

もし天における天体観測と解析のすすめ

東北大天文学部生 SLA 一同

2022 年 10 月 10 日

目次

第 1 章	はじめに	1
1.1	作成経緯と目標	1
1.2	テキストの構成	2
1.3	掲載内容についての諸注意	3
第 2 章	天文学の基本	5
2.1	天体の種類	5
2.2	天文学の観測的方法	5
2.3	電磁波とは	5
2.4	天体からの放射	5
第 3 章	天文学における可視光観測の基本	7
3.1	撮像観測の基本	7
3.2	分光観測の基本	7
第 4 章	もし天における観測	9
第 5 章	天文学における解析の基本	11
5.1	撮像データ解析	13
5.2	分光データ解析	29
第 6 章	天文データ考察の基本	31
第 7 章	終わりに	33
参考文献		35
付録		37
A	プログラミング環境の構築	37

B	Python 実行環境の構築	40
C	IRAF 実行環境の構築	40
D	バグ取りの翁	40
E	Makali'i 実習	40
F	SAOImage DS9 実習	40

図目次

5.1	撮像/分光観測データの解析の流れ	11
5.2	1次処理の仕組み	12
5.3	生データ画像	14
5.4	ダークの結合画像	17
5.5	ダーク引き済みの画像	19
5.6	フラット画像	21
5.7	フラットフィールディング済みの画像	23
5.8	スカイ画像	24
5.9	sky引きまでの処理が終わった画像	25
A.1	VSCode のインストール画面	38
A.2	Error Lens によって表示された警告メッセージ	39

表目次

第 1 章

はじめに

1.1 作成経緯と目標

もし天とは、東北大学天文学教室、宮城教育大学、加速キッチン合同会社、仙台市天文台が主催している高校生向けの研究体験事業で、全国の宇宙、科学に興味のある高校生たちが天文学者としての研究を体験するものです。2022 年で 12 年目となり、多くのもし天卒業生、通称「もしチル」が多方面で活躍されています。また、SLA (Student Learning Assistant) の参加者の中にも、もしチルの方々が多く在籍しています。

そんな「もし天」ですが、もし天に限らず、天文学における観測は単に写真を撮ることにとどまりません。観測の後には必ずそのデータが存在し、データがどんな意味を持つのかを読み取るために、その解析を行う必要があります。

このテキストは、解析を行う SLA の方のために、光・赤外における観測とは何か、撮像観測と分光観測、また観測画像の処理等についてを解説し、その負担を軽減する目的で作られたものです。

解析の方法にはデータ毎に様々な方法があります。「もし天」で行う観測としては主に「撮像観測」と「分光観測」があります。これらの観測データの解析には歴史的に「IRAF」^[1]という、アメリカ国立光学天文台 (NOAO) が開発した画像解析ソフトが使われてきました。もし天でもこれまで、このソフトを使って画像解析を行ってきました。

しかし先日、IRAF の NOAO の公式のサポートが今後行われない、つまりディスコンになるという発表がありました。今後の天文学の研究では、IRAF ではなく Python の「Astropy」モジュールを使用した解析が行われていくと考えられます。この Astropy は、現状撮像観測には対応していますが、分光観測には対応していません。そのため、分光観測には IRAF を使い続けることになりますが、撮像観測にはもし天での解析についても Python への移行が必要となります。

また、もし天に参加されたことのある方ならご存知かもしれません、もし天の実習期間

中の観測可能な時間は非常に限られたものです。その中で、参加高校生の研究テーマに見合うだけの膨大な量のデータの解析を行わなければならず、毎年、SLA が毎夜のように徹夜で作業している光景が当たり前となっていました。これはもし天の存続性という観点や、SLA の健康という観点からも大変問題であり、解決する必要があります。

以上の理由から、

- ・「撮像観測での Python を使った解析」と「分光観測での IRAF を使った解析」のわかりやすい解説
- ・ SLA の負担軽減

を目指した解説資料が必要との判断から、東北大天文の SLA が解説資料を作成することとなりました。

1.2 テキストの構成

このテキストでは主に以下のことを解説しています。

- ・ 天文学の基本 → [2 章](#)
- ・ 天文学における可視光観測の基本 → [3 章](#)
 - 撮像観測
 - 分光観測
- ・ もし天における観測 → [4 章](#)
- ・ 天文学における解析の基本 → [5 章](#)
 - 撮像データ解析
 - 分光データ解析
- ・ 天文学におけるデータ考察の基本 → [6 章](#)
- ・ 付録 → [7 章](#)
 - プログラミング環境の構築
 - Python 実行環境の構築
 - IRAF 実行環境の構築
 - バグ取りの翁
 - Makali'i 実習
 - SAOImage DS9 実習

1.3 掲載内容についての諸注意

このテキストで解説されている内容は、東北大学天文学教室の学部3年向け通年授業「天体観測」の講義内容や、2011年度のもし天観測班の方の資料を参考に、書籍等を参照して肉付けしたものになっています。また、添付のPythonコードや、別資料の解析コードについては、「天体観測」の講義内容や、その担当教員である板先生の作成したものや、板研究室の資料を参考にしたものとなっています。

従って、文書、コードについては板さんをはじめとした作成者に、著作権を要求しないような広く一般に普及した部分を除いて権利が帰属しますので、無断で外部への配布を行ったり、公的な文書への転載を行ったりしないように注意してください。これは、万が一このテキストの内容に間違いがあったりした場合に責任を取りきれなくなってしまうためです。

また、この文書には文書内の相互リンク機能がついています。具体的には「[2章](#)」の数字部分をクリックすると、[2章](#)の初めのページにリンクされるようになっています。その他にも節のラベルや、図、数式のラベルにも同様のリンクがついています。適宜活用して頂けると幸いです。

