0, 0).
* nine average flux values of different KP values between 1 and 9.
* numbers of non-zero flux values used to compute the average flux values in each cell of different KP values between 1 and 9.

Notes:

* It seems that the model data tables were created with Tsyganenko's geopack.f, but we cannot confirm that. If it is true, since the geopack.f and its parameter files were updated recently, our data files are probably out of date.

Magnetosheath

If the satellite is in the magnetosheath:

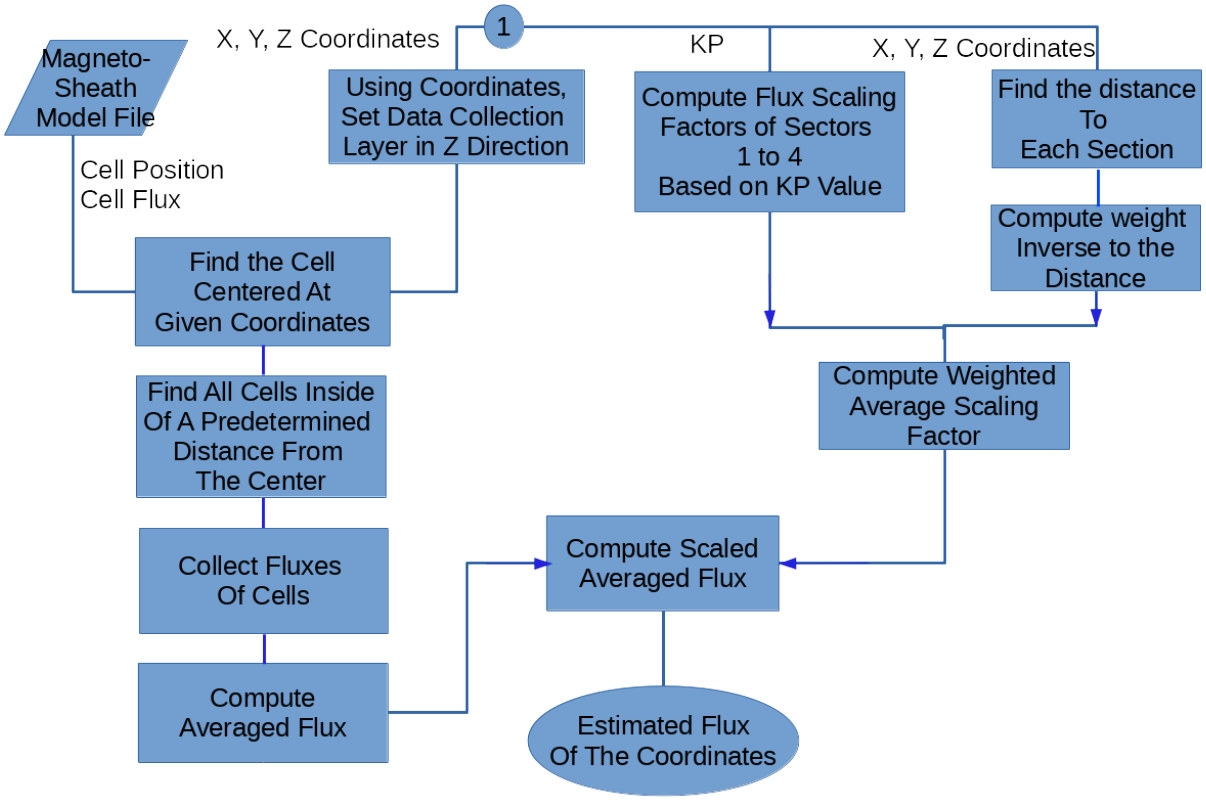
1. Compute a scaling factor.
    * divide the region into four sectors
    * compute a scaling factor of each sector based on the *KP* value given
    * find the distance to each sector from the satellite position
    * order them from the nearest to the farthest
    * compute a distance weighted scaling factor based on the scaling factors from all sectors

2. Determine how far the flux cell data should be included in the z-direction based on x value in the geotail aligned coordinates.

3. Compute an average flux.
    * from magnetosheath model ion flux database, collect the flux cell data around the satellite
    * collect all cells inside of the predetermined distance from the satellite position
    * collect flux values and take an average of the fluxes

4. Compute the scaled flux
    * multiply the scaling factor from step 1 to the averaged flux value from step 3 to estimate the final flux.

Figure1b: *CRM* Flux Computation Step: Magnetosheath



Magnetosphere

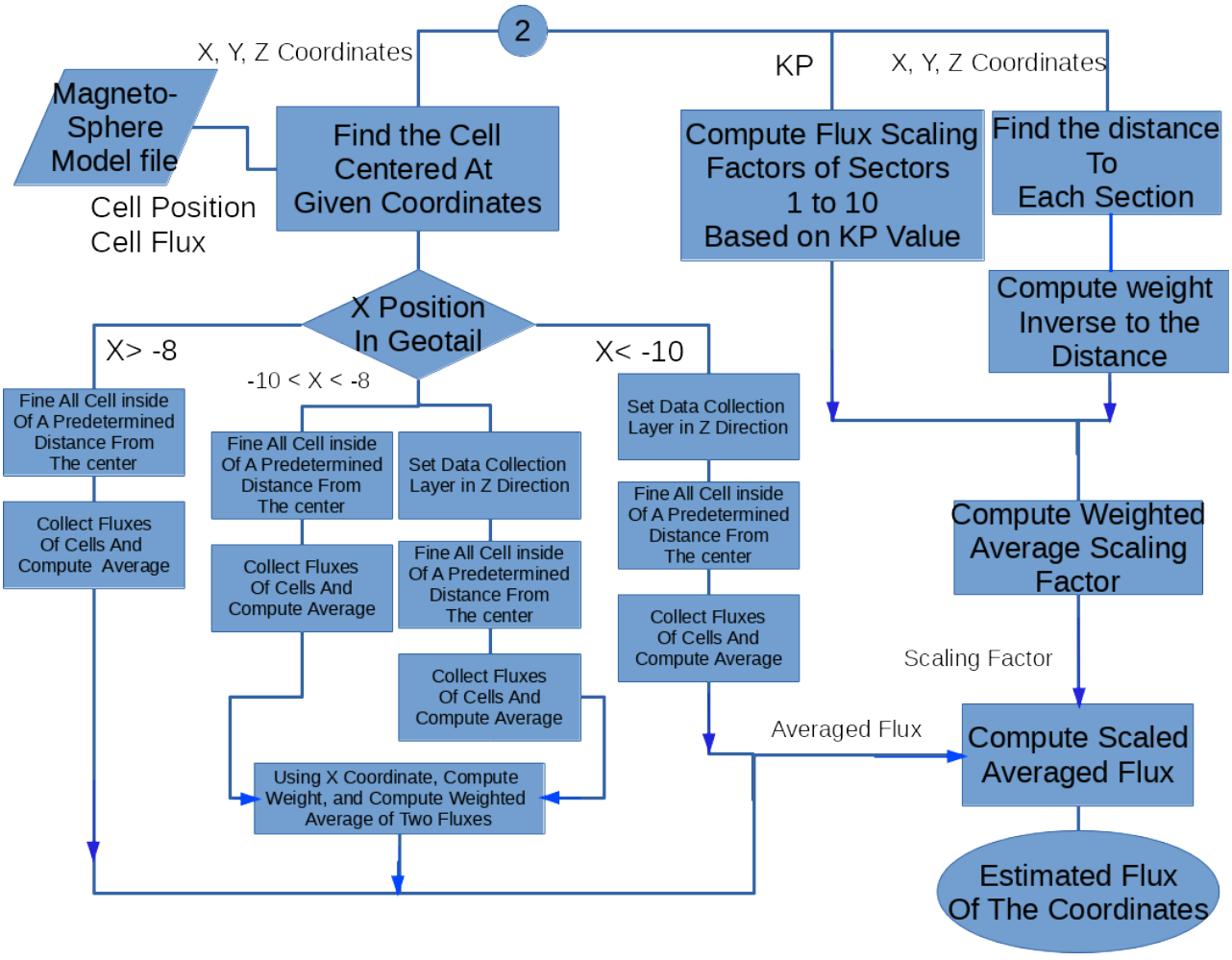
If the satellite is in the magnetosphere:

1. Compute a scaling factor.
    * divide the region into ten sectors
    * compute a scaling factor of each sector based on the *KP* value given
    * find the distance to each sector from the satellite position
    * order them from the nearest to the farthest
    * compute a distance weighted scaling factor based on the scaling factors from all sectors

2. Read data and set a cell region.
    * from magnetosphere database, collect the flux cell data around the satellite

3. If x coordinate in the geotail is in the head side or less than 8 Earth radii in the tail side:
    * use all available fluxes.
    * find all cells closer than the predetermined distances
    * collect fluxes from the cells and compute the averaged flux

Figure1c: *CRM* Flux Computation Step: Magnetosphere



4. If the satellite is located beyond 10 radii in the tail side:
   * determine a layer in the z direction where we collect data cell
   * find all cells closer than the predetermined distances in that layer
   * collect fluxes from the cells and compute the averaged flux

5. If the the satellite is between 8 and 10 radii in the tail side:
   * compute fluxes with the both step 3 and step 4 so we have two flux values.
   * compute weights for the fluxes depending of the satellite x position in the geotail
   * compute the weighted average of two flux values.

6. Using the averaged flux from either step 3, 4, or 5, and the averaged scaling factor from step 1, compute the final estimated ion flux.

How to Speed Up the PYTHON Script

The PYTHON script, which was directly translated from the FORTRAN code, took nearly sixty minutes to compute the entire *CRM* data set (FORTRAN code takes less than one minute.) To shorten the computation time, the following changes are made.

* When possible/useful, *NumPy* arrays are used which assign the memory space and speed up the computation.

* The original FORTRAN code is designed to take many options. We streamlined the code and left only essential for our needs. This reduced many condition statements and loops which could slow down the computation.

* The model data in the data table files were mostly null data. We removed them from the data file, and let the script read only valid data. This reduced the computation time slightly.

The above modifications did not speed up the code much; however, the following two steps significantly improved the computation time.

* The code is compiled with Cython. The computation time is reduced to fifteen minutes.

Reference: https://cython.readthedocs.io/en/latest/src/tutorial/cython_tutorial.html

* Use of "*typed memoryview*." The script spends the majority of time in two functions. By assigning *typed memoryviews* to the variables in these two functions, the computation time is reduced to three minutes.

Reference:: https://cython.readthedocs.io/en/latest/src/userguide/memoryviews.html

* We tried *typed memoryviews* in the other functions in the code, but it did not reduce the computation time any farther, or it actually increased the computation time in some other instances.

The Flux Discrepancy in the Magnetosphere between the FORTRAN Code and the PYTHON Code

When computing the fluxes in the magnetosphere, there are some discrepancies between values calculated by the FORTRAN code and those by the PYTHON code (***Figure 3***). It is because the FORTRAN code often misses collecting the flux values in the outer area.

The algorithm of the FORTRAN code to compute the flux is the following.

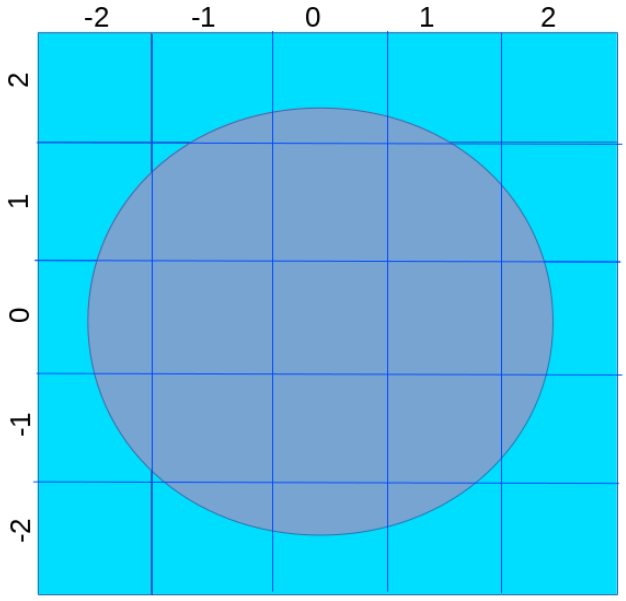
1. In the flux map of n by n by n 3D binned space, locate the satellite position.
2. Start accumulating the flux from the nearest bin to the satellite position.
3. Increment step 1 in one of the three directions and find the distance to the cell.
4. If the distance is smaller than the predetermined distance, add the flux.
5. If the distance is larger than the predetermined distance, stop.

The problem is step 5. Please see the ***Figure 2*** below. In this 2D surface, the satellite is located at the bin (0, 0) and supposed to accumulate all flux values in the bins covered by the gray circle (predetermined distance from the center).

The algorithm checks (0, 1), (0, -1), (1, 0), (-1, 0), then moves to check (0, 2), (0,-2) etc. However, once it reaches the -2/2 region, the problem happens. The code checks (0, -2), (-1, -2), (1, -2), but when it reaches (-2, -2), which is outside of the predetermined distance, it stops accumulating the fluxes. However, there are still some other bins (e.g.,(0, 2), (-1, 2)) which are inside of the predetermined distances.

The PYTHON version fixes this deficiency and covers all bins under the predetermined distance by checking region much farther than the predetermined distance, but only collecting fluxes from the cells inside of the predetermined distance.

Figure 2: Binning Example



Let's see the computation in the actual codes.

Table 1 shows the functions compute magnetosphere ion fluxes. The left side shows the FORTRAN code, and the right side shows the PYTHON code.

The first difference between the two codes is introduced in lines 8-11 in both codes. The values *ioffset/joffsee/koffset* are the bins in the previous example, and sorted in order of the distance from the center (0, 0, 0). However, since the sorting algorithms are different between two codes, the order of the offsets are different.

Please see the first several lines of ***Table 2***, which shows the outputs from these two codes. The left side is the output from the FORTRAN code, and the right side is from the PYTHON code. X, Y, Z are the values of *ioffset, joffset, and koffset.*

As you can see, X, Y, Z values appear in different orders, and the order of flux values collected are different between the two codes.

The second difference comes in while collecting the ion flux values from the cells. The two codes use the same criteria (*rngabs* <4.0) to collect the flux values (lines 23-25 in FORTRAN / lines 29-31 in PYTHON). Steps are:

1. Compute the distance from the center to the cell (*dist)*.
2. Compute the difference between the distance and the minimum distance (*rngabs)*. The minimum distance is the first entry of the table.
3. If *rngabs* is smaller than then criteria (4.0), include the flux of the cell in the computation.

However, in the codes, these steps are implemented differently . Please see red highlighted lines.

At line 33 of the FORTRAN code, if the *rngabs* value is found to be larger than the criteria (4.0), the code stops collecting fluxes.

In PYTHON, even if the *rngabs* exceeds the criteria, it keeps looking for more cells (lines 52-54) to ensure that all cells with *rngabs* value smaller than the criteria are included.

Please see ***Table 2*** again.

In FORTRAN code, the code stops collecting the flux at N=287 as *rngabs* value (4.09) exceeds 4.0 (red highlighted entries.) If we do the same in PYTHON code, it stops at N=269. Notice that flux values of N=266, 269 in FORTRAN output are not included in the PYTHON output, if the PYTHON code stops collecting fluxes at the first time when the *rngabs* exceeds the criteria (see blue highlighted entries).

(N value is the index of the flattened 3D space, but it is not essential for this discussion and think it as an ID of the flux cell. Note, FORTRAN starts at 1, but PYTHON begins at 0).

To make sure to include all fluxes inside of the criteria, PYTHON code searches farther and finds several more flux cells that meet the criteria (green highlighted entries). The FORTRAN code misses these fluxes.

Figure 3 shows fluxes computed by FORTRAN (blue) and those by PYTHON (red) in the magnetosphere region. The top figure is the case when the PYTHON stops collecting flux values at the first time when the *rngabs* value exceeds the criteria. The second figure shows that the PYTHON code keeps looking for the flux values until the father distance criteria is met.

Note that, since the different algorithm is used, this type of discrepancy does not happen in the computation of fluxes in the magnetosheath region (see ***Figure 4***).

Table 1: Magnetosphere Computation in FORTRAN and PYTHON

FORTRAN FLXDAT_MAP_Z

```
01:      indx = int((xgsm - xmin)/xinc) + 1
02:      indy = int((ygsm - ymin)/yinc) + 1
03:      indz = int((zgsm - zmin)/zinc) + 1
04:c
05:      rngcell = 1.e+25
06:      numcell = 0
07:c
08:      do i = 1,nsphvol
09:        ii = indx + ioffset(i)
10:        jj = indy + joffset(i)
11:        kk = indz + koffset(i)
12:        if ((ii.ge.1).and.(jj.ge.1).and.(kk.ge.1).and.(ii.le.maxnum)
13:     $  .and.(jj.le.maxnum).and.(kk.le.maxnum)) then
14:          indexnow = imapindx(ikp,ii,jj,kk)
15:          if(indexnow.gt.0) then
16:            fve = fluxbin(ikp,indexnow)
17:c
18:            if(fve .gt.0.) then
19:              zve = zflux(ikp,indexnow)
20:              if((zve.ge.zcklo).and.(zve.le.zckhi)) then
21:                xve = xflux(ikp,indexnow)
22:                yve = yflux(ikp,indexnow)
23:                rng = sqrt((xve-xgsm)**2 + (yve-ygsm)**2 + (zve-zgsm)**2)
24:                rngdiff = rng - rngcell
25:                rngabs = abs(rngdiff)
26:                if(numcell.eq.0) then
27:c                 there is a new nearest neighbor data cell.
28:                  numcell = 1
29:                  rngcell = rng
30:                  flxsto(1) = fve
31:                  numsto(1) = numbin(ikp,indexnow)
32:                else
33:                  if(rngabs.le.rngchk) then
34:c                   there is a new data cell within the range
35:c                   tolerance to the nearest neighbor.  this cell's flux
36:c                   should be included in the average for this location.
37:                    numcell = numcell + 1
38:                    flxsto(numcell) = fve
39:                    numsto(numcell) = numbin(ikp,indexnow)
40:                    if(numcell.eq.maxcell) go to 1000
41:                  else
42:                    go to 1000
43:                  end if
44:                end if
45:              end if
46:            end if
47:          end if
48:          end if
49:        end do
50:c
51:1000  continue
52:c
53:c     use the average of the flux from all bins at the same distance.
54:c
55:      flux = 0.
56:      avgnum = 0.
57:      if(numcell.eq.1) then
58:        flux = flxsto(1)
59:        avgnum = float(numsto(1))
60:      else if(numcell.gt.1) then
61:        numavg = 0
62:        do i = 1,numcell
63:          flux = flux + flxsto(i)
64:          numavg = numavg + numsto(i)
65:        end do
66:        flux = flux/float(numcell)
67:        avgnum = float(numavg)/float(numcell)
68:      end if
69:c
70:      return
71:      end
```

PYTHON FLXDAT_MAP(_Z)

```
01:    rngcell = 1.0e25
02:    numcell = 0
03:#
04:#-- give extra margin on rngchk so that most of the neighbors are included
05:#
06:    rngchk2 = 1.20 * rngchk
07:
08:    for n in range(0, nsphvol):
09:        i = indx + ioffset[n]
10:        j = indy + joffset[n]
11:        k = indz + koffset[n]
12:        if (i >= 0) and (j >= 0) and (k >= 0)\
13:           and (i < maxnum) and (j < maxnum) and (k < maxnum):
14:            indexnow = imapindx[i][j][k]
15:
16:            if indexnow >= 0:
17:                zve     = zflux[indexnow]
18:#
19:#--- for the map_z, extra check is required
20:#
21:                if zchk > 0:
22:                    if (zve < zcklo) or (zve > zckhi):
23:                        continue
24:
25:                fve     = fluxbin[indexnow]
26:                xve     = xflux[indexnow]
27:                yve     = yflux[indexnow]
28:
29:                rng     = compute_rng(xve, yve, zve, xgsm, ygsm, zgsm)
30:                rngdiff = rng - rngcell
31:                rngabs  = abs(rngdiff)
32:#
33:#--- there is a new nearest neighbor data cell
34:#
35:                if numcell == 0:
36:                    flxsto[numcell] = fve
37:                    numsto[numcell] = numbin[indexnow]
38:                    numcell = 1
39:                    rngcell = rng
40:#
41:#--- there is a new data cell within the range tolerance to the nearest neighbor.
42:#--- this cell's flux should be included in the average for this location.
43:#
44:                else:
45:                    if rngabs <= rngchk:
46:                        flxsto[numcell] = fve
47:                        numsto[numcell] = numbin[indexnow]
48:                        numcell += 1
49:
50:                        if numcell > maxcell-1:
51:                            break
52:                    else:
53:                        if rngabs > rngchk2:
54:                            break
55:#
56:#--- use the average of the flux from all bins at the same distance
57:#
58:    if numcell == 0:
59:        flux    = 0.0
60:        avgnum  = 0.0
61:        rngcell = 0.0
62:
63:    elif numcell == 1:
64:        flux    = flxsto[0]
65:        avgnum  = float(numsto[0])
66:
67:    elif numcell > 1:
68:        flux    = numpy.mean(flxsto[:numcell])
69:        avgnum  = numpy.mean(numsto[:numcell])
70:
71:    return flux, avgnum, rngcell, numcell
```

Table 2: Output from FORTRAN and PYTHON Codes

FORTRAN Output

N	x	y	z	flux	dist	rngabs
1	0	0	0	70.0400009	0.47384930	0.0
3	1	0	0	497.799988	1.33221877	0.85836947
4	0	1	0	1160.00000	1.41181505	0.93796575
6	0	-1	0	27.3500004	0.67516255	0.20131328
7	-1	0	0	59.8600006	0.82113289	0.34728363
9	-1	-1	0	25.5799999	0.95161473	0.47776547
12	-1	1	0	67.3799973	1.56299329	1.08914399
17	1	1	0	127.500000	1.88241756	1.40856826
19	1	-1	0	95.4100037	1.41637504	0.94252574
28	-2	0	0	68.8399963	1.76747990	1.29363060
29	0	-2	0	36.5499992	1.63925457	1.16540527
31	0	2	0	1514.00000	2.40039778	1.92654848
41	-1	-2	0	9.99199963	1.77112448	1.29727519
42	1	-2	0	102.900002	2.05850172	1.58465242
44	-2	-1	0	29.7800007	1.83174682	1.35789752
45	-2	1	0	45.4300003	2.21193886	1.73808956
53	-1	2	0	1441.00000	2.49231529	2.01846600
55	1	2	0	82.8099976	2.70410490	2.23025560
82	2	2	0	79.0800018	3.29582429	2.82197499
88	-2	-2	0	31.4799995	2.36360073	1.88975143
90	-2	2	0	681.500000	2.94301915	2.46916986
100	0	-3	0	26.8600006	2.63029790	2.15644860
105	0	3	0	67.9899979	3.39567375	2.92182446
107	-3	0	0	17.7500000	2.75203776	2.27818847
130	-1	3	0	51.7700005	3.46126080	2.98741150
133	1	-3	0	20.1299992	2.91011000	2.43626070
137	-3	-1	0	30.6399994	2.79374695	2.31989765
140	1	3	0	59.7099991	3.61674905	3.14289975
142	-3	1	0	49.9799995	3.05653381	2.58268452
147	-1	-3	0	35.6300011	2.71444154	2.24059224
184	-3	2	0	563.799988	3.62092352	3.14707422
187	2	3	0	70.0599976	4.07813025	3.60428095
193	-2	3	0	32.3100014	3.79869080	3.32484150
253	0	4	0	72.4899979	4.39309502	3.91924572
257	-4	0	0	53.5299988	3.74478793	3.27093863
263	-1	-4	0	31.9200001	3.68775058	3.21390128
266	-1	4	0	14.0500002	4.44398594	3.97013664
269	-4	-1	0	15.8400002	3.77554631	3.30169702
284	-4	1	0	41.6899986	3.97393084	3.50008154
287	1	4	0	70.4400024	4.56613159	4.09228230

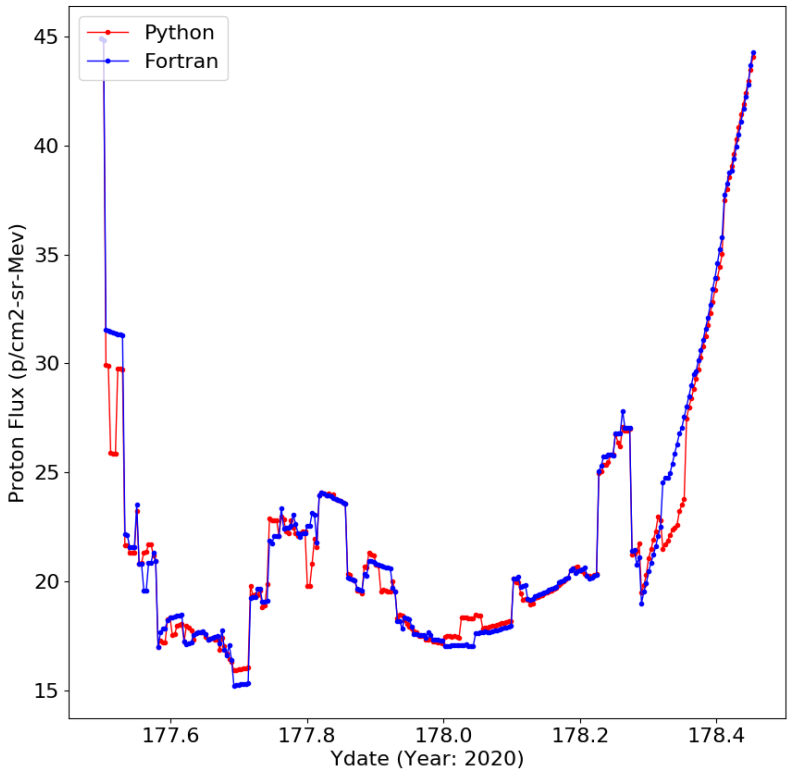
flux: 192.575165

PYTHON Output

N	x	y	z	flux	dist	rngabs
0	0	0	0	70.04000	0.47385	0.0
1	1	0	0	497.8000	1.33222	0.85837
2	-1	0	0	59.86000	0.82113	0.34728
3	0	1	0	1160.000	1.41182	0.93797
6	0	-1	0	27.35000	0.67516	0.20131
9	1	-1	0	95.41000	1.41638	0.94253
14	-1	1	0	67.38000	1.56299	1.08914
17	-1	-1	0	25.58000	0.95161	0.47776
18	1	1	0	127.5000	1.88242	1.40857
29	0	-2	0	36.55000	1.63925	1.16540
30	0	2	0	1514.000	2.40040	1.92655
31	-2	0	0	68.84000	1.76748	1.29363
35	-1	2	0	1441.000	2.49232	2.01847
42	1	2	0	82.81000	2.70411	2.23026
45	-2	1	0	45.43000	2.21194	1.73809
46	-2	-1	0	29.78000	1.83175	1.35790
47	-1	-2	0	9.992000	1.77112	1.29727
55	1	-2	0	102.9000	2.05850	1.58465
82	-2	2	0	681.5000	2.94302	2.46917
86	2	2	0	79.08000	3.29583	2.82198
87	-2	-2	0	31.48000	2.36360	1.88975
93	0	-3	0	26.86000	2.63030	2.15645
104	0	3	0	67.99000	3.39567	2.92182
112	-3	0	0	17.75000	2.75204	2.27819
124	1	-3	0	20.13000	2.91011	2.43626
132	-1	3	0	51.77000	3.46126	2.98741
133	-3	-1	0	30.64000	2.79375	2.31990
135	-1	-3	0	35.63000	2.71444	2.24059
136	-3	1	0	49.98000	3.05653	2.58268
138	1	3	0	59.71000	3.61675	3.14290
191	2	3	0	70.06000	4.07813	3.60428
192	-3	2	0	563.8000	3.62092	3.14707
201	-2	3	0	32.31000	3.79869	3.32484
253	-4	0	0	53.53000	3.74479	3.27094
254	0	4	0	72.49000	4.39310	3.91925
265	-1	-4	0	31.92000	3.68775	3.21390
268	-4	1	0	41.69000	3.97393	3.50008
269	1	4	0	70.44000	4.56613	4.09228
270	-1	4	0	14.05000	4.44399	3.97014
272	-4	-1	0	15.84000	3.77555	3.30170
294	1	-4	0	164.3000	3.83406	3.36022
320	-3	-3	0	19.27000	3.77725	3.30340
325	-3	3	0	19.95000	4.34509	3.87124
366	2	4	0	2862.000	4.93962	4.46577
368	-4	-2	0	17.25000	4.06030	3.58645
369	-4	2	0	1247.000	4.42276	3.94891
387	-2	4	0	29.42000	4.71155	4.23770
485	0	-5	0	31.83000	4.62397	4.15012
486	-4	3	0	694.3000	5.03284	4.55899
487	0	5	0	4057.000	5.39147	4.91762

flux: 204.0500

Figure 3: Flux Comparison between FORTRAN and PYTHON: Before and After PYTHON Code modification



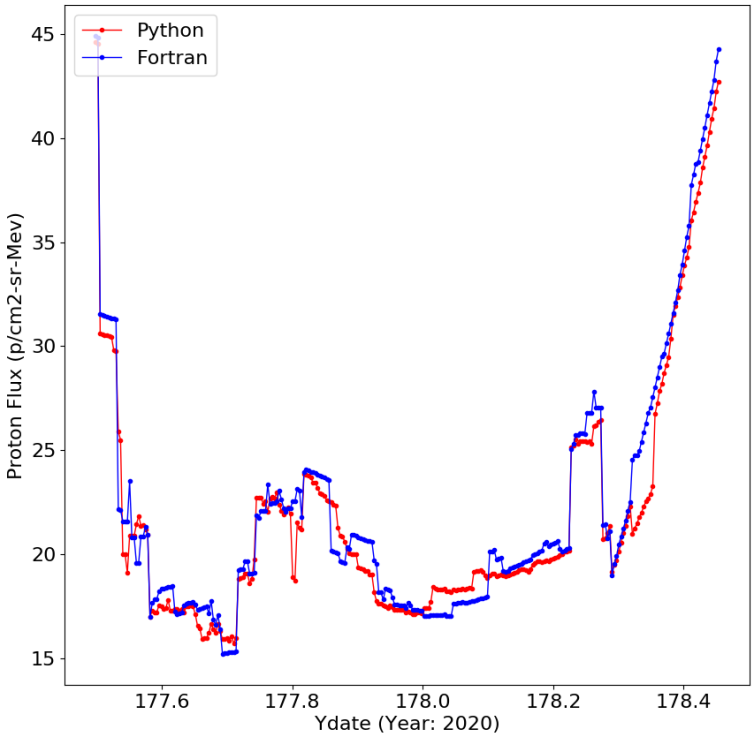
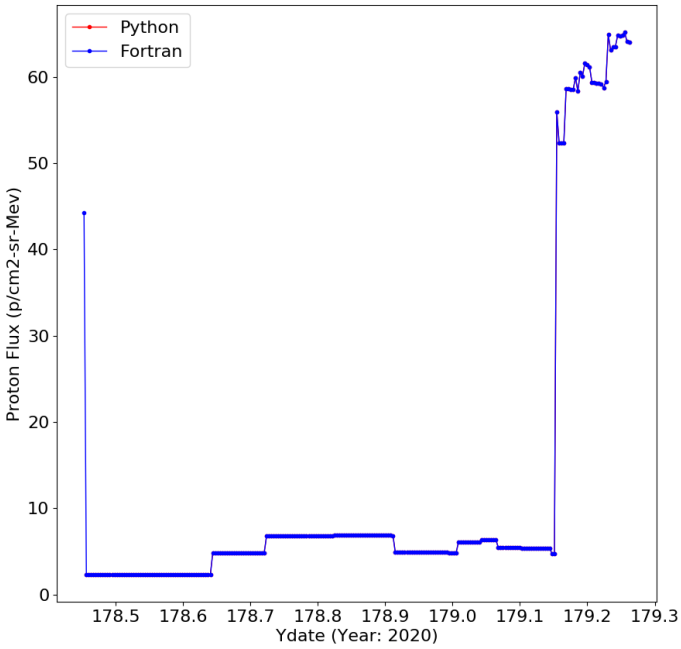


Figure 4: Flux Comparison of FORTRAN and PYTHON: Magnetosheath



The Script Details

runcrmf.py:
==========
This is a control PYTHON code. It reads the model data files, EPHEM data in GSM coordinates, and sets up KP values.
It calls swinit, mshinit, and mspinit (from crmflx.py), and run the main function crmflx (from crmflx.py)

crmflx.pyx
==========

crmflx:
----
the primary function to calculates the ion flux as a function of the magnetic activity kp index.
main inputs: (X, Y, Z) GSM coordinates of the satellite
KP index
Magnetsheath data table
Magnetosphere data table
Data table binning map

output: ion flux at the GSM coordinates

function used: locreg --- identify which region the satellite is in
scalkp2 --- collect data from sections in the magnetosheath
nbrflux --- compute the ion flux in the magnetosheath
scalkp3 --- collect data from sections in the magnetosphere
nbrflux_map_z --- compute the ion flux in the magnetosphere

mspinit:
----
read magnetosphere database
mshinit:
----
read magnetosheath database
swinit:
----
read solar wind database

output: cell positions in x, y, z between -30 and 30 with an increment of one Earth radius at the center of the Earth as (0, 0, 0).
average flux value of each cell of different KP values between 1 and 9.
numbers of non-zero flux values used to compute the average flux in each cell of different KP values between 1 and 9.

function used: read_init_data_file --- read the data

mapsphere:
---
finds the (i,j,k) index offset values used to define the search volume for the near-neighbor flux

input: step and step sizes
output: sets of offset in x, y, z coordinates

logreg:
---
determines which phenomenological region the spacecraft is in

input: KP value
GSM coordinates of the satellite

output: id to indicate in which region the satellite is in
x, y, z coordinates in the geotail system.

function used: solwnd --- set solar wind parameters
rog8ang --- rotate coordinates
locate --- defines the position of a point at the model magnetopause
bowshk2 --- give the bow shock radius, at a given x

solwind:
---
set solar wind parameters

input: KP value
output: a set of solar wind parameters for the given KP value

bowshk2:
---
give the bow shock radius, at a given x

input: a set of solar wind parameters given by "solwind" function
output: cylindrical radius of the bow shock

function used: fast --- local fast magnetosonic speed

fast:
---
local fast magnetosonic speed

input: a set of solar wind parameters
output: local fast magnetosonic speed

locate:
---
defines the position of a point at the model magnetopause

input: gsm coordinates
       solar wind proton density/ram pressure/velocity

output: coordinates of a point at the magnetopause

function used: compute_rng --- compute a distance between two points

nbrflux:
---
provides the region's ion flux as a function of k

input: sector information
          geotail coordinates
          data tables
output: estimated flux

function used: neighbr --- the nearest neighbors to the point
              wtscal --- compute weighted scaling factors
              zbinner --- find which z layer we should collect data
              flxdat1 --- finds the flux corresponding to the satellite's position

flxdat1
---
finds the flux corresponding to the satellite's position (magnetosheath)

input: satellite coordinates
          data table
          range in the z-direction of data collection
output: flux

function used: comput_rng --- compute a distance between two points

nbrflux_map_z:
---
provides the region's ion flux as a function of k
this function is used magnetosphere computation and requires distinguishing where in the geotail the satellite is located. See the algorithm page for more details.

input: sector information
          geotail coordinates
          data tables
output: estimated flux

function used: neighbr   --- the nearest neighbors to the point
              wtscal   --- compute weighted scaling factors
              zbinner   --- find which z layer we should collect data
              flxdat1_map --- finds the flux corresponding to the satellite's position

flxdat1_map:

---
finds the flux corresponding to the satellite's position (magnetosphere)

input: satellite coordinates
data table
range in the z-direction of data collection
output: flux

function used: comput_rng --- compute a distance between two points

compute_rng:
compute a distance between two points

input: two coordinates
output: the distance between two coordinate

neighbr:
---
finds the nearest neighbors from the given location to the sector location

input: satellite x, y coordinates
arrays of sector locations

output: sorted ranking of distance to each sector

rot8ang:
---
rotates the 2-d vector about its hinge point in the XY-plane

input: angle, x, y, and x value of aberration hinge point

output: computed x, y values

scalkp1:
---
compute scaling parameters in the solar wind region
scalkp2:
---
compute scaling parameters in the magnetosheath region
scalkp3:
---
compute scaling parameters in the magnetosphere region

input: KP value
output: scaling factors

function used: get_scalkp --- compute the scaling factor

get_scalkp:
---
compute the scaling factor

input: KP value
a list of section functions (see sectr** below)

output: scaling factors

sectr11 --- sectr13:
---
provide the proton flux against kp scaling for the solar wind region
input: KP value
output: scaling factor

function used: sect_comp --- compute scaling factor for the given sector

sectr21 --- sectr24:
---
provide the proton flux against kp scaling for the magnetosheath region
input: KP value
output: scaling factor

function used: sect_comp --- compute scaling factor for the given sector

sectr31 --- sectr310:
---
provide the proton flux against kp scaling for the magnetosphere region
input: KP value
output: scaling factor

function used: sect_comp --- compute scaling factor for the given sector

sect_comp:
---
compute scaling factor for given parameters

input: KP values
a parameter list
output: scaling factors

wtscal:
---
calculates the distance weighted sum of the Kp scaling factors

input: sector information including flux scale factor
distances to each sector

output: distance weighted scaling factor

function used: create_weighted_sum --- calculate a weighted sum

create_weighted_sum:
---
calculate a weighted sum

y_interpolate:
---
interpolate the coordinates between two points
input: coordinates of two points
x coordinates of the third point
output: estimated y coordinate

zbinner:
---
determine the z-layer of the magnetosphere

input: x and z gsm coordinates
output: bottom and top z coordinates