jbldarf 徹底ガイド

日本語 第 1.2 版

石崎 欣尚

東京大学大学院理学系研究科

ishisaki@miranda.phys.s.u-tokyo.ac.jp

平山 昌治

東京大学大学院理学系研究科

hirayama@astro.isas.ac.jp

石田 学

宇宙科学研究所宇宙圏研究系

ishida@astro.isas.ac.jp

1995 年 4 月 01 日　　第 1.0 版

1995 年 7 月 20 日　　第 1.1 版

1995 年 8 月 27 日　　第 1.2 版

目次

はじめに

jbldarf は簡単なパラメーターの設定により、 SIS と GIS の ARF(Auxiliary/ Ancillary Response Function[1]) を作るソフトです。このマニュアルでは、 **jbldarf** の使い方から適用限界にいたるまで、 **jbldarf** に関するさまざまな解説がなされいます。内容は、現時点での最新バージョンである 2.10 に基づいて書かれています。jbldarf-1.00 と比べて新しくなった節には、欄外に ◁U▷ (UPDATED) や ◁N▷ (NEW) のマークをつけてあります。

第 1章では、 **jbldarf** のインストール方法を説明します。あなたが使っている計算機に **jbldarf** をインストールするときにお読み下さい。すでに誰かの手によってインストールがされているならここを読む必要はありません。インストールされているかどうかわからない人は、あなたが使っている計算機の管理者に問い合わせてみて下さい。

第 2章では、 **jbldarf** の具体的な操作法や、リージョン、ポジションなどの基本的な概念について説明されています。初心者の方は、ここから読まれることをお勧めします。また、 **jbldarf** に十分慣れている方でも、忘れてしまったコマンドの使い方を思い出すためのコマンドリファレンスとしてお役立て下さい。

第 3章には、 **jbldarf** をより良く理解していただくための詳しい説明があります。 ARF の概念やファイル構造が記述され、 **jbldarf** がどのようなデータに基づいてどのような計算をして ARF を作成しているのかを説明しています。その上で、現時点で考えうる問題点や **jbldarf** の適用限界についても触れ、やや高度な応用についても紹介します。この章では、第 2章での知識を前提にしますので、始めて読まれる方は、この章を読まれる前に第 2章を読んで下さい。また、 **jbldarf** をお使いになられる方はこの章までは必ずお読みになって下さい。

第 4章では、 **jbldarf** のリリースキットに附属している、 ARF 関連のツールを紹介します。ツールの使い方を知りたい時など必要に応じてお読み下さい。

第 5章では、これまでにユーザの皆様から著者宛に寄せられたさまざまなご意見の中から、比較的よく耳にする疑問や質問を集めて作った「質問 & 回答」集です。必要に応じてご参考になさって下さい。

jbldarf に関するご意見やご質問などは著者までお知らせ下さい。解析してみて始めてわかった、レスポンスに関する新事実などもお待ちしております。 **jbldarf** にみなさんのご意見や解析結果などを反映させ、より正確で、より使いやすい **jbldarf** にしていきましょう。

[1]XSPEC では auxiliary、 FTOOLS では ancillary という単語が使われているようである。

第 1 章

インストール

この章では **jbldarf-2.10** のインストールの方法について説明します。

1.1 どこにあるか？

jbldarf-2.10 は圧縮された形で宇宙研の anonymous ftp サイトに置かれています。

`ftp://ftp.astro.isas.ac.jp/asca/software/isas/jbldarf-2.10.tar.gz`

を取ってきて展開して下さい。カレント・ディレクトリに **jbldarf-2.10/** というディレクトリができます。

1.2 コンパイルの方法

jbldarf-2.10/ の下にはいくつかのファイルと 2 つのディレクトリ (**src/** と **doc/**) があります。まずインストールの方法が書かれている **INSTALL** を読んで下さい。この中にコンパイルの方法が書かれています。コンパイルに必要なすべてのファイルは **src/** に収められています。またこのマニュアルを含む **jbldarf** に関するドキュメントが **doc/** にあります。

1.3 パラメーター・ファイル ◁U▷

jbldarf は ftools と同じようにパラメーター・ファイルを持っています。パラメーター・ファイルは

`jbldarf-2.10/jbldarf.par`

にあります。 **jbldarf-2.10** のパラメーター・ファイルは **jbldarf-2.10** より古いバージョンとの互換性がありませんので注意が必要です。

jbldarf-2.10 のパラメーター・ファイルは、展開直後は以下のようになっています。

```
Auto_build,b,h,no,,,"for automatic building"
XRT_EA_table,s,h,"/caldb/data/asca/xrt/bcf/xrt_ea_110494.fits",,,"XRT effective area table file"
XRT_PSF_type,s,h,"Table",,,"type of XRT PSF data (Analytic/Table)"
XRT_PSF_table,s,h,"/asca/caldb/xrt_psf_V1.1.fits",,,"table file of XRT point spread function"
XRT_PSF_energy_step,r,h,0,,,"Energy step in keV for XRT point spread function"
S0_teldef,s,h,"/caldb/data/asca/sis/bcf/s0_teldef_070294.fits",,,"teldef file for SIS0"
S1_teldef,s,h,"/caldb/data/asca/sis/bcf/s1_teldef_070294.fits",,,"teldef file for SIS1"
S2_teldef,s,h,"/caldb/data/asca/gis/bcf/gis2_ano_on_flf_180295.fits",,,"teldef file for GIS2"
S3_teldef,s,h,"/caldb/data/asca/gis/bcf/gis3_ano_on_flf_180295.fits",,,"teldef file for GIS3"
S2_grid_map,s,h,"/caldb/data/asca/gis/bcf/s2gridv3.fits",,,"GIS2 grid map FITS file"
S3_grid_map,s,h,"/caldb/data/asca/gis/bcf/s3gridv3.fits",,,"GIS3 grid map FITS file"
S2_Be_map,s,h,"/caldb/data/asca/gis/bcf/s2bev1.fits",,,"GIS2 Be thickness map FITS file"
S3_Be_map,s,h,"/caldb/data/asca/gis/bcf/s3bev1.fits",,,"GIS3 Be thickness map FITS file"
S2_Xe_thick,r,h,1.0,,,"GIS2 Xe thickness in cm"
```

```
S3_Xe_thick,r,h,1.0,,,"GIS3 Xe thickness in cm"
Detector_ID,i,h,0,,,"sensor ID (0-3)"
CHIPMASK_0,b,h,no,,,"mask SIS region map with Chip-0?"
CHIPMASK_1,b,h,no,,,"mask SIS region map with Chip-1?"
CHIPMASK_2,b,h,no,,,"mask SIS region map with Chip-2?"
CHIPMASK_3,b,h,no,,,"mask SIS region map with Chip-3?"
RMF_file,s,h,"default",,,"reference RMF file"
S0C0_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c0g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C0"
S0C1_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c1g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C1"
S0C2_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c2g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C2"
S0C3_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s0c3g0234p40e0_512v0_8i.rmf",,,"rmf file for S0C3"
S1C0_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c0g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C0"
S1C1_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c1g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C1"
S1C2_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c2g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C2"
S1C3_rmf,s,h,"/caldb/data/asca/sis/cpf/94nov9/s1c3g0234p40e0_512v0_8i.rmf",,,"rmf file for S1C3"
S2_rmf,s,h,"/caldb/data/asca/gis/cpf/95mar06/gis2v4_0.rmf",,,"rmf file for GIS2"
S3_rmf,s,h,"/caldb/data/asca/gis/cpf/95mar06/gis3v4_0.rmf",,,"rmf file for GIS3"
Output_file,s,h,"a.arf",,,"output ARF file"
Region_type,s,h,"Circle",,,"type of integrated region (FITS-file/Circle)"
Region_file,s,h,"reg.fits",,,"selected region file"
Center_X,r,h,0,,,"data region center x position (mm)"
Center_Y,r,h,0,,,"data region center y position (mm)"
Inner_radius,r,h,0,,,"data region inner radius (mm)"
Outer_radius,r,h,3,,,"data region outer radius (mm)"
SIS_region_rebin_X,i,h,1,,,"binning factor for SIS region map in x axis"
SIS_region_rebin_Y,i,h,1,,,"binning factor for SIS region map in y axis"
Position_type,s,h,"One-Point",,,"type of source position (FITS-file/One-Point)"
Position_file,s,h,"pos.fits",,,"source position file"
Source_X_pos,r,h,0,,,"source x position (mm)"
Source_Y_pos,r,h,0,,,"source y position (mm)"
SIS_position_rebin_X,i,h,1,,,"binning factor for SIS position map in x axis"
SIS_position_rebin_Y,i,h,1,,,"binning factor for SIS posotion map in y axis"
SIS_kobu,b,h,no,,,"Whether to multiply "kobu" factor by SIS ARF"
DEBUG,i,h,0,,,"debug flag"
clobber,b,h,no,,,"Overwrite output file if it already exists ?"
mode,s,h,"ql",,,""
```

ここでキャリブレーション・ファイルの置き場所などは宇宙研での設定になっていますので、他の機関ではリリース前に適当に書き直しておく必要があります。

宇宙研では **jbldarf-2.10.tar.gz** は展開されて

```
/adsoft/jbldarf/
```

の下に置かれていて、**jbldarf-2.10/** は単に **2.10/** と rename されています。サンプルとなる **jbldarf.par** は

```
/adsoft/jbldarf/2.10/
```

にあります。

第 2 章

はじめての jbldarf

この章では **jbldarf-2.10** の簡単な使い方について説明します。

まず、パラメータファイルを自分の $LOCPFILES 領域にコピーして下さい。パラメータファイルがどこにあるかは、あなたがお使いの計算機に **jbldarf** をインストールした人に聞いて下さい。なお、**jbldarf-2.10** のパラメーター・ファイルは **jbldarf-2.10** より古いバージョンとの互換性がありませんので注意が必要です。

パスが正しく切ってあれば、コマンドラインから **jbldarf** と打つだけで **jbldarf** が起動できます。“CMD>” というプロンプトに続けて “show” と打つと次のようなメッセージが返されます。

```
CMD> show
                   ++==================================++
                   || jbldarf: Parameters and Status ||
                   ++==================================++

 Integrated Region
             Region type : Circle
           Center X (mm) :        0.0000000000000000
           Center Y (mm) :        0.0000000000000000
       Inner radius (mm) :        0.0000000000000000
       Outer radius (mm) :        3.0000000000000000
 Target Position
           Position type : One-Point
       Source X pos (mm) :        0.0000000000000000
       Source Y pos (mm) :        0.0000000000000000
 Map Rebinning (for SIS only)
       Region map X-axis :            1
       Region map Y-axis :            1
     Position map X-axis :            1
     Position map Y-axis :            1
 Detector ID
                  Sensor :  SIS-0
 Chip Mask (for SIS only)
        Masking chip(s) :  No Mask
 Kobu (for SIS only)
          "Kobu" factor :  NOT mutiplyed
 File Specification
          RMF file name : /caldb/data/asca/sis/cpf/94nov9/s0c1g0234p40e0_512v0_8i.rmf (default)
       Output file name : a.arf
 XRT PSF
           XRT PSF type : Table
      EA table file name : /caldb/data/asca/xrt/bcf/xrt_ea_110494.fits
     PSF table file name : /asca/caldb/xrt_psf_V1.1.fits
          Calculated at : Each energy bin
```

```
 CMD:  ********** Command **********

 Show-Parameters     Region-Type         Integrated-Region
 Position-Type       Target-Position     Map-Rebinning
 Detector-ID         Chip-Mask           Kobu-for-SIS
 File-Specification XRT-PSF-Type        Build-ARF
 Exit                Quit

 CMD: Select  1 Option

CMD>
```