第 2 章

天文学の基本

2.1 天体の種類

2.1.1 惑星

2.1.2 恒星

2.1.3 分子雲

2.1.4 コンパクト天体

2.1.5 銀河/銀河団

2.2 天文学の観測的方法

2.3 電磁波とは

2.4 天体からの放射

2.4.1 連続光

2.4.2 輝線/吸収線

第3章

天文学における可視光観測の基本

3.1 撮像観測の基本

3.2 分光観測の基本

第4章

もし天における観測

第 5 章

天文学における解析の基本

観測によって得られたデータをもとに科学的な議論を行うためには、適切な調理を行うことで、必要な情報が得られるようにしなければなりません。ここでは実際のデータを例にしてデータ解析の基本をおさえていきたいと思います。

解析の流れは図 5.1 のようになります。3 章の流れに則って得られた観測データのことを、ここでは**生データ**と呼ぶことにします。生データに自分の欲しい天体からの光だけが入っていれば苦労することはないのですが、**検出機由来の雑音や夜光などの影響**を受けています。図 5.2 がそれを模式的に表したものです。実際に式的にその影響を表すと

$$(生データ) = (\text{感度の違い}) \times \{(\text{真の天体信号}) + (\text{暗電流}) + (\text{夜光などの雑音})\} \quad (5.1)$$

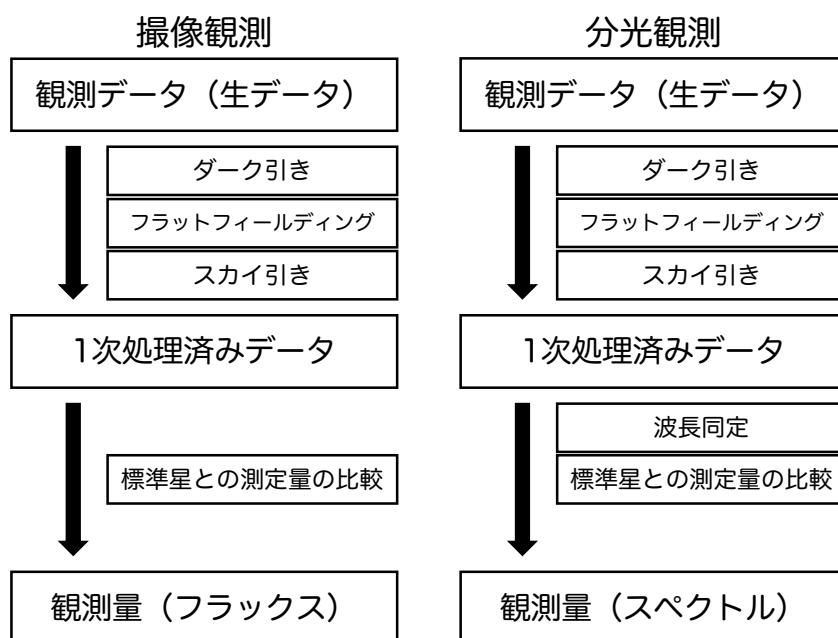


図 5.1 撮像/分光観測データの解析の流れ

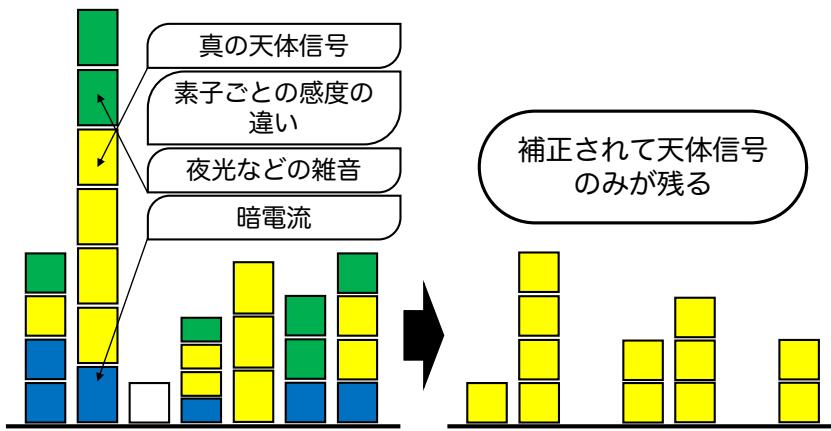


図 5.2 1次処理の仕組み。黄色が天体からの光、青が暗電流（ダーク）、緑がスカイ成分を表す。また、大きさの違いは検出機の素子の感度が場所によって異なることを表し、色がついていないのはその素子が死んでいることを表す。このような、観測したい天体のみの情報を使って議論をしたいにも関わらず、混ざってしまう雑音などを取り除く作業が1次処理である。

のようになります。

暗電流とは、検出機が温度を持っていることで流れてしまう電子雑音のことです。天体放射の検出の仕組みは大まかに

$$(天体からの光子) \xrightarrow{\text{検出器}} (\text{光電効果による電子}) \rightarrow (\text{光電流の発生})$$

となっています。この二つ目の段階で、天体放射やその他の雑音に加えて、熱雑音による電流が加わります。これを「暗電流」と呼びます。この雑音は単純に検出器を冷やせば減らすことができます。実際、赤外線検出器は -76 K 、市販の可視光の検出器は -10 K に冷やして使われています。しかし、天体からの放射がわからなくなるほどの雑音の影響を軽減することはできても、完全に 0 にすることは不可能なため、この成分を引く作業が必要となります。この作業を「暗電流」が「ダーク」と呼ばれることから、「ダーク引き」と呼びます。

CCD カメラなどの検出機は、多数の素子によって構成されていますが、望遠鏡や、付属する検出器を合わせた系の、光子検出に対する感度は必ずしも一様ではありません。とても強く光子に対して反応する素子も存在する一方で、全く反応しない、「死んだ素子」も存在します。このような感度の違いを補正する処理を フラットフィールディングと呼びます。

「夜光などの雑音」とはその名の通り、夜光や大気による雑音の成分です。今回考える主な成分は

- 主に K バンドより長い波長で、大気や望遠鏡の熱的な放射成分を見てしまう ($\sim 300\text{ K}$)
- 主に H バンドで、OH 夜光の輝線成分によるフリンジパターン

です。これを以降は「スカイ」と呼び、スカイを引く補正のことを、「スカイ引き」と呼ぶことにします。

これらの補正のことをまとめて**1次処理**と呼びます。撮像、分光のどちらでもこの処理が必要になりますが、分光はまとめて IRAF でやってしまった方が楽なので、撮像では Python、分光では IRAF を使った処理を行い、観測量を得るにはどうすればよいかをまとめていきます。

5.1 撮像データ解析

今回サンプルとして扱うデータは以下の URL に置いてあります。

https://drive.google.com/drive/folders/1ZGkWFySTgc-MVdViP_U_AfD8XPUrSbWi

中身は次のようになっています。`raw` が天体を撮像した生データ、`flat` がフラットフィールディングに用いるデータ、`dark` がダーク引きで用いるデータです。

```
dark % ls
ir0029.fits  ir0030.fits  ir0031.fits  ir0032.fits  ir0033.fits
flat % ls
ir0006.fits  ir0007.fits  ir0008.fits  ir0009.fits  ir0010.fits
raw % ls
ir0011.fits  ir0012.fits  ir0013.fits  ir0014.fits  ir0015.fits
ir0016.fits  ir0017.fits  ir0018.fits  ir0019.fits
```

また、解析コードとして `initial_data_reduction.py` と `photometry.py` が付属しています。まずは `initial_data_reduction.py` の内容を扱います。

解析にさきだって、今回扱う生データを見てみましょう。図 5.3 です。星のようなものが写ってはいますが、あまり綺麗とは言えない状態です。これは先ほど述べた様々な雑音などの成分の影響を受けているためです。これから行う処理によって綺麗な画像になります。

コードの実行の仕方

ここでは Python やその他言語の扱いに慣れていない方のために、Python のコード実行の仕方を説明します。慣れている方はそのまま読み飛ばしてください。

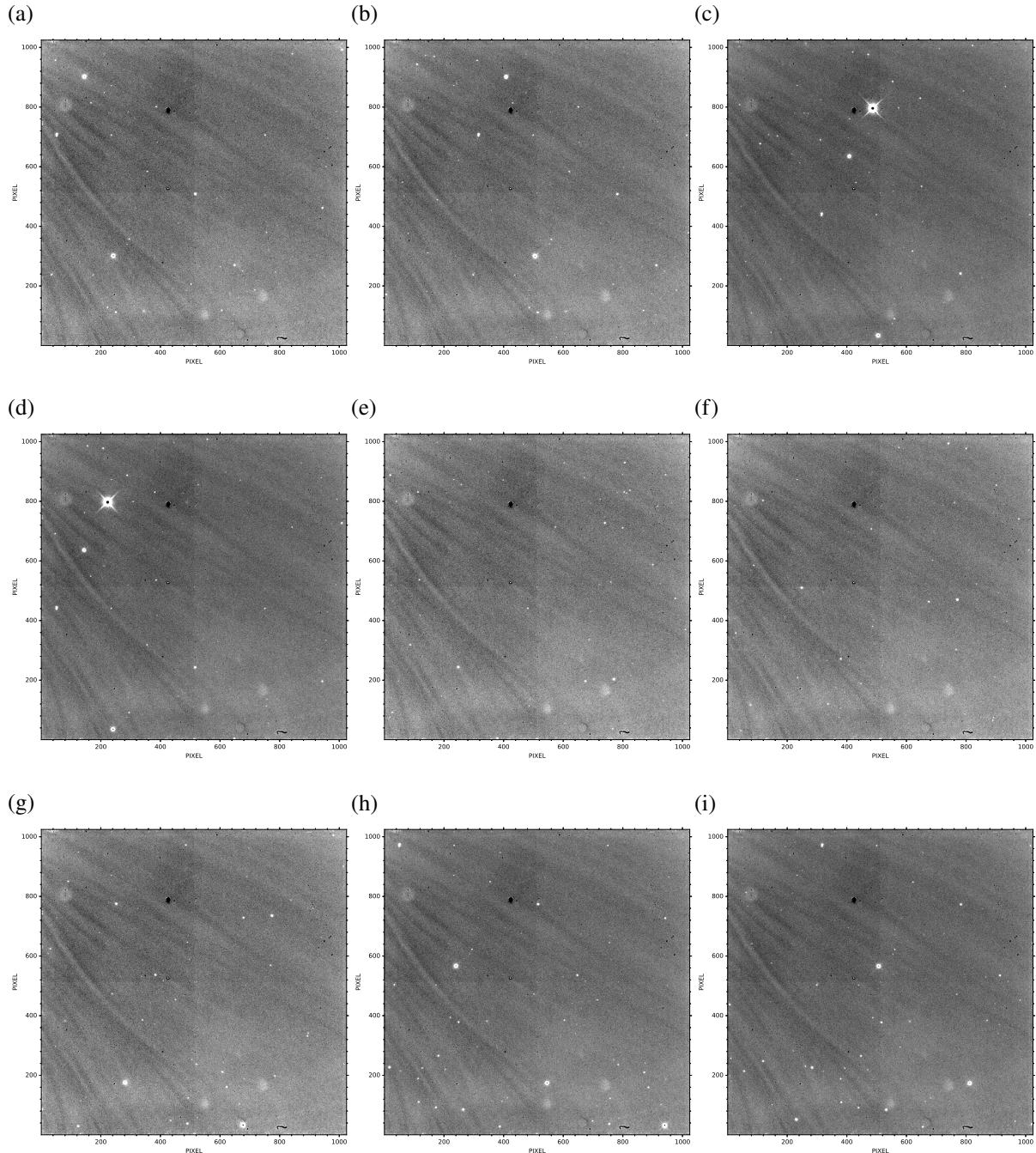


図 5.3 生データ画像

バイアス引き

実際の作業に入る前に、「バイアス引き」について解説します。バイアスとは、人為的にデータに足されている量です。これは、本来観測で得られる出力値が、不良素子の影響で負になることを防ぐために足される一定値です。この値を調べるために、「露出 0 秒」で画像を撮れば良いことになります。しかし、これはダークの成分にも含まれているので、ダークを引けばバイアスを引いたことにもなりますから、この作業は必要ありません。

ダーク引き

「ダーク」はカメラのシャッター、または望遠鏡の蓋を閉じた状態で、**天体の露出時間と同じ時間撮影を行うこと**によって出力されるものです。実際、ダークがどんな画像なのかを見てみましょう。

まずは準備として、必要なモジュールを import します。次のコードを実行してください。

Listing 5.1 モジュールの import

```
1 import os
2 import re
3 import numpy as np
4 import astropy.io.fits as fits
5 import glob
6 import matplotlib.pyplot as plt
7 import aplpy
8
9 # create a directory if that directory does not exist
10 def my_makedirs(path):
11     if not os.path.isdir(path):
12         os.makedirs(path)
13 my_makedirs('./out')
14
15
16 # functions for the function "sorted()"
17 def atoi(text):
18     return int(text) if text.isdigit() else text
19 def natural_keys(text):
20     return [ atoi(c) for c in re.split(r'(\d+)', text) ]
```

import から始まる文が今回の 1 次処理で使用するモジュール類です。また、引数として取った文字列を名前として持つディレクトリが存在しない場合に、ディレクトリを作成する関数 my_makedirs を作り、out というディレクトリを作成しました。これから出力さ

れる.fits ファイルは全てここに保存するようにします。

また、sorted という関数をこの後に使用するため、この関数を使用するのに必要な関数を2つ定義しています。

次にダーク画像を作成します。ダークは複数回撮ることが一般的なので、それらを全て結合する必要があります。今回は5枚とあるので、それらを結合します。処理としては次のようになります。

Listing 5.2 ダーク画像の作成

```

1 # dark
2 ## merge the dark images
3 dark_images = np.empty((0, 1024, 1024))
4 dark.fits = sorted(glob.glob('./dark/*.fits'), key=natural_keys)
5 for data in dark.fits:
6     dark = fits.getdata(data)
7     dark_images = np.append(dark_images, dark[np.newaxis, :], axis
8                             =0)
9 median_dark = np.median(dark_images, axis=0)
10 fits.writeto('./out/dark.fits', median_dark, overwrite=True)

```

ここでの流れは次のようになっています。

- (1) $1024 \times 1024 \times 0$ の3次元の画像を用意
- (2) 今回扱うダークのfitsを全て取得
- (3) 取得したダークfitsでループ処理
 - データを一つずつ取得してその中央値をとる
- (4) できた画像を保存

できたfits形式のファイルを表示してみます。下のコードを実行すると、図5.4がfigというフォルダの中にあるはずです。

Listing 5.3 ダーク画像の表示

```

1 my_makedirs('./fig')
2
3 img = fits.open('./out/dark.fits')
4 pic = aplpy.FITSFigure(img)
5 pic.show_grayscale()
6 pic.save('./fig/dark.fits.pdf')

```

明るい点が複数個確認できると思います。これは数理統計で習う、正規分布に従ったノ

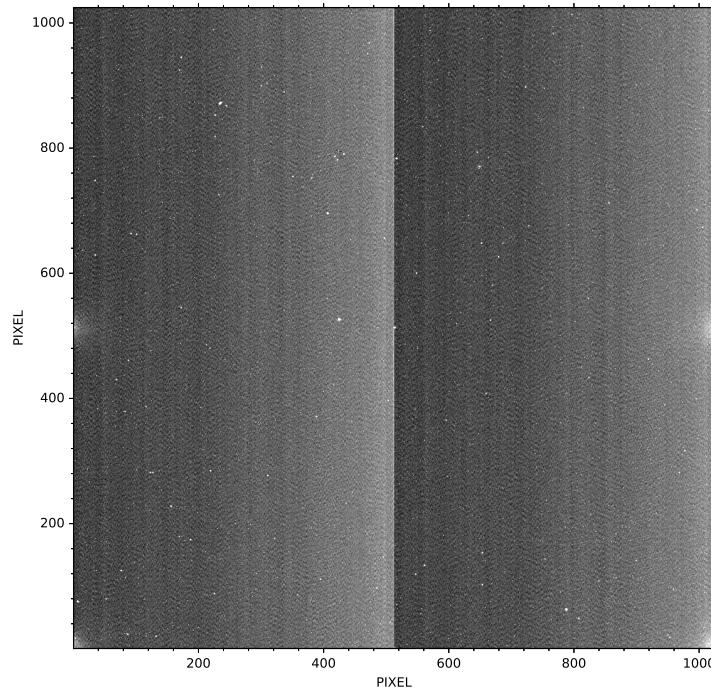


図 5.4 ダークの結合画像

ノイズの成分で、よく「 3σ ノイズ」だったり、「 5σ ノイズ」と読んだりします。この σ は標準偏差の意味です。正規分布表と調べたりすると出てくるので、適宜調べて欲しいのですが、 3σ は大体 $1/1000$ の確率で発生する事象ということになります。今回のダークや天体画像のピクセル数は 1024×1024 で、 10^6 のオーダーなので、1枚の画像に 1000 の、リアルでないノイズが存在することになります。 5σ であれば大体 $1/100000$ なので、10 個くらいです。

このダークを生データから引きます。

Listing 5.4 生データからダークを引く

```

1  ## subtract the dark data from raw data
2  dark = fits.getdata('./out/dark.fits')
3  raw_fits = sorted(glob.glob('./raw/*.fits'), key=natural_keys)
4  for i,data in enumerate(raw_fits):
5      raw_image = fits.getdata(data)
6      sub_dark = raw_image - dark
7      file_name = 'dir_' + str('{:02d}'.format(i)) + '.fits'
8      fits.writeto('./out/' + file_name, sub_dark, overwrite=True)

```

これによって、ダークによって生まれてしまう成分をデータから引くことができたことに

なり、式(5.1)的に言えば、これで得られる画像は

$$(感度の違い) \times \{(真の天体信号) + (夜光などの雑音)\} \quad (5.2)$$

となっているはずです。ここで注意しなければならないのは、ダークにも検出機の素子の感度の違いがのってしまっていることです。できた画像を表示してみます。

Listing 5.5 ダーク引き済み画像の表示

```

1 dir_images = sorted(glob.glob('./out/dir_*.fits'), key=
2     natural_keys)
3 for i,data in enumerate(dir_images):
4     img = fits.open(data)
5     pic = aplpy.FITSFigure(img)
6     pic.show_grayscale()
7     img_name = 'fig_dir_' + str('{:02d}'.format(i)) + '.pdf'
8     pic.save('./fig/' + img_name)

```

このコードによって作成される画像が図 5.5 です。正直あまり変わった気がしないと思います。これは、十分に冷やされた検出機を使っている場合、熱雑音の成分が目に見えるほど、画像内で支配的になることはないためです。

フラットフィールディング

次に、フラットフィールディングの作業に入ります。フラット割りとも言います。現在のデータの状況は

$$(生データ) \xrightarrow{\text{ダーク引き}} (感度の違い) \times \{(真の天体信号) + (夜光などの雑音)\}$$

であって、フラット"割り"というのは、割り算の意味です。感度の違いを補正する画像を作って、割り算をする、ということになります。

この「フラット」処理に必要な補正用画像の作成方法には主に次のような方法があります。

- ドームフラット
 - フラット盤に一様光を反射させる。
 - 「ライトをつけた状態で撮ったもの」から「ライトを消した状態で撮ったもの」を引けばフラット画像から熱雑音成分が消せる。
 - 精度悪め。
 - 天候に左右されない。
 - 東北大天文の屋上望遠鏡のドームにもフラット盤があるので確認してみて欲しい。

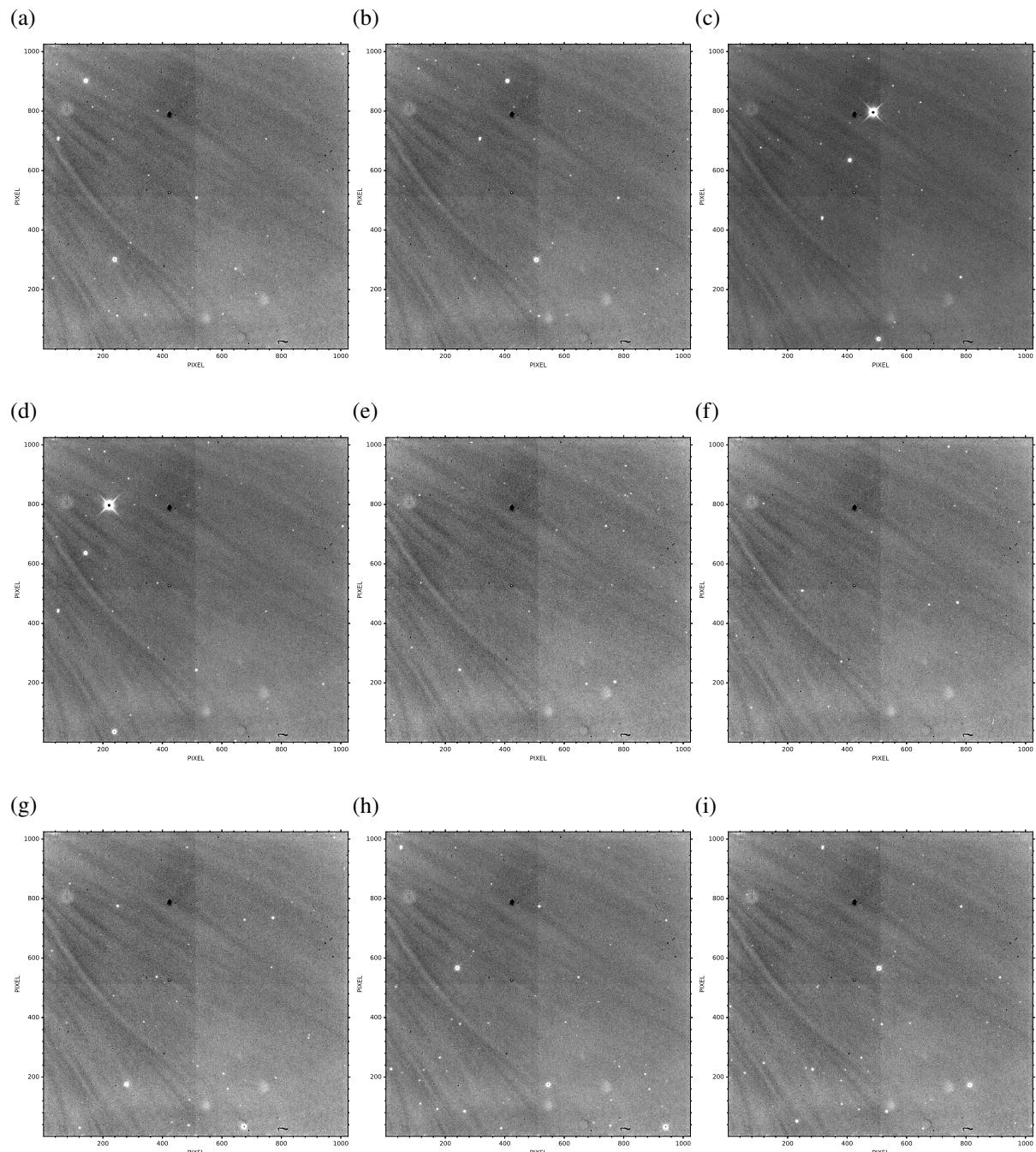


図 5.5 ダーク引き済みの画像

- 雲
 - 雲の散乱光を観測する。
 - 雲が一様に光っていると考え、雲の一部を観測する。
 - ドームフラットよりも精度が良い。
- トワイライト
 - 夕暮れ時、明け方に撮る。
 - 太陽が沈む、または昇る時の、明るすぎない時の空を観測する。

今回はこの中でも、処理がより簡単なドームフラットを扱うことにします。ドームフラットの工程は次の通りです。

- (1) ランプ ON のドームフラットの平均画像を作成する。
- (2) ランプ OFF のドームフラットの平均画像を作成する。
- (3) ON から OFF を引いた画像を作成する。
- (4) この画像を規格化する。

まずはランプ ON のドームフラットの平均画像を作成します。ダークのときと同様に、画像を結合させれば良いので、

Listing 5.6 ランプ ON のフラット画像

```

1 # flat
2 ## create the flat image
3 ### light on
4 light_on = np.empty((0, 1024, 1024))
5 light_on_fits = sorted(glob.glob('./flat/*.fits')), key=
   natural_keys)
6 for data in light_on_fits:
7     light_on_image = fits.getdata(data)
8     light_on = np.append(light_on, light_on_image[np.newaxis, :],
   axis=0)
9
10 median_light_on = np.median(light_on, axis=0)

```

とします。

また、ランプ OFF の画像ですが、これは作成の仕方からわかるように、これはダーク画像と同等のものなので、今回はダーク画像をランプ OFF の画像として扱いましょう。

Listing 5.7 ランプ OFF のフラット画像とフラット画像の作成

```

1 ### identify the median dark image as the flat off image
2 flat_image = median_light_on - median_dark

```

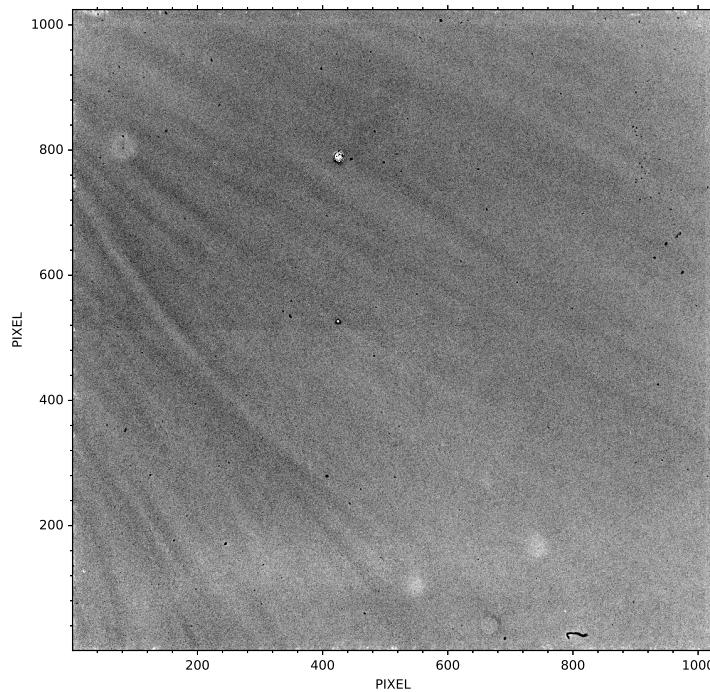


図 5.6 フラット画像

出来上がった flat 画像を規格化します。規格化について説明する前に、コード ?? を先に示します。

Listing 5.8 フラット画像の規格化

```

1  ## normalize the flat image
2  median = np.median(flat_image)
3  stddev = np.std(flat_image)
4  normalized = np.where(flat_image/median < 0.001, 9999, flat_image
   /median)
5  fits.writeto('./out/flat.fits', normalized, overwrite=True)

```

要するに規格化とは、死んだ素子を除外することです。「ON から OFF を引いた画像」で値が極端に小さくなってしまっている素子、具体的には

`on_minus_off/median < 0.001`

であるような素子の情報は、一様に削ぎ落としてしまいたいものです。そこで、flat 画像のその素子の値に 9999 という極端な値を入れてしまいます。後でデータを flat で割るのですが、極端に大きな値で割ってしまえば、その素子のデータは極端に 0 に限りなく近い値となって、無視できるようになるという原理です。ここで作成された flat 画像が図 5.6 です。では、作成した flat 画像でデータを割る作業に移ります。割るという作業も単純な演

算で、

Listing 5.9 フラット割り

```

1 ## flat the images
2 dir_images = sorted(glob.glob('./out/dir_*.fits'), key=
3     natural_keys)
4 for i, data in enumerate(dir_images):
5     img = fits.getdata(data)
6     flat = img/normalized
7     file_name = 'fdir_' + str('{:02d}'.format(i)) + '.fits'
8     fits.writeto('./out/' + file_name, flat, overwrite=True)

```

となります。ここまで作業が完了した fdir 画像を表示してみると図 5.7 のようになります。よくみると画像に模様のようなものがうっすらと入っています。これは次のスカイ引きの処理で取り除くものなので、ここでは問題ありません。

スカイ引き

1 次処理の最後はスカイ引きの処理に入ります。スカイ画像は、感度補正が終わった全ての画像を重ねて、その中央値をとれば作成できます。

Listing 5.10 スカイ画像の作成

```

1 # sky
2 ## create the sky image
3 sky_images = np.empty((0, 1024, 1024))
4 fdir_images = sorted(glob.glob('./out/fdir_*.fits'), key=
5     natural_keys)
6 for data in fdir_images:
7     sky = fits.getdata(data)
8     sky_images = np.append(sky_images, sky[np.newaxis, :], axis=0)
9 median_sky = np.median(sky_images, axis=0)
10 fits.writeto('./out/sky.fits', median_sky, overwrite=True)

```

できたスカイ画像が図 5.8 のようになります。星が消えて、模様のようなものだけ見えています。ここではダーク引きと同様に単純に引くだけなので、

Listing 5.11 スカイ引き

```

1 ## subtract the sky data from fdir data
2 for i, data in enumerate(fdir_images):
3     img = fits.getdata(data)

```

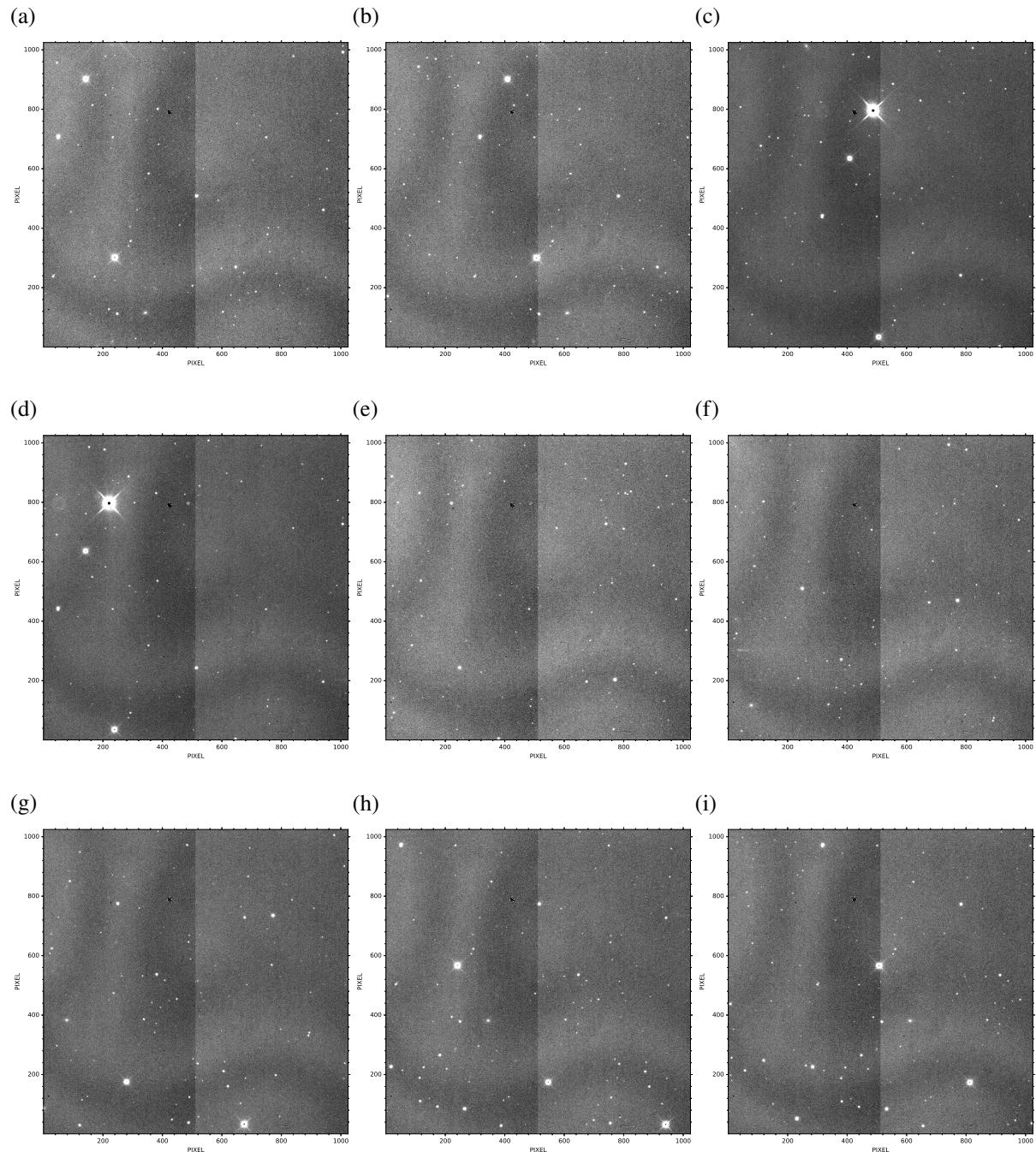


図 5.7 フラットフィールディング済みの画像

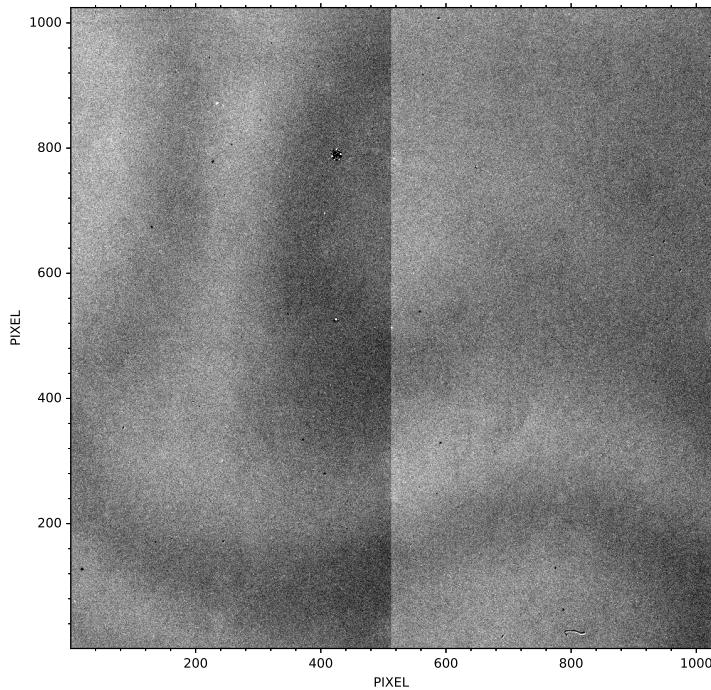


図 5.8 スカイ画像

```

4     sub_sky = img - median_sky
5     file_name = 'sfdir_' + str('{:02d}'.format(i)) + '.fits'
6     fits.writeto('./out/' + file_name, sub_sky, overwrite=True)

```

となります。ここでも例によって、「`s`」をつけてスカイ引き済みという意味にしています。できた画像を表示させてみます。コード的には次のようにします。

Listing 5.12 作成した画像の表示

```

1  sfdir_images = sorted(glob.glob('./out/sfdir_*.fits'), key=
   natural_keys)
2  for i,data in enumerate(sfdir_images):
3      img = fits.open(data)
4      pic = aplpy.FITSFigure(img)
5      pic.show_grayscale()
6      img_name = 'fig_' + str('{:02d}'.format(i)) + '.pdf'
7      pic.save('./fig/' + img_name)

```

これで表示される画像は図 5.9 です。変な模様が消えたので、綺麗な画像になっているのがわかります。これで 1 次処理は終わりです。

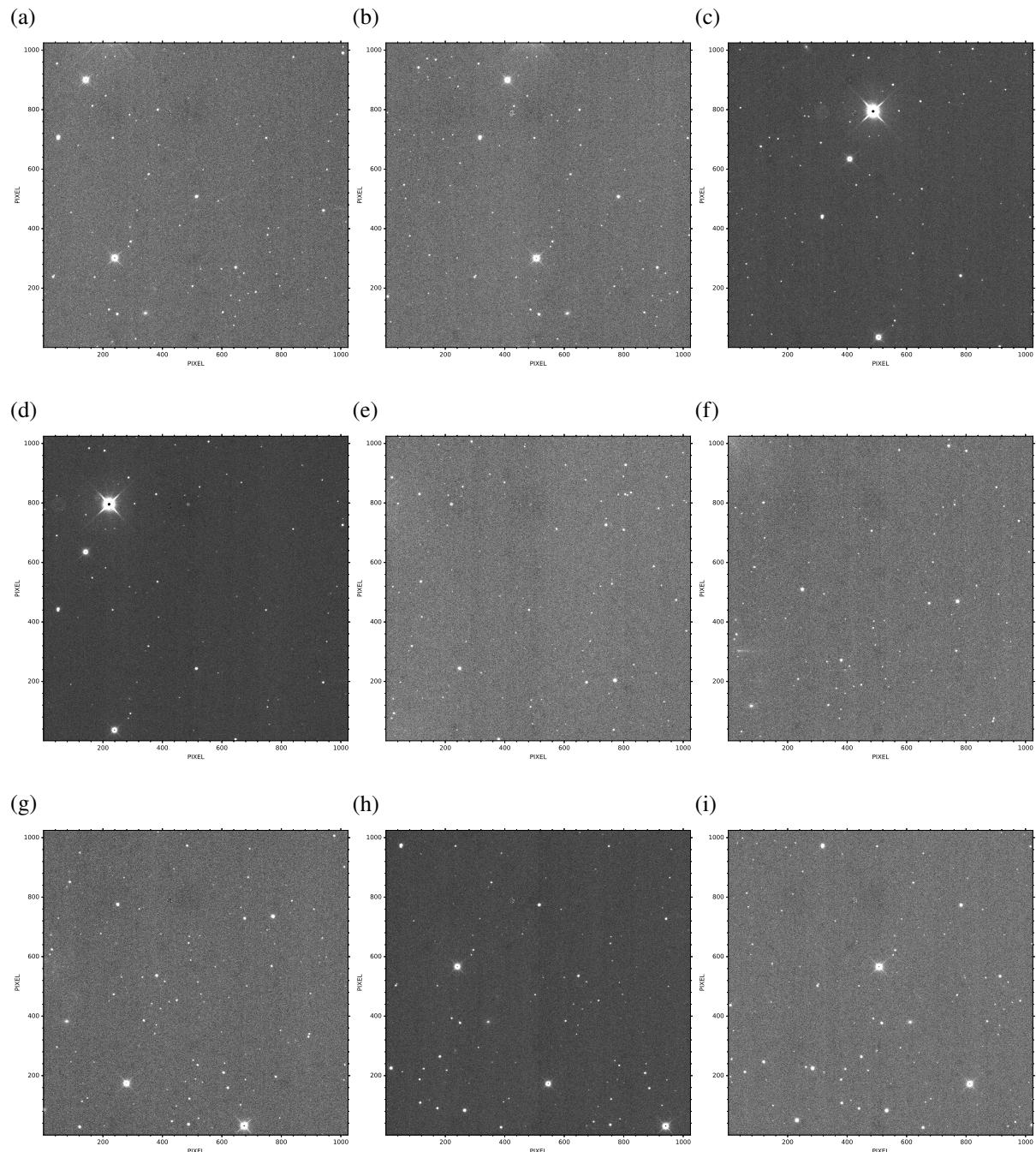


図 5.9 sky 引きまでの処理が終わった画像

1次処理のまとめ

ここまでで行った1次処理の解析コードをまとめます。実行的には、`raw`に観測の生データ、`dark`にダークのデータ、`flat`にフラット画像を作る際のライトオンのデータを入れてしまえば、以下のコードを走らせるだけで1次処理が完了します^{*1}。

Listing 5.13 1次処理のまとめ

```
1 import os
2 import re
3 import numpy as np
4 import astropy.io.fits as fits
5 import glob
6 import matplotlib.pyplot as plt
7 import aplpy
8
9 # create a directory if that directory does not exist
10 def my_makedirs(path):
11     if not os.path.isdir(path):
12         os.makedirs(path)
13 my_makedirs('./out')
14
15
16 # functions for the function "sorted()"
17 def atoi(text):
18     return int(text) if text.isdigit() else text
19 def natural_keys(text):
20     return [ atoi(c) for c in re.split(r'(\d+)', text) ]
21
22
23 # dark
24 ## merge the dark images
25 dark_images = np.empty((0, 1024, 1024))
26 dark_fits = sorted(glob.glob('./dark/*.fits'), key=natural_keys)
27 for data in dark_fits:
28     dark = fits.getdata(data)
```

*1 ただし、2021年度のもし天のように、データ自体に不具合が生じている場合、例えば生データ取得時とフラットデータ取得時でCCDの角度が異なっているなどした時には、画像を回転させるなどの処置が必要になります。当日そのような事態におちいると非常に焦るので、あらかじめどのような不具合が想定されるかどうか考えておくとよいかもしれません。

```
29     dark_images = np.append(dark_images, dark[np.newaxis, :], axis
   =0)
30
31 median_dark = np.median(dark_images, axis=0)
32 fits.writeto('./out/dark.fits', median_dark, overwrite=True)
33
34 ## subtract the dark data from raw data
35 dark = fits.getdata('./out/dark.fits')
36 raw_fits = sorted(glob.glob('./raw/*.fits'), key=natural_keys)
37 for i,data in enumerate(raw_fits):
38     raw_image = fits.getdata(data)
39     sub_dark = raw_image - dark
40     file_name = 'dir