このメッセージには **jbldarf** で設定可能な全てのパラメータとそのステータスが含まれていますので、"CMD>" というプロンプトに続けて "show" と打てば、いつでも現在のパラメーターの値をチェックすることができます。**jbldarf** でユーザーが設定するのは "Integrated Region"、"Target Position"、"Detector ID"、"File Specification"、"XRT PSF" という 5 つの項目で、SIS の場合にはこれに加えてオプションとして "Map Rebinning"、"Chip Mask"、"Kobu-for-SIS" という 3 つの項目を設定することができます[1]。

jbldarf で積分領域やターゲットの位置を指定する時には DETX/DETY 座標系を使います。XSELECT で XYNAME に DETX/DETY を指定して "bin image" し、できた image を SAOIMAGE で読めば DETX/DETY 座標系でのターゲットの位置が pixel 単位で得られます。**jbldarf** の入力にはこれを "mm" に変換したものを用います (セクション 3.1.1)。

2.1 Integrated Region

Integrated Region では、ターゲットのフラックスを集積する時の積分領域の中心とその大きさを指定します。但し現時点ではまだ以下の点に注意が必要です。

XRT で点源を見た場合、各 Quadrant に対応して 4 枚の羽ができます。実際には 4 枚の羽のそれぞれでエネルギー・レスポンスが異なりますが、現在の XRT の応答関数にはこの効果が含まれておらず、角度方向に積分して平均した時のみ正しい応答が得られるようになっています。**jbldarf** ではこの点を考慮し、円、または円環状の積分領域しか選べないようになっています。また積分領域の中心がターゲットの位置と同じでない場合には、出来上がった ARF の精度は保証されません。XSELECT で "bin spectrum" する際にはこの点に注意して下さい。

積分領域を指定するモードに入るには、"CMD>" プロンプトに続けて "integ" と打ちます。そうするとプロンプトが "REG>" に変わります。

2.1.1 pixel から mm への変換

ここで検出器上での積分領域の中心と外径 (必要なら内径も) を "mm" 単位で入力します。変換に必要なパラメーターは以下の通りです。

(a) pixel の大きさ :
0.027mm/pixel for the SIS
0.250mm/pixel for the GIS

(b) 検出器の中心座標 (DETX, DETY) :
(640.5 pixel, 640.5 pixel) for the SIS
(128.5 pixel, 128.5 pixel) for the GIS

続いて SIS、GIS それぞれについて、以下に簡単な例を示します。

[1] **jbldarf-0.45** までは検出器の calibration の目的で各 detector の光軸位置を変えることができましたが、バージョン 0.50 以降は "teldef" ファイルから自動的に読み込まれるようになりました。

2.1.2 SIS の場合

SAOIMAGE で読みとった積分領域の中心が (715 pixel, 678 pixel)、半径が 185 pixel の場合、

```
X: (715-640.5) * 0.027 = 2.01 mm,
Y: (678-640.5) * 0.027 = 1.01 mm,
R: (185+  0.5) * 0.027 = 5.01 mm,
```

となります。“REG>” というプロンプトに続けて

```
REG> x 2.01 y 1.01 o 5.01
```

と打てば積分領域が指定できます。もし内径が 2.00mm だったら

```
REG> x 2.01 y 1.01 i 2.00 o 5.01
```

と打てば OK です。

2.1.3 Example for the GIS

SAOIMAGE で読みとった積分領域の中心が (137 pixel, 133 pixel)、半径が 20 pixel の場合、

```
X: (137-128.5) * 0.25 = 2.13 mm,
Y: (133-128.5) * 0.25 = 1.13 mm,
R: ( 20+  0.5) * 0.25 = 5.13 mm,
```

となります。“REG>” というプロンプトに続けて

```
REG> x 2.13 y 1.13 o 5.13
```

と打てば積分領域が指定できます。もし内径が 2.00mm の場合には

```
REG> x 2.13 y 1.13 i 2.00 o 5.13
```

と打てば OK です。

2.1.4 Region Map

◁ U ▷

jbldarf では積分領域をイメージ FITS ファイルから読み込むことができます。このファイルを領域ファイルと呼びます。現在は領域ファイルを作るために **makemap** という tool が用意されています。

```
/adsoft/jbldarf/2.10/{ulx,osf}/makemap.
```

使い方はコマンドラインから **makemap** と打てば見ることができて、

```
usage: makemap teldef cent-x cent-y r-out [r-in] (mm)
```

となります。GIS、SIS それぞれについて “teldef” ファイルを指定する必要がありますが、これは

```
/caldb/data/asca/sis/bcf/s0_teldef_070294.fits
                         s1_teldef_070294.fits
                         gis2_ano_on_flf_180295.fits
                         gis3_ano_on_flf_180295.fits
```

に置かれています。従って、セクション 3.1.2 の SIS の積分領域を表す領域ファイル “region.fits” を SIS-0 について作るには

```
makemap /caldb/data/asca/sis/bcf/s0_teldef_070294.fits 2.01 1.01 5.01 2.00 > region.fits
```

とします。この region.fits は SAOIMAGE で見るとわかるように、積分領域に含まれるピクセルには 1 が、含まれないピクセルには 0 が入ったイメージ FITS ファイルです。

jbldarf にこのファイルを読ませるには以下のようにします。まず "CMD>" プロンプトに続けて "region" と入力します (Region-Type の省略)。

```
CMD> region
```

すると以下のようなメッセージが出て、プロンプトが "TYP>" に変わります。

```
TYP:  ********** Region Type **********

FITS-file Circle
TYP: Select  1 Option
TYP>
```

ここで "fits" と打って "FITS-file" の方を選んで下さい。"CMD>" プロンプトの状態に戻って "integ" と打てば、今作った領域ファイルを入力することができます。ただし SIS の場合にはイメージのピクセル数が多いため、ARF を作る時に、場合によっては膨大な時間がかかることがあります。このため **jbldarf** にはイメージを任意の大きさでビンまとめする機能があります (セクション 3.6)。

2.2 Target Position

ターゲットの位置を指定するには以下の二通りの方法があります。

2.2.1 One-Point

この方法は点源に対してのみ有効で、点源の位置を検出器上の座標で手で入力します。最初に show コマンドで Position type のステータスが "One-Point" になっていることを確認して下さい。もしそうでない場合には "CMD>" プロンプトが出ている状態で "position" と入力して下さい ("Position-Type" の省略形)。すると次のようなメッセージが出ます。

```
CMD> position

 TYP:  ********** Position Type **********

 FITS-file     One-Point     FFT-FITS-file XRT-PSF

 TYP: Select  1 Option
TYP>
```

ここで "one" と打って One-Point を選択します。

```
TYP> one

 CMD:  ********** Command **********

 Show-Parameters    Region-Type        Integrated-Region
 Position-Type      Target-Position    Map-Rebinning
 Detector-ID        Chip-Mask          File-Specification
 XRT-PSF-Type       Build-ARF          Exit
 Quit

 CMD: Select  1 Option
```

```
CMD>
```

これでターゲットの位置を入力できます。"CMD>" に引き続いて "target" と入力するとプロンプトが POS> に変わります。この状態でターゲットの位置を "mm" 単位で以下のように入力します。

```
POS> x 2.01 y 1.01
```

ピクセルから mm への変換はセクション 3.1.1 に示した通りです。

2.2.2 FITS file

積分領域を指定する場合と同様に、ターゲットの位置もイメージ FITS ファイルから入力することができます。以下ではこのファイルを「ターゲット・マップ」と呼ぶことにします。ターゲット・マップの各ピクセルは 0 ~ 1 の間の値を持っていて、**jbldarf** はターゲット位置をこの重みで平均して ARF を作ります。この性質は

1. 点源の場合には、衛星の姿勢変動に伴う視野の揺れ
2. 広がったソースの場合には、表面輝度分布

を ARF の計算に反映させるのに使われます。
まず "CMD>" プロンプトに続いて position と入力して下さい。すると

```
CMD> position

 TYP:  ********** Position Type **********

 FITS-file     One-Point     FFT-FITS-file XRT-PSF

 TYP: Select  1 Option
TYP>
```

というメッセージが返ってきます。ターゲット・マップに指定できるものとしては 3 種類があります。この違いについてはセクション 3.2.2、および 3.3.2を参照して下さい。ここではとりあえず "FITS-file" を選びます。

```
TYP> FITS

 CMD:  ********** Command **********

 Show-Parameters    Region-Type        Integrated-Region
 Position-Type      Target-Position    Map-Rebinning
 Detector-ID        Chip-Mask          File-Specification
 XRT-PSF-Type       Build-ARF          Exit
 Quit

 CMD: Select  1 Option
CMD>
```

ここで target と入力すればターゲット・マップの名前を聞いてきますので、適当に入力します。

```
CMD> target
Source position filename: ./pos.fits ?
```

積分領域の場合と同様、SIS でターゲット・マップを読ませた場合にも計算時間が膨大になることがあります。このような場合にはターゲット・マップのピクセルを適当にビンまとめすることで時間を短縮することができます (セクション 3.6)。

2.3 Detector ID

CMD> プロンプトが出ている状態で detector と打つと detector の ID を変えることができます。選択肢は 0、 1、 2、 3 のいずれかで、それぞれ SIS-0、 SIS-1、 GIS-2、 GIS-3 に対応します。

2.4 File Specification

ARF を作る時に参照する RMF と出力の ARF のファイル名を指定します。 CMD> プロンプトが出ている状態で file と入力します。すると次のような答が返ってきます。

```
RMF filename: default ?
```

ここで何も指定せずにリターン・キーを押すと default が選ばれ、 **jbldarf.par** に書かれている RMF のうち、セクション 3.3 で指定した Detector ID に対応するものが自動的に読み込まれます[2]。これを変更したい時にはここに RMF のファイル名を書けば OK です。

続いて作られる ARF の出力先を

```
Output response filename: a.arf ?
```

というふうに聞いてきますので、適当に答えて下さい。

2.4.1 SIS の RMF

宇宙研では SIS の標準的な RMF は

```
/rukio/h1/sis/rsp1.1alpha/rsp/
```

に置かれています。ここに置かれている RMF の詳細については SIS チームの人に聞いて下さい。また自分で SIS の RMF を作るために **sis_bldrsp2** という tool が用意されています。 **sis_bldrsp2** で "Disable-XRT-data" を選択し、出力として "FITS-Format" を選べば RMF を作ることができます。 **sis_bldrsp2** は宇宙研では ◁U▷

```
/adsoft/sis_bldrsp2/latest/ulxbin/sis_bldrsp2   for ULTRIX,
/adsoft/sis_bldrsp2/latest/osfbin/sis_bldrsp2   for OSF/1,
```

に置かれています。詳しい使い方については README を読むか、 SIS チームの人に聞いて下さい。

2.4.2 GIS の RMF ◁U▷

GIS の 1024ch の最新の RMF は

```
/caldb/data/asca/gis/cpf/95mar06/gis2v4_0.rmf
/caldb/data/asca/gis/cpf/95mar06/gis3v4_0.rmf
```

に置かれています。

2.5 XRT の PSF ◁N▷

改良された Ray-tracing プログラムによる新しい XRT の PSF が公開されています[3]。 **jbldarf** ではユーザーが従来の PSF と新しい PSF を選択できるようになっています。 CMD> プロンプトに続いて xrtpsf と入力すると以下のようなメッセージが返ってきて、 PSF の選択ができます。

[2] SIS で複数のチップに跨る ARF を作る場合 (セクション 3.7)、番号の一番小さいチップに対応する RMF が読み込まれます。もしチップを選択していない場合には SIS-0、 SIS-1 に対してそれぞれチップ 1、チップ 3 の RMF が読み込まれます。

[3] XRT チームによる

```
CMD> xrt-psf
 PSF:  ********** XRT PSF Type **********

 Analytic Table

 PSF: Select  1 Option

PSF>
```

ここで Analytic というのが古い PSF、 Table が新しい PSF です。普通の解析には新しい方を使うため、 table と入力します。すると以下のように、有効面積のテーブルファイル名、 PSF のテーブルファイル名、 PSF を計算するエネルギーステップを順番に聞いています。

```
PSF> table
XRT effective area table file name: /caldb/data/asca/xrt/bcf/xrt_ea_110494.fits
XRT PSF table file name: /asca/caldb/xrtpsf95a.fits ?
Energy step in keV for XRT PSF calculation: 0.0 ?
```

有効面積のテーブルと PSF のテーブルは、普通の解析ではデフォルトのままでいいので空リターンします。エネルギーステップに 0 を指定すると、 **jbldarf** は RMF と同じエネルギーステップで PSF を計算します。計算速度を上げたい場合にはここで正の数を入れて下さい。 1.0 を入れれば PSF の計算は 1.0keV 刻みで行なわれます。

2.6 マップのビンまとめ (SIS のみ)

領域ファイルやターゲット・マップのピクセルをまとめてから ARF を計算させるには "CMD>" プロンプトにつづいて map と入力します。すると以下のようなメッセージが返ってきますので、まとめるピクセルの数を入力します。

```
CMD> map
 MAP:  -----<< Map Rebinnig Factor (for SIS only) >>-----

 RX-Region                     1
 RY-Region                     1
 PX-Position                   1
 PY-Position                   1

 MAP: Press <ENTER> to Accept

MAP>
```

例えば領域ファイルは X/Y とも 8 ピクセルごとにまとめ、ターゲット・マップの方は 16 ピクセルごとにまとめたい場合には

```
MAP> rx 8 ry 8 px 16 py 16 ok
```

とします。もちろん X、 Y 別々の大きさでまとめることも可能です。

このようなビンまとめの操作のために、結果の ARF が変わってくることがありますが、まとめる大きさをいろいろ変えてみて影響を調べて下さい。計算速度は当然早くなります。なお、この設定は SIS の ARF を作る時のみ有効で、 GIS の ARF を作る時 (Detector ID が 2 または 3 となっている時) には何が設定してあっても無視されます。

2.7 2/4CCD モードでの ARF

2CCD モード、あるいは 4CCD モードの観測で PI コラムを使って、スペクトルを足し上げる場合の ARF の作り方を説明します。次のような 2 通りの方法があります。

2.7.1 Chip Mask

Chip Mask 機能を使うと、 One-point またはイメージ FITS ファイルによって選ばれた積分領域と、チップとのイメージの積をとることで、特定の chip のみの ARF を作ることができます。 “CMD>” プロンプトに引き続いて chip と入力して下さい。

```
CMD> chip
 MSK:  -----<< Chip Mask (for SIS only) >>-----

 No-Mask 0       1       2       3

 MSK: Select  0 or More, Then OK

MSK>
```

2CCD デフォルト位置の観測で得られた PI チャンネルのスペクトルを扱うには SIS-0 ではチップ 1 とチップ 2 を合わせた ARF が必要になります。この場合には

```
MSK> 1 2 ok
```

と入力して ARF の計算に含めるチップ (この場合は 1 と 2) を指定します。もしチップ 1 だけの ARF が欲しければ

```
MSK> 1 ok
```

とすれば OK です。同様に 4CCD モード用の ARF は

```
MSK> 0 1 2 3 ok
```

で作ることができます。 Chip Mask ではチップ間の溝による実効的な有効面積の低下を正しく評価します。

これに対して “No-Mask” を選択すると **jbldarf-0.50** 以前と同様の動作をし、チップ間の溝があることを考慮しませんので、 XRT の有効面積が額面通りに取り込まれ、結果的にソースのフラックスが小さめに出てきます。

2.7.2 Chip Maps

キャリブレーション・データー・ベースには chip map と呼ばれるイメージが入っています。これは SIS の各チップが DETX/Y 上のどの位置を占めるかを表すマップで、各チップごとに、チップにかかっているピクセルには 1 が、チップから外れているピクセルには 0 が入ったマップになっています。この chip map と **makemap** によって作られる領域ファイルの積をとることで、目的とするチップのみの ARF を作ることができます。

イメージ FITS ファイル同士に四則演算を施す tool としては **farith** があります。使い方は

```
farith <input region file> <chip map> <resulting region file> mul
```

です。ここで “mul” はイメージの対応するピクセル同士の “and” をとることを意味しています。例えば領域ファイル circle.fits のうち、 S0C1 にかかっている部分のみをとりだして result.fits という領域ファイルを作りたい時は以下のようにします。

```
farith circle.fits s0c1.fits result.fits mul
```

念のため SAOIMAGE などで result.fits を見て、目的通りの領域ファイルになっているかどうか確認して下さい。また、 Chip Map を使う時には Chip Mask のステータスが “No-Mask” になっていることを確認して下さい。

2.8 2keV 付近の「こぶ」(SIS のみ) ◁N▷

「あすか」の応答関数には、 2keV 付近に原因不明の「こぶ」のような構造があることが知られています。SIS ARF を作成する時には、この構造を ARF 計算に採り入れるかどうかを選択することができます[4]。 “CMD>” プロンプトにつづいて kobu と入力すると以下のようなメッセージが返ってきます。

```
CMD> kobu
Multiply "kobu" factor by SIS ARF (Y/N): N ?
```

ここで「こぶ」を ARF に採り入れたいなら、

```
Multiply "kobu" factor by SIS ARF (Y/N): N ? y
```

と入力します。「こぶ」を採り入れたくなかったら “y” の代わりに “n” と入力します。

2.9 Building the ARF response

以上の全ての設定が終ったら “CMD>” プロンプトに続けて “build” と打てば計算が始まります。もしうまく ARF ができない時や、ソフトが原因不明の異常終了をするときは是非連絡を下さい。

We Need Feedback !!

[4]「こぶ」は RMF と ARF のどちらか一方にだけ入っているのが正しい状態です。 ARF に「こぶ」を採り入れた場合には「こぶ」の入っていない RMF を、 ARF に「こぶ」を採り入れなかった場合には「こぶ」の入っている RMF を、それぞれ組み合わせる必要があります。

第 3 章

jbldarf-2.10 を正しく使うために

本章は **jbldarf** をよりよく理解し、活用したい人のための解説です。

jbldarf は、いろいろな観測条件や解析方法について、対応するレスポンスを計算することができます。しかし、「あすか」のレスポンスはまだまだ開発途上であり、現状では不十分な点も多いと感じています (§3.4参照)。「あすか」の解析で **jbldarf** を使われる方には、ぜひとも、本章を一読なさって、**jbldarf** の特長や弱点を踏まえた上で使ってほしいと思います。

3.1 ARF ファイルとは

3.1.1 XSPEC とレスポンス

jbldarf がどのような計算を行なうかを見る前に、簡単にレスポンスについておさらいしておこう。スペクトルフィットを行なう場合、XSPEC はモデル関数 $\mathrm{M}[E]$、RMF レスポンス $\mathrm{R}[PHA, E]$、および ARF レスポンス $\mathrm{A}[E]$ から検出器での検出が予想される PHA スペクトル $\mathrm{S}[PHA]$ を次のように計算する[1]。

$$\mathrm{S}[PHA] = \sum_{E} \mathrm{R}[PHA, E]\ \mathrm{A}[E]\ \mathrm{M}[E]$$

XSPEC は、この $\mathrm{S}[PHA]$ と観測された PHA スペクトルが合うようにモデル関数のパラメータを調整する[2]。

レスポンスをこのように 2 つにわける利点は、レスポンスのうち観測や解析の方法に依存しない部分や、センサ間で共通な部分を共有することができる点にある[3]。こうすることで、レスポンスの計算にかかる時間を短縮でき、さらにディスク領域も節約することができる。

余談になるが、上の式では、いわゆる検出器の応答関数に相当するものは $\mathrm{R}[PHA, E]\ \mathrm{A}[E]$ という積の形で表現されているため、RMF レスポンスと ARF レスポンスの定義の方法は一意的でない[4]。実際には、SIS の場合、$\sum_{PHA} \mathrm{R}[PHA, E]$ はエネルギー E での検出効率に相当し、ARF は有効面積の部分を受け持っている。GIS の場合には、$\sum_{PHA} \mathrm{R}[PHA, E]$ は 1 に規格化されており[5]、ARF が有効面積と検出効率の部分を受けもつようになっている。

3.1.2 ARF ファイルの構造

レスポンスが思った通りに作成できたかどうかを調べるには、ARF ファイルをのぞいてみるのが一番である。この節では、そのために必要な知識をまとめておこう。

---

[1] そのはずである。確かめたわけではない。

[2] 実際には、バックグラウンドの差し引きなども行なわれる。

[3] 実際、現状では GIS の RMF レスポンスは、GIS2 と GIS3 で同じものを使用している。

[4] 例えば、RMF レスポンスを 2 倍にするのと同時に ARF レスポンスを 2 分の 1 にすれば、結果として同じ応答関数を得る。

[5] スペクトルが RMF のエネルギーレンジからはみ出す場合もあるので、常に 1 になるわけではない。

ARF ファイルは FITS 形式のファイルで最低 1 つの BINTABLE extension を持つ。 fstruct[6]でみてみると、例えば次のようになっているはずである。

```
No. Type     EXTNAME      BITPIX Dimensions(columns)      PCOUNT  GCOUNT
 0  PRIMARY                 8     0                          0     1
 1  BINTABLE SPECRESP       8     20(5) 201                  0     1
```

EXTNAME が 'SPECRESP' となっているところが ARF の本体である[7]。 jbldarf の場合は、 1st extension に BINTABLE を作るようになっている。

さて、その BINTABLE には、最低でも次の 3 つの項目があるはずである。

ENERG_LO, ENERG_HI, SPECRESP

ENERG_LO、 ENERG_HI はそのビンのエネルギー範囲を決め、 SPECRESP がそのエネルギー範囲での $A[E]$ を与える。 jbldarf はその他に、 **EFFAREA, EXPOSURE** という項目も ARF ファイルに書き出すようになっている。 EFFAREA には目安としての有効面積が、 EXPOSURE には SPECRESP を EFFAREA で割ったものが入っている。いずれも、 XSPEC では参照されないのでスペクトルフィットには無関係だが、つくられた ARF のだいたいの様子を知りたい時には便利である。 fplot[8]で一度それらの項目をプロットしてみるといいだろう。

3.1.3 ヘッダのキーワード ◁U▷

jbldarf は、 ARF ファイル作成時のさまざまな情報を ARF ファイルのヘッダに書き出すようになっている。レスポンスが何か変だと思った時は、ヘッダを注意深く見てみれば何が悪かったのかわかることが多い。

以下に jbldarf が ARF に書き出すヘッダの主なものの一覧とその内容を示す。なお、以下では、座標軸毎に書き出されるものは ???X(Y) などとして一括して表記してある。また、右肩に 2.00 とついているキーワードは、 jbldarf-2.00 から新しく導入されたものである。

プライマリヘッダ ◁U▷

- DATE　この ARF ファイルを作った日 / 月 / 年
- CREATOR$^{2.00}$　この ARF ファイルを作った人のログイン名[9] ◁U▷
- AUTHOR　jbldarf のバージョン
- ORIGIN　jbldarf working group

SIS/GIS 共通のヘッダ ◁U▷

- NAXIS2　エネルギーのビン数。参照した RMF と同じ数になる。
- TELESCOP　通常 'ASCA'。参照した RMF からコピーする。
- INSTRUME　通常 'SIS0' 'SIS1' 'GIS2' 'GIS3' のいずれか。参照した RMF からコピーする。
- DET_ID　ARF を作った時の Detector-ID。 0 から 3 の値をとる。
- TELDEF　検出器のアライメント情報を引っ張ってきた teldef ファイル[10] の名前。
- XRT_EA　XRT の有効面積を引っ張ってきたテーブルのファイル名。
- PSFTYPE $^{2.00}$　XRT の Point Spread Function の計算方法。'Analytic'、'Table' のいずれか。 ◁N▷
- PSFTABLE $^{2.00}$　XRT の Point Spread Function を引っ張ってきたテーブルのファイル名。 PSF-type が 'Table' の時のみ書かれる。 ◁N▷

[6]FTOOLS のコマンドのひとつで、 FITS ファイルの構造を調べるのに役立つ。

[7]jbldarf-0.5 より前のものでは 'ARF' となっている。

[8]FTOOLS のコマンドのひとつ。 FITS のテーブルをグラフにできる。

[9]jbldarf-2.00 より前は CREATER だったので注意。

[10]*Telescope definition* の略。もともと ASCALIN 用のキャリブレーションファイル。 jbldarf では、 XRT 座標、検出器ミリメートル座標、検出器チャンネル座標の相互変換のために使用する。

PSFESTEP [2.00] XRT の Point Spread Function を計算するエネルギーの刻み。単位は keV。 PSF-type が 'Table' の時のみ書かれる。 ◁N▷

FC_X(Y)_SCL 検出器上での X(Y) 方向角度スケール (arcmin/mm)。 teldef ファイルからコピーされる。X/Y で値が違っていた場合の計算結果は保証できない。

OPTAXISX(Y) 検出器ミリメートル座標での光軸の X(Y) 位置。 teldef ファイルにも同様なキーワードがあるが、衛星座標系で書かれているので注意。

DET_X(Y)CEN 検出器チャンネル座標の X(Y) 方向の中心 (ミリメートル座標が 0 となる位置)。 teldef ファイルからコピーされる。

DET_X(Y)SCL 検出器上での X(Y) 方向ピクセルサイズ (mm/pixel)。 teldef ファイルからコピーされる。X/Y で値が違っていた場合の計算結果は保証できない。

DETX(Y)PIX1 検出器チャンネル座標系での X(Y) 方向の最小ピクセル番号。 1 で固定。

REGTYPE ARF 作成時の Region-type。'Circle'、'FITS-file' のいずれか。

REGCENTX(Y) リージョン中心の X(Y) 座標 (mm)。 Region-type が 'Circle' の時のみ書かれる。

REGRIN リージョンの内半径 (mm)。 Region-type が 'Circle' の時のみ書かれる。

REGROUT リージョンの外半径 (mm)。 Region-type が 'Circle' の時のみ書かれる。

REGFILE Region-file の値。 Region-type が 'FITS-file' の時のみ書かれる。

POSTYPE ARF 作成時の Position-type。'One-Point'、'FITS-file'、'FFT-FITS-file'、'XRT-PSF' のいずれか。

POSCENTX(Y) 検出器ミリメートル座標での X(Y) 方向ソース位置。 Position-type が 'One-Point' の時のみ書かれる。

POSFILE Position-file の値。 Position-type が 'FITS-file' か 'FFT-FITS-file' か 'XRT-PSF' の時のみ書かれる。

XRT_OFF ソース位置の XRT の光軸からの離角 (arcmin)。

XRT_AZM ソース位置の XRT の光軸からの方位角 (degree)。衛星座標上で定義されているので、検出器座標では X 軸の正方向から時計回りに測られるので注意。

SIS 専用のヘッダ ◁U▷

REGRBINX(Y) リージョンマップの X(Y) 方向のビンまとめ数。

POSRBINX(Y) ポジションマップの X(Y) 方向のビンまとめ数。 Position-type が 'FITS-file' の時のみ書かれる。

CHIPMASK リージョンマップのマスクに用いたチップの CCD ID (例:`'0 1 2 3'`)。マスクしなかった時は`'No Mask'` となる。

SIS_KOBU [2.10] 2keV 付近の「こぶ」を ARF 計算に採り入れたかどうか。採り入れたら'Multiplied'、採り入れなかったら'NOT multiplied'。

GIS 専用のヘッダ

REGPIXLS $\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)$ の値 (§3.3.1)。要するに、リージョンの面積 (pixels)。

INTG_X(Y)0 計算を行なった領域の X(Y) 方向の最小ピクセル番号。ただし、ピクセル番号は 0 から始まっている。

INTG_NX(Y) 計算を行なった領域の X(Y) 方向の大きさ (pixel)。

POSRSIZE ポジションマップの広がり (pixel)。 Position-type が 'FITS-file' か 'FFT-FITS-file' か 'XRT-PSF' の時のみ書かれる。

GRIDFILE　井桁マップのファイル名。

GRIDTRNS　リージョンでの井桁マップの平均透過率 (%)。

BE_FILE　Be 膜厚マップのファイル名。

BE_THICK　リージョンでの Be 膜厚の平均値 (μm)。

XE_THICK　ドリフト領域の Xe の厚さ (cm)。

RSPVERSN　GIS のレスポンス関数のバージョン。

3.2 何を計算するのか (SIS 編)

3.2.1 計算式 ◁U▷

jbldarf が計算していることは、次のような式で書くことができる[11]。

$$\mathrm{A}[E] = F_{\mathrm{kobu}}(E)T_{\mathrm{xrt}}(E)\int d\vec{r}_1 w_{\mathrm{P}}(\vec{r}_1)g_{\mathrm{xrt}}(E;\vec{r}_1)\int d\vec{r}_3 w_{\mathrm{R}}(\vec{r}_3)P_{\mathrm{xrt}}(E;\vec{r}_3,\vec{r}_1)$$

各記号の意味は、次のようになっている。

$P_{\mathrm{xrt}}(E;\vec{r}_3,\vec{r}_1)$　XRT の *Point Spread Function,* すなわち、 $\vec{r}_1$ 方向から X 線が来た時に、検出器上で $\vec{r}_3$ の位置に反射される確率。

$g_{\mathrm{xrt}}(E;\vec{r}_1)$　$\vec{r}_1$ 方向からエネルギー E の X 線が来た時の XRT の *Effective Area*。

$w_{\mathrm{P}}(\vec{r}_1)$　*Position Map,* すなわち、 $\vec{r}_1$ 方向にソースが存在する確率。 $\vec{r}_1$ で積分した値が 1 になるように規格化されている。

$w_{\mathrm{R}}(\vec{r}_3)$　*Region Map,* すなわち、 $\vec{r}_3$ の位置に検出されたイベントを採用する確率。

$T_{\mathrm{xrt}}(E)$　XRT に施された Thermal Shield の X 線透過率。

$F_{\mathrm{kobu}}(E)$　2keV 付近の「こぶ」を表した因子。中心約 2keV、幅約 0.1keV のガウシアンに 1.0 を加えたもの。

式を見てわかるように、 $g_{\mathrm{xrt}}(E;\vec{r}_1)$ と $P_{\mathrm{xrt}}(E;\vec{r}_3,\vec{r}_1)$ は特に分ける必要はないのだが、プログラムの便宜上、分けられている。また、 $\vec{r}_1,\vec{r}_3$ は、 X 線が来る方向[12]、実際に検出された位置 (SIS の場合、検出器面上で X 線が来た位置に一致する[13]) にそれぞれ対応する。積分領域は、計算式の上では全領域に対してということになるが、実際には、 $w_{\mathrm{R}}(\vec{r}_3)$ が 0 でない領域の近辺でのみ積分を行なっている。

3.2.2 より具体的には ◁U▷

もっと具体的に計算の流れを書くと、以下のようになる。

(1) ポジションマップを準備する。

  (a) Position-type が **'One-Point'** のとき
  $\vec{r}_1$ = (Source-X-pos, Source-Y-pos) をセットする。

  (b) Position-type が **'FITS-file'** のとき
  Position-file を読み込み、指定されたビンまとめを行ない、さらに、足して 1 になるようあらかじめ規格化する。格子点の数が多いと、後の計算で非常に時間がかかって使用に耐えないので、解析の主旨や XRT の性能などを十分に考慮してビンまとめ数を指定する必要がある (3.4.4参照)。

[11] ただし、 $F_{\mathrm{kobu}}(E)$ は、 ARF 計算に「こぶ」を採り入れる時のみ乗ずる。

[12] 天球の絶対的な方向ではなく、あくまで XRT に対する相対的な方向であることに注意。

[13] このことは、検出器の *Point Spread Function* がデルタ関数的であることに対応している。

(2) リージョンマップを準備する。

まず、以下のように、もとになるマップを作る。

(a) Region-type が **'Circle'** のとき
リージョンの中心や半径などに従ってマップを作り、メモリ上に保持する。

(b) Position-type が **'FITS-file'** のとき
Region-file を読み込み、メモリ上に保持する。

次に、指定されたチップの位置にマスクをかける。すなわち、指定されたチップの上にない $\vec{r}_3$ に対して $w_{\rm R}(\vec{r}_3)=0$ をセットする。[14]チップマスクが **'No-Mask'** のときには何もしない。[15]

最後に、指定されたビンまとめを行なう。ここでも、格子点の数が多いと、後の計算で非常に時間がかかって使用に耐えないので、解析の主旨や XRT の性能などを十分に考慮してビンまとめ数を指定する必要がある (3.4.4参照)。

(3) ソースポジション $\vec{r}_1$ に対する $g_{\rm xrt}(E;\vec{r}_1)$ を計算する。
$\vec{r}_1$ に対する $g_{\rm xrt}(E;\vec{r}_1)$ をすべてのエネルギーについてまとめて計算し、メモリ上に保持する。これ以降の計算ではメモリ上の値を参照する。このような操作を行なうのは計算を高速化するためである。

(4) $w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)P_{\rm xrt}(E;\vec{r}_3,\vec{r}_1)$ を計算し、 A$[E]$ に加算する。
積分計算の部分はループを回して足し算をするだけである。ただし、この足し算は、すべてのエネルギーに対して並列に行なう。このとき、前のステップで計算しておいた $g_{\rm xrt}(E;\vec{r}_1)$ をメモリから読み出して用いる。 A$[E]$ への加算は $\vec{r}_1$ 積分の 1 ステップに対応する。

なお、計算速度の向上のため、 PSF の計算は、以下のように、特定のエネルギーでのみ行なっている。 ◁N▷

(a) PSF-type が 'Analytic' のとき
供給されている XRT の PSF がエネルギーに依存しない形に書かれていることを利用し、エネルギーごとの PSF の計算を省略する。つまり、すべてのエネルギーに対して、始めに一度だけ PSF が計算される。

(b) PSF-type が 'Table' のとき
ユーザの指定に応じたエネルギー刻みで、 PSF を計算する。例えば、エネルギー刻みとして 1keV を指定した場合、 0.5keV、 1.5keV、 2.5keV、 $\cdots$ で PSF を計算し、それぞれ、 0〜1keV、 1〜2keV、 2〜3keV、 $\cdots$ における ARF の計算に用いる。

(5) $\vec{r}_1$ を変えて、 (3) へ戻る。

(6) すべての $\vec{r}_1$ について計算し終ったら、 $T_{\rm xrt}(E)$ を掛ける。
加算された A$[E]$ には、 $\int d\vec{r}_1 w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)P_{\rm xrt}(E;\vec{r}_3,\vec{r}_1)$ の計算結果が入っている。
そこで、これにエネルギー毎に $T_{\rm xrt}(E)$ を掛ける。

(7) 「こぶ」を採り入れるよう指定された時には、 $F_{\rm kobu}(E)$ を掛ける。
この時点で A$[E]$ には、 $T_{\rm xrt}(E)\int d\vec{r}_1 w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)\int d\vec{r}_3 w_{\rm R}(\vec{r}_3)P_{\rm xrt}(E;\vec{r}_3,\vec{r}_1)$ の計算結果が入っている。そこで、これにエネルギー毎に $F_{\rm kobu}(E)$ を掛け、最終的な A$[E]$ を得る。

[14] この操作によって、視野外や溝の部分は常に $w_{\rm R}(\vec{r}_3)=0$ となり、 ARF の計算から除外される。

[15] したがって、この場合、視野の外や溝の部分も (リージョンマップに含まれていれば) ARF に加算される。

3.2.3 DEBUG パラメータ

jbldarf の DEBUG パラメータを ON にすると、計算の途中結果のイメージをカレントディレクトリの ./map/ というディレクトリに書き出すようになっている[16]。書き出されるイメージには、次のようなものがある。

original_posmap.fits	ビニング前の *Position Map*
binned_posmap.fits	ビンまとめされた *Position Map*
original_region.fits	チップマスク前の *Region Map*
masked_region.fits	チップマスクされた *Region Map*
binned_region.fits	ビンまとめされた *Region Map*

実際の計算に用いられた $w_{\rm R}(\vec{r}_3)$ と $w_{\rm P}(\vec{r}_1)$ は、それぞれ、 binned_regmap.fits、 binned_posmap.fits である。これらを saoimage などで見れば、ビンまとめが適切であったかどうかを確認することができる。

3.3 何を計算するのか (GIS 編)

3.3.1 計算式

jbldarf が計算していることは、次のような式で書くことができる。

$$\mathrm{A}[E] = \int d\vec{r}_3 w_{\rm R}(\vec{r}_3) \int d\vec{r}_2 g_{\rm gis}(E;\vec{r}_2) P_{\rm gis}(E;\vec{r}_3,\vec{r}_2) \int d\vec{r}_1 w_{\rm P}(\vec{r}_1) g_{\rm xrt}(E;\vec{r}_1) P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$$

各記号の意味は、次のようになっている。

- $P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$　XRT の *Point Spread Function,* すなわち、 $\vec{r}_1$ 方向から X 線が来た時に、検出器上で $\vec{r}_2$ の位置に反射される確率。
- $g_{\rm xrt}(E;\vec{r}_1)$　$\vec{r}_1$ 方向からエネルギー E の X 線が来た時の XRT の *Effective Area*。
- $w_{\rm P}(\vec{r}_1)$　*Position Map,* すなわち、 $\vec{r}_1$ 方向にソースが存在する確率。 $\vec{r}_1$ で積分した値が 1 になるように規格化されている。
- $P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$　GIS の *Point Spread Function,* すなわち、検出器上で $\vec{r}_2$ の位置に X 線が来た時に、その位置が $\vec{r}_3$ として検出される確率。
- $g_{\rm gis}(E;\vec{r}_2)$　検出器上で $\vec{r}_2$ の位置に X 線が来た時の GIS の検出効率。
- $w_{\rm R}(\vec{r}_3)$　*Region Map,* すなわち、 $\vec{r}_3$ の位置に検出されたイベントを採用する確率。

式を見てわかるように、 $g_{\rm xrt}(E;\vec{r}_1)$ と $P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$、あるいは $g_{\rm gis}(E;\vec{r}_2)$ と $P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$ は特に分ける必要はないのだが、プログラムの便宜上、分けられている。また、 $\vec{r}_1,\vec{r}_2,\vec{r}_3$ は、 X 線が来る方向[17]、検出器面上で X 線が来た位置、実際に検出された位置、にそれぞれ対応する。積分領域は、計算式の上では全領域に対してということになるが、実際には、 $w_{\rm R}(\vec{r}_3)$ が 0 でない領域の近辺でのみ積分を行なっている。

3.3.2 より具体的には

◁U ▷

もっと具体的に計算の流れを書くと、以下のようになる。

(1) $psf(E;\vec{r}_2) = \int d\vec{r}_1 w_{\rm P}(\vec{r}_1) g_{\rm xrt}(E;\vec{r}_1) P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ を計算する。

(a) Position-type が **'One-Point'** のとき
$\vec{r}_1$ = (Source-X-pos, Source-Y-pos) での $g_{\rm xrt}(E;\vec{r}_1) P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ のみを計算する。

---

[16] ./map/ ディレクトリがなければ作らない。

[17] 天球の絶対的な方向ではなく、あくまで XRT に対する相対的な方向であることに注意。

(b) Position-type が **'FITS-file'** のとき
$w_{\rm P}(\vec{r}_1)$ が 0 でない格子点で $g_{\rm xrt}(E;\vec{r}_1)P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ を計算し、 $w_{\rm P}(\vec{r}_1)$ の重みをつけて足し合わせる。 $w_{\rm P}(\vec{r}_1)$ は Position-file から読み込まれ、足して 1 になるようあらかじめ規格化されている。格子点の数が多い時は、非常に遅くなって、使用に耐えない。

(c) Position-type が **'FFT-FITS-file'** のとき[18]
$w_{\rm P}(\vec{r}_1)$ の重心で $P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ を計算し、 $w_{\rm P}(\vec{r}_1)g_{\rm xrt}(E;\vec{r}_1)$ との convolution をとる。 convolution には FFT を用いるので、上の場合より一般に速くなる。

(d) Position-type が **'XRT-PSF'** のとき[19]
FITS ファイルから $psf(E;\vec{r}_2)$ を読み込む。詳しくは、§3.5.3 を参照のこと。

なお、計算速度の向上のため、 PSF の計算は以下に示す特定のエネルギーでのみ行なわれる。 ◁N▷

(i) PSF-type が 'Analytic' のとき
供給されている XRT の PSF がエネルギーに依存しない形に書かれていることを利用し、エネルギーごとの PSF の計算を省略する。つまり、すべてのエネルギーに対して、始めに一度だけ PSF が計算される。

(ii) PSF-type が 'Table' のとき
ユーザの指定に応じたエネルギー刻みで、 PSF を計算する。例えば、エネルギー刻みとして 1keV を指定した場合、 0.5keV、 1.5keV、 2.5keV、 $\cdots$ で PSF を計算し、それぞれ、 0〜1keV、 1〜2keV、 2〜3keV、 $\cdots$ における ARF の計算に用いる。

(2) $e\!f\!f(E;\vec{r}_2) = g_{\rm gis}(E;\vec{r}_2)psf(E;\vec{r}_2)$ を計算する。
$g_{\rm gis}(E;\vec{r}_2)$ は、 Be 膜厚マップと井桁マップから計算される。 $\vec{r}_2$ での Be 膜厚マップの値から、そこでの GIS の検出効率[20]を計算し、それに井桁マップの値と、 (1) で計算した $psf(E;\vec{r}_2)$ が掛け算されて、 $e\!f\!f(E;\vec{r}_2)$ ができ上がる。

(3) $expo(E;\vec{r}_3) = \int d\vec{r}_2 P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)e\!f\!f(E;\vec{r}_2)$ を計算する。
通常使用される範囲では、 GIS の位置分解能はあまり変化しない。そこで、ソース位置での $P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$ と $e\!f\!f(E;\vec{r}_2)$ の FFT を用いた convolution を行なうことで、計算時間を短縮している。

(4) $\mathrm{A}[E] = \int d\vec{r}_3 w_{\rm R}(\vec{r}_3)expo(E;\vec{r}_3)$ を計算する。
これはもう、ループを回して足し算をするだけである。

(5) E を変えて、 (1) へ戻る。

3.3.3 DEBUG パラメータ

jbldarf の DEBUG パラメータを ON にすると、計算の途中結果のイメージをカレントディレクトリの ./map/ というディレクトリに書き出すようになっている[21]。書き出されるイメージには、次のようなものがある。

[18] GIS のみ有効。
[19] これも、 GIS のみ有効。
[20] ここでは Be 膜の透過率だけでなく、 Xe ガスの吸収係数、サーマル / プラズマシールドの透過率も計算される。
[21] ./map/ ディレクトリがなければ作らない。

bemap.fits	Be 膜厚マップ
effnnn.fits	$\mathit{eff}(E;\vec{r}_2)$
exponnn.fits	$\mathit{expo}(E;\vec{r}_3)$
gispsfnnn.fits	$P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$
grid.fits	井桁マップ
posmap.fits	*Position Map*
region.fits	*Region Map*
xrtpsf.fits	$\mathit{psf}(E;\vec{r}_2)$

nnn がつくものは、エネルギーの刻みごとにつくられる。 xrtpsf.fits が 1 つしか作られないのは、現状の XRT レスポンスでは、エネルギーによって *Point Spread Function* が変化しないからである[22]。また、これらのイメージは、実際に計算が行なわれる範囲に限って書き出される。書き出された範囲は ARF のヘッダに書いてある。

3.4 適用限界

jbldarf の適用限界は、大まかに分けて、提供されているレスポンスの信頼性と、jbldarf での ARF の計算方式とに起因すると言える。さらに、提供されているレスポンスは XRT 部 と GIS 部に分けられ[23]、$g_{\rm xrt}(E;\vec{r}_1)$ と $P_{\rm xrt}(E;\vec{r}_2,\vec{r}_1)$ が XRT 部、$g_{\rm gis}(E;\vec{r}_2)$ と $P_{\rm gis}(E;\vec{r}_3,\vec{r}_2)$ が GIS 部に属する。以下では、これらの関数について概説し、 ARF を作る上での問題点を述べる。

3.4.1 XRT のレスポンスの観点から ◁U▷

jbldarf が使用している XRT のレスポンスは、有効面積を与えるものと、 PSF を与えるものの 2 種類である。いずれも、 FORTRAN の関数で、有効面積を与えるものは、

```
Real Function xrt_eff_area(e_min, e_max, theta, phi)
Real e_min, e_ma, theta, phi
```

PSF を与えるものは、

(a) PSF-type が 'Analytic' のとき

```
Real Function xrtpsf(E,x1,y1,x2,y2)
Real E, x1, y1, x2, y2
```

(b) PSF-type が 'Table' のとき ◁N▷

```
Real Function xrtpsf95a(E,theta,phi,ox,oy)
Real*8 E, theta, phi, ox, oy
```

という形になっている。

現在の jbldarf で使用しているものでは `xrt_eff_area()` はパラメータファイルで指定された FITS の有効面積テーブル (XRT-EA-table) から読み込んだものを適当に補間して返すようになっていて、信頼性はそのテーブルファイルによる。関数形を見てわかるように、センサー ID を渡すことはできず、センサーごとの個性は考慮していない。 XRT チームによると、現状の XRT-EA-table には Ray-tracing によるデータを元に、次の範囲のデータが入っている。

名前	単位	形式	範囲 (刻み)
Energy	keV	1E	0.1–2.0 (0.1), 2.01–4.00 (0.01), 4.1–12.0 (0.1)
Theta	arcmin	1I	0–20 (1)
Phi	degree	1I	0–45 (15)
EA	$\rm cm^2$	1E	—

[22] Position-type に 'XRT-PSF' を選んだ時はその限りではない。

[23] SIS のレスポンスはすべて RMF に吸収されているため、 ARF の計算には使われない。

当然、この範囲を越えた指定をした場合のレスポンスは保証されない。

PSF-type が 'Analytic' のときに用いられる `xrtpsf()` は、 XRT の PSF を内部で適当な算術関数で近似して計算を行なっている。これも、 FORTRAN の関数形を見てわかるように、センサーごとの個性は考慮していない。また、現在使用しているものでは「ちょうちょ」とか「よつばのクローバー」とかよばれる形は入っておらず[24]、端にいくと PSF が歪む効果も入っていないようである。さらに、引数にはエネルギーが入っているが、関数の中では全く使われておらず、エネルギーの依存性は有効面積の方に任せた形になっている。ついでにいっておけば、ピークもちょっと鋭すぎて、現実と合わないようだ。以上のことから、リージョンをソース位置を中心とした円以外の形で切ったり[25]、リージョンの半径を大きくとりすぎたり小さくとり過ぎたり[26]した場合の ARF は信頼度が低くなる。

PSF-type が 'Table' のときに用いられる `xrtpsf95a()` は、 Ray-tracing を用いて計算した PSF のテーブルファイルを読み込み、一次補間を行なって ARF 計算に必要な位置での PSF を求めている。これもセンサーごとの個性は考慮していないが、現時点での最新の Ray-tracing の結果が反映されているため、 PSF のエネルギー依存性や、いわゆる「つの」の効果も採り入れられている。ただし、テーブルは、 EEF(Encircled Energy Function) の形で供給されているので、ソース位置を中心とした円形領域以外でスペクトルを切りとったりした場合、 ARF は信頼度が低くなる。 ◁N▷

まとめると XRT のレスポンスの観点から気をつけるべきことは、次のようなことである。

(1) エネルギーは 0.1 – 12 keV、ソース位置は光軸から 20 arcmin 以内。
(2) リージョンはソース位置を中心とした円または円環で切る。
(3) リージョンの半径は 1.5 – 6 arcmin 程度。

ただし、これらの値は、今後 XRT の有効面積テーブルや PSF テーブルが改訂されれば、それにともなって改善する可能性がある。両テーブルの適用限界については XRT チームから発表されるものを参照すること。 ◁N▷

3.4.2 GIS のレスポンスの観点から

jbldarf が使用している GIS のレスポンスは C の関数で、

```
double get_gis_eff(double e_lo, double e_hi);
double gis_psf(double xmm, double ymm, double Ekev, double xoff, double yoff);
```

があり、それぞれ GIS の検出効率と PSF を与えるものである。関数形からわかるように、センサーごとの違いは考慮されていない。また、井桁マップや Be 膜厚マップもレスポンスを決める重要な要素である。

`get_gis_eff()` では、 XRT のサーマルシールド、プラズマ防御膜、 Be 膜、サポートメッシュ[27]の透過率と、 Xe+He ガスの吸収効率を計算している。エネルギーの範囲は 0.5–15keV くらいまでを想定している。位置による違いは入っていないが、特に問題ないと思われる。

`gis_psf()` は、実験のデータと、 XRT の入射角のばらつきによる広がりを考慮して、中心対称な 2 次元ガウシアンの重ね合わせとして PSF を計算している。中心から 20mm より外では位置分解能が悪くなる効果も入っているが、 3.3.2で述べたように、 jbldarf の計算ではソース位置での PSF しか使っていないので、あまり意味はない。したがって、センサーの周辺付近ではレスポンスの信頼度が落ちる。また、 GIS の位置分解能はエネルギーによって違うため、エネルギーによってリージョンの外に洩れ出すフォトンの数は違ってくることになる。当然、リージョンが小さい方がこの効果が大きくなるが、これがどこまで正確に再現できているかについては、はっきりとは調べられていない。したがって、リージョンをあまり小さくとると、 ARF の信頼度は落ちる。

Be 膜厚マップは、平均的な厚さは地上実験を元にしたデータが入っているが、位置によるばらつきに関してはあまり精度は良くない。しかし、位置によるばらつきは、最大で 0.6μm(GIS-2) / 0.4μm(GIS-3) 程度 (上田修論、 1993) であり、それほど大きな影響はないと考えられる。

[24] 試しに適当な位置での PSF のイメージを作らせてみるとわかる。そのために、./src/gis/xrtpsf.c というプログラムが用意されている。デフォルトではコンパイルされない。

[25] 「ちょうちょ」や PSF の歪みが効いてくると思われる。

[26] 中心付近や tail 付近では、 PSF が正しく再現されていない。

[27] 0.87 でエネルギーによらず固定となっている。

井桁マップは、位置や透過率がエネルギーによらないことになっているが、これが本当かどうかを調べるのは統計的に苦しく、よくはわかっていない。また、井桁の位置は jbldarf の仕様上も、現実的にも、 1pixel の精度となっている (松下修論、 1994)。ソースが井桁のすぐそばにあった時の影響についてはまだ良くは調べられておらず、jbldarf がそれをちゃんと再現できるかどうかは怪しい。従って、井桁付近にソースがある時の ARF も信頼度は低い。

まとめると GIS のレスポンスの観点から気をつけるべきことは、次のようなことである。

(1) エネルギーは 0.5 – 15 keV。
(2) センサーの周辺部 ($\sim$ 20mm より外) は避ける。
(3) リージョンの半径は $\sim$ 1.5 arcmin より大きく。
(4) 井桁のそばのソースは避ける。

3.4.3 jbldarf の構造に起因する限界

現状の jbldarf では、以下のような場合に対応する ARF を作るのは困難である。

(1) 視野外にまでソースが広がっている場合。

Position-file は視野内しかカバーしていないため、視野外からの洩れ込みの影響は考慮されない。 Position-type を'XRT-PSF' にして、あらかじめその効果を計算しておけば、取り込むことは可能である。

(2) 位置によりスペクトルが違っている場合。

そもそも XSPEC 自体が、位置ごとにスペクトルが違う時のフィットをできるようになっていない[28]。このような解析をする時は、スペクトルが同じだと思える範囲でリージョンを切り分けたり、エネルギーバンドごとにイメージを作ったりして攻めるのが適当であろう。

3.4.4 SIS のマップのビンまとめ

SIS の ARF の計算式 (3.2.1参照) に現れる積分は、実際の計算では、 SIS の 1 ピクセルを単位として数値積分される (ビンまとめしない場合)。ところが、計算速度を測定してみると、通常の解析においては、 SIS の ARF の計算にかかる時間はほとんどピクセル数のみが支配していることがわかる。例えば、円形のリージョンマップを使っている場合、半径を 2 倍にすると計算にかかる時間は 4 倍になる。広がったソースの場合には、有限の広がりを持つポジションマップを使わざるを得ないため、ポジションマップの非零のピクセル数が N 個の場合には、計算時間はポイントソースの場合の N 倍かかってしまい、実用に耐えない。

そこで jbldarf では、計算速度向上のため、リージョンマップとポジションマップのそれぞれについて、指定された数だけビンまとめしてから計算するようにしている。この場合に気をつけなければならないのが、ビンまとめ数をいくつに設定すべきか、という点である。

まず、リージョンマップのビンまとめは、 XRT の PSF を考慮して決めなければならない。計算式 (3.2.1参照) から明らかなように、リージョンマップの刻みは、 ARF の数値積分における PSF の値のサンプリングに対応する。したがって、リージョンマップをあまり大きくビンまとめしてしまうと、 PSF の数値積分に支障をきたす。具体的には、 XRT の PSF の空間変化のスケールは、大雑把に言って分角程度なので、 10 ビン程度 (およそ 1/4 分角相当) にまとめることは問題ないであろうと思われる。

また、ポジションマップについては、天球上での輝度分布のスケールを考慮して決めなければならない。jbldarf はポジションマップで指定した通りの輝度分布について ARF を計算するので、例えば、輝度分布の起伏の空間スケールが 1 分角程度であるとわかっているものに対して、ポジションマップを 10 ビン (およそ 1/4 分角相当) 以上にまとめると、有効面積を不当に評価するおそれがある。

以上をまとめると以下のようになる。

(1) リージョンマップのビンまとめは、ビンまとめ後の 1 ビンの大きさが XRT の PSF の空間スケールよりも十分に小さくなるようにする。

[28] こうなると、スペクトル解析だけでなく、イメージ解析もからんでくるが…

(2) ポジションマップのビンまとめは、ビンまとめ後の 1 ビンの大きさがソースの輝度分布の空間スケールよりも十分に小さくなるようにする。

ただし、いずれの場合も、ビンまとめ数の決定は解析の主旨と密接に関わる問題である。解析者は、自分が行なっている解析の目的や必要な計算精度を十分に考慮し、その目的や精度にあったビンまとめ数を選ぶ必要がある。

3.5 応用編

ちょっと高度な使い方になるが、工夫すれば以下のような場合にも、 jbldarf を適用することは原理的に可能である。しかし、§3.4 で述べた制限に引っかかる場合が多く、意味のあるものができるかどうかは、解析者がよく考えておく必要がある。

3.5.1 衛星の姿勢の揺れの補正

衛星の姿勢の揺れを補正した ARF を作る場合、その ARF を使うスペクトルが、検出器座標 (DET_X/Y) で集積したものか、天空座標 (X/Y) で集積したものかで厳密にはやり方が異なる。ここでは、それぞれについて考察してみる。

ただし、通常、衛星の揺れはせいぜい 1arcmin 程度なので、よほどリージョンを狭くとったり、 GIS の井桁のそばのソースを扱ったりしない限り、姿勢の揺れがそれほど影響するとは思えない。また、 $\sim 1arcmin$ 以上に姿勢が外れている部分のデータは、使わないようにするのが賢明であろう。さらに、現状では姿勢ファイルの精度が揺れに追従できていないので、これを行なっても虚しい場合が多い。

(1) 検出器座標で集積したスペクトルに対する ARF

姿勢が揺れるということは、ソース位置が時間によって動くということである。そのソース位置を $r(t)$ とした場合、 $w_{\rm P}(\vec{r}_1)$ は次のように書ける。

$$w_{\rm P}(\vec{r}_1) = \int \delta(\vec{r}_1 - r(t))dt$$

従って、 Position-type を 'FITS-file' か 'FFT-FITS-file' として Position-file に上記のものを入れておけば、姿勢の揺れが考慮されることになる[29]。

(2) 天空座標で集積したスペクトルに対する ARF

この場合は、ソース位置にしたがって、リージョンも動くことになる。だったら Position-file と同じく Region-file も姿勢の揺れでなまされたものを jbldarf に与えればいい[30]、と思うかもしれないが、ソース位置とリージョンは相関を持つので、その方法では正しい ARF は作れない。正しくは、ソース位置ごとにいちいちリージョンを決めて ARF を作り[31]、それをそのソース位置の重みをつけて足し合わせるという方法をとらなければならない[32]。式で書くと、 (1) の $w_{\rm P}(\vec{r}_1)$ とそこでの ARF ($\mathrm{A}^{\vec{r}_1}[E]$ と書く) を使って、

$$\mathrm{A}[E] = \sum_{\vec{r}_1} w_{\rm P}(\vec{r}_1)\mathrm{A}^{\vec{r}_1}[E]$$

となる。 jbldarf を何度も走らせる必要があるので、計算時間は (1) よりも多くかかる。

[29] このような Position-file をつくる機能は、私が知る限りでは XSELECT にはついていない。 giphs を使う場合には makposmap というプログラムで作ることができる。

[30] この場合 Region-file は 0 か 1 だけではなく、そのピクセルの Exposure time に従って 0.5 とかの半端な値を持つことになる。

[31] 厳密には、回転の自由度もあるのでソース位置だけでは決まらないが、リージョンを円形にとった場合は、対称性により無視できる。それ以外の場合でも、リージョンが回転方向に 1arcmin 程度動いてもほとんど無視できるので、ソース位置だけで決まると思って問題ない。

[32] giphs を使う場合は、これを行なわせる shell script を発生する skyarf というプログラムがある。

3.5.2 広がったソースのレスポンス

ソースが広がっている場合でも、その全領域でスペクトルが同じだと仮定するなら、 Position-type を 'FITS-file' あるいは 'FFT-FITS-file' として、 Position-file にもともとのソースの形を入れることで、広がったソース用の ARF を作ることができる。本当は、もともとのソースの形はわからないはずであるが、他の衛星で得られているイメージやモデル、あるいは「あすか」のイメージを deconvolve したものを突っ込むのが妥当であろうと考えられる。こうやって作った ARF でスペクトルフィットした時のフラックスは、モデルで与えた全領域から出てくるフラックスとなる。しかし、大きく広がったイメージの場合、 XRT の PSF に「ちょうちょ」が入っていないことや、 PSF の tail がきちんと入っていないことが効いてくる可能性がある。

また、 GIS の場合、計算時間を考えると Position-type は 'FFT-FITS-file' にせざるをえないと思われるが、これだと PSF の場所による違いを反映することができないのも問題である。これを解決する策としては、jbldarf の外で Ray-tracing を走らせてその結果を jbldarf に取り込む、という方法がある。これについては、次の節で紹介する。

3.5.3 Ray-tracing の結果の利用 (GIS のみ)

通常 jbldarf は、 XRT のレスポンス関数 (§3.4.1参照) を使って XRT の有効面積を評価し、 ARF レスポンスを計算する。それとは別に、レスポンス関数を使わずに XRT のモンテカルロシミュレーション (Ray-tracing) の結果を参照して ARF を計算する機能もついている[33]。

その際に必要なものは、各エネルギーバンドごとの、検出器面上での光子数分布[34]を表わす FITS のイメージファイルである。サイズは 256×256、光軸を中心 (128.5, 128.5) とし、座標軸は衛星座標の X/Y 軸と平行[35]、 1pixel のサイズは 0.25mm でなければならない。また、あるエネルギーに対する FITS イメージファイルの各ピクセルの値の総和は、そのエネルギーでの XRT の有効面積になるように規格化しておく必要がある[36]。

このようなイメージを、例えば エネルギー 1keV でのものは psf010.fits、 1.1keV でのものは psf011.fits、というような名前[37]でひとつのディレクトリ (仮に *directory* とする) に置いておく。そうして、jbldarf で Position-type に 'XRT-PSF' を、 Position-file に *directory*/psf%03d.fits を指定すれば[38]、jbldarf は ARF を計算する時にこれらの FITS ファイルにアクセスして XRT の有効面積を評価するようになる。より正確に言えば $psf(E;\vec{r}_2)$ (§3.3.2参照) が FITS ファイルから読み込まれる。 ARF 計算時に、計算すべきエネルギー E に対応するイメージが存在しない場合は (ほとんどの場合そうだと思うが)、その前後のエネルギーのイメージから直線補間した値が使われる。

この方法は、広がったソースの解析等、jbldarf に入っている XRT のレスポンスでは ARF レスポンスの十分な評価ができない時に有効である[39]。また、 Ray-tracing を用いてイメージファイルを作るのに必要な時間を度外視すれば、jbldarf の計算時間も一般に速くなる。しかし、現状では Ray-tracing のプログラムは XRT チーム以外の人が簡単に使える形にはなっていないし、仮に使えたとしても、得られた結果にどの程度のエラーがあるかはなんともいえない。

この機能の別の使い方として、「ちょうちょ」の形を SIS のイメージから得る、という方法も考えられよう。この場合、 Ray-tracing は用いずに SIS で観測された実際のイメージを光子数分布だと仮定する。イメージファイルの規格化に用いる有効面積は、例えば、jbldarf に入っている XRT レスポンスを使用する[40]。この場合 Ray-tracing にかかる時間がないため、短時間で ARF を作成できるという利点があるが、 SIS は視野が狭い、十分な統計があるとは限らない、などの理由でこの方法が使える場面も限定されるであろう。

[33] この機能は GIS に対してのみ有効である。

[34] もちろん、これを Ray-tracing で求める。

[35] 従って、検出器座標とは Y 軸の向きが反対である

[36] XRT の有効面積は、通常、半径 6 分以内を積分した時の値で定義されるので、「(点源の場合) 半径 6 分以内を積分した値が有効面積になるように規格化すべき」というのがより正確な言い方である。

[37] これを読んでお気づきだと思うが、イメージは 0.1keV 刻みでしか与えられない。

[38] このように指定すると、 jbldarf は内部で sprintf() を用いて %03d のところに (エネルギー ×10) を展開する。エネルギーの単位は keV 。

[39] 実用例として、現在、 X 線源が全天一様に広がっていると仮定した CXB 用の ARF がこの方法で作られている。

[40] ARF ファイルの EFFAREA カラムを見れば簡単にわかる。

3.5.4 イメージ解析への利用 (GIS のみ)

jbldarf 本来の用途とは少し外れるが、 DEBUG パラメータを ON にした時に吐き出されるファイルをイメージ解析に利用することができる。 jbldarf はもともとのイメージ (Position-file) が XRT や 検出器でなまされる効果を内部で計算しているので、./map/exponnn.fits をエネルギーごとに重みをつけて足し合わせたものは、 GIS で検出されたイメージと同じになっているはずである。これを逆手にとれば、いろいろな広がりのモデルを Position-file に入れて検出が予想されるイメージを作り、 GIS のイメージと一番良く合ったものがもっともらしい広がりのモデル、といえよう。

もっとも、現状の XRT のレスポンスを考えると jbldarf が作ったイメージがどの程度正確に検出器上でのイメージを再現するかは疑問が残る。

第 4 章

関連ツール

この章で説明するツール類は jbldarf リリースキットの ./src/tools/ ディレクトリの中に収められている。

4.1 addarf

addarf は、複数の ARF ファイルの加重平均をとるプログラムである。名前には、'add' とついているが、実際は平均をとっているので気をつけて欲しい。使い方は、次のようになっている。

`addarf [-norm]` *Output-file* g_1 ARF_1 $\cdots$ g_n ARF_n

`-norm` オプションをつけた時とつけない時では、計算式は違っている。計算式は、出来る ARF を $\mathrm{A}[E]$、足し合わせる ARF を $\mathrm{A}_n[E]$、スケーリング係数を g_n とすると、次のようになる。

(1) `-norm` オプションをつけない時

$$\mathrm{A}[E] = \sum_{i=1}^{n} g_i \mathrm{A}_i[E] \Big/ \sum_{i=1}^{n} g_i$$

(2) `-norm` オプションをつけた時

$$\mathrm{A}[E] = \sum_{i=1}^{n} g'_i \mathrm{A}_i[E] \Big/ \sum_{i=1}^{n} g'_i$$

ただし、$g'_i = g_i \sum_{E} \mathrm{A}_{\mathit{1}}[E] \Big/ \sum_{E} \mathrm{A}_i[E]$

使い方としては、次の *SPEC1*, *SPEC2* のような 2 つのスペクトルとそれ用の ARF があった場合、(*SPEC1*+*SPEC2*) 用の ARF (ARF_{1+2}) は、近似的に次のようにして作ることができる。

スペクトル	積分時間	カウント数	ARF
SPEC1	T_1	N_1	ARF_1
SPEC2	T_2	N_2	ARF_2
(*SPEC1* + *SPEC2*)	$T_1 + T_2$	$N_1 + N_2$	ARF_{1+2}

(1) `-norm` オプションを使わない場合

`addarf` ARF_{1+2} T_1 ARF_1 T_2 ARF_2

(2) `-norm` オプションを使った場合

`addarf -norm` ARF_{1+2} N_1 ARF_1 N_2 ARF_2

本来は別々にスペクトルフィットを行なうのが正しいのだろうが、足してしまった方が簡単に統計も上げられるし、XSPEC の画面も繁雑にならないですむ。

4.2 makemap

makemap は Region-file や Position-file を作るためのサンプルプログラムである。使い方は、次のようになっている。

`makemap` *teldef-file* *center-X* *center-Y* *outer-radius* [*inner-radius*] > *Output-file* jbldarf と同じく、座標は検出器ミリメートル座標で指定する。正しいマップを作るためには、 jbldarf が使うのと同じ teldef ファイルを指定してやる必要がある。 *inner-radius* を省略した場合は、 0 がセットされる。リング状の領域しか指定できないため、これを単独で使用することにあまり意味はない[1]。しかし、こうやって作ったイメージを SIS のチップマップと AND をとったり (2.7.2 参照)、スムージングをかけたりすれば、使い道は広がるだろう。

4.3 mkarf.sh ◁U▷

mkarf.sh は shell script から jbldarf を使うサンプルプログラムであるが、座標の指定を検出器座標で行なえるという便利な特徴を持っている。使い方は、次のようになっている。

`mkarf.sh` *Output-file Detector-ID center-Xch center-Ych outer-radius*

上記のように引数を指定して動かしてやると、自動的に jbldarf を呼んで ARF を作成し、終了するようになっている。ただし参照する RMF は、 shell script の中でセンサーごとに定義しているので、これをきちんと設定しておく必要がある。

なお、 FTOOLS のバージョンによっては、**jbldarf.par** の中の パラメータの名前をすべて小文字にしないと、正常に動作しない場合がある。 ◁N▷

4.4 dummyrmf ◁U▷

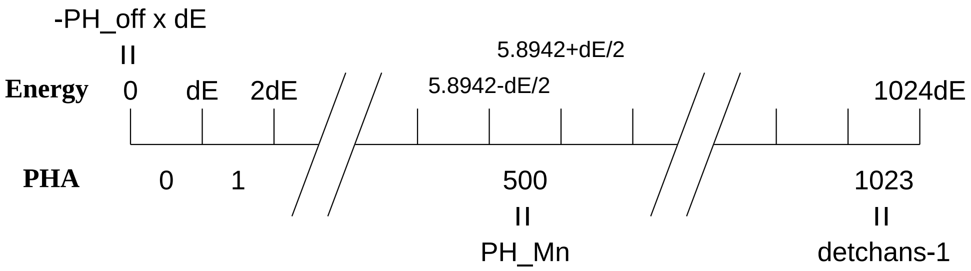
dummyrmf は対角成分のみが 1 の、ダミーのレスポンスファイルを作るプログラムである。使い方は、次のようになっている。

`dummyrmf` *Output-RMF-file PH_Mn PH_off Emin Emax detchans*

XSPEC で検出器のレスポンスを無視してスペクトルをフィットしたい時などに使えるだろう。 FTOOLS のパラメータインターフェイスを使ったプログラムになっているので、実行には dummyrmf.par が必要である。各パラメータの意味は、次の通り。

Output-RMF-file	作成するダミーレスポンスのファイル名。
PH_Mn	5.8942 keV (Mn の K_α 線) のエネルギーが対応する PHA チャンネル。
PH_off	0 keV のエネルギーが対応する PHA チャンネル。
Emin	E_MIN がこれよりも大きいカラムが RMF に書かれる。
Emax	E_MAX がこれよりも小さいカラムが RMF に書かれる。
detchans	PHA のチャンネル数。

dummyrmf で作成した RMF レスポンスの Energy と PHA の関係は、 $PH\_Mn = 500$、 $PH\_off = 0$、 $detchans = 1024$ とした場合、次の図のようになる。 Energy 0 keV に対応するところが PHA の最初のビンの下端になること、 PHA が 0 から始まること、の 2 点に注意して欲しい。



[1] ARF のヘッダに情報が残る分だけ、 jbldarf に直接パラメータで指定した方が得である。

4.5 pressrmf

pressrmf は、 RMF のマトリックス成分が 0 のところを圧縮して、 RMF ファイルを小さくするプログラムである。使い方は、次のようになっている。

`pressrmf` *Input-RMF-file* [*Output-file*]

Output-file の指定がないときは、 *Input-RMF-file* にオーバーライトされる。 sisbldrsp2 で作られるレスポンスや、 GIS チームからリリースされる RMF はすでに圧縮されているので、 pressrmf をかける必要はない。

4.6 addarfrmf ◁N▷

addarfrmf は、複数の ARF×RMF ファイルの加重平均をとるプログラムである。平均のアルゴリズムは addarf と同様なので、そちらを参照されたい。使い方は次のようになっている。

`addarfrmf [-norm]` *Output-file* g_1 ARF_1 RMF_1 $\cdots$ g_n ARF_n RMF_n

addarf と異なって、作成されるファイルは RMF レスポンスであり、 ARF も含んだものとなる。 SIS の場合は、チップごとに RMF レスポンスも異なるので、 addarf では不十分で、 addarfrmf を使った方が良いだろう。

4.7 gisnortarf ◁N▷

jbldarf で作成した GIS の ARF レスポンスは、強力ライズタイムカット (gisclean) のかかったスペクトル用であるが、 gisnortarf は、これを強力ライズタイムカットのかかっていないスペクトル用の ARF レスポンスへ変換するツールである。使い方は次のようになっている。

`gisnortarf` *ARF-file* *qdp-flag*

指定した *ARF-file* を直接書き換えるようになっている。 *qdp-flag* は、変換定数を qdp ファイルに書きだすかどうかのフラグである。 qdp ファイルの名前は、 *ARF-file* で指定したファイルの拡張子を`.qdp` としたものが付けられる。ヘッダに gisnortarf をかけたという情報を書き込むので、 2 回以上、同じ *ARF-file* に gisnortarf をかけることはできない。 FTOOLS のパラメータインターフェイスを使ったプログラムになっているので、実行には gisnortarf.par が必要である。

第 5 章

FAQ

FAQ とは、「Frequently Asded Question の略で『よくある質問と、その回答集』と言う意味」です。(UNIX-SIG FAQ List with partial answers (Last Update: 19-Feb-1994) より)

質問 1. 誰が jbldarf を作っているの？

答. おもに、著者 (石崎、平山) がプログラムを書いています。内訳を言うと、 SIS ARF の計算部を平山、 GIS ARF の計算部を石崎が書き、ユーザインターフェース部は石崎、平山が共同で書いています。 XRT と GIS のレスポンスの部分はそれぞれ、 XRT、 GIS 両チームから公表された関数を使いますので、 XRT、 GIS 両チームの方々にもご参加いただいていることになります。

質問 2. どうして RMF を読むの？

答. jbldarf は、エネルギーの刻み (ENERG_LO、 ENERG_HI のカラム) を RMF から読み込んで、同じ刻みで ARF を計算します。また、 TELESCOP や INSTRUME などのヘッダの値も RMF からコピーされるようになっています。これらはいずれも、 XSPEC でのスペクトルフィットの際のデータの整合性を確保するために行なっています。

なお XSPEC は RMF と ARF でエネルギーの刻みが同じになっているかどうかをチェックしないようなので[1]、 RMF と ARF の組み合わせを間違えるととんでもないフィット結果が得られたりしますので、気をつけましょう。

質問 3. むかし作った ARF が XSPEC や addarf で読めなくなっちゃったんだけど

答. 0.50 より前の jbldarf では 1st extension の名前が 'ARF' となっていて、 XSPEC 8.40 や jbldarf-0.50 に付属の addarf ではこのような ARF は受け付けられません。 FTOOLS の fmodhead などを使って、次のようにヘッダを書き換えれば、読めるようになります。

```
EXTNAME = 'SPECRESP' / name of this binary table extension
```

質問 4. XSELECT で saoimage の中から選んだ領域をそのまま jbldarf に指定すると、大きさが微妙に違うような気がするのですが？

答. 一般に、 XSELECT から選んだ方が領域が大きくなるようです。今のところはっきりとした原因はわかっていませんが、池辺さんが次のようなことじゃないかと教えてくれました。

IRAF では、ピクセルがちょっとでも指定した範囲に入っていれば、そのピクセルは選ばれたことになるそうです。一方、 jbldarf では、ピクセルの中心が、境界を含む範囲に入っていれば選ばれたことになります。そのため IRAF で選んだ領域の方が普通大きくなるそうです。 XSELECT も、これと同じように領域が指定されるのでしょう。

なお、互換性が損なわれるため jbldarf を上記のように変更する予定はありません。

[1] 少なくとも Version 8.34 ではそうだった。ビン数が同じかどうかはチェックする。

質問 5. jbldarf は FTOOL なの？

答. FITS ファイルを取り扱うツールではありますが、 GSFC から出されている FTOOL の規定には沿っていないので、そういう点では FTOOL ではありません。ちなみに、 GSFC からは、 ARF 計算用の FTOOL として ASCAARF というものがリリースされています。詳しくは、 fhelp などで調べてみて下さい。

質問 6. jbldarf は表街道なの？

答. 座標系などは表街道で使われているものを採用しています。そのため、場合によっては、裏街道ユーザには使いにくい面もあるかも知れません。しかし、 GSFC からリリースされている訳ではありませんから、表街道とも呼べないでしょう。

質問 7. システムパラメータファイルがないっていわれるんですけど

答. このエラーメッセージは、環境変数の `SYSPFILES` で指定されるディレクトリに **jbldarf.par** がない時に表示されます。 (FTOOLS 3.0 以降の libhost.a を使用した場合にこの問題が起こります)

解決方法としては、次の 3 つのやり方があります。

(1) システムマネージャに頼んで、 **jbldarf.par** をシステム領域にコピーしてもらう。

(2) 環境変数の `PFILES` からシステムパラメータファイルのパスを除く。

(3) 環境変数の `PFCLOBBER` を unsetenv する。

方法 (1) がもっとも望ましいですが、やむを得ない時は、方法 (2) や (3) をとる必要があります。ただし、方法 (2) または (3) をとった時は、他の FTOOLS の動作にも影響を与えるので、注意して下さい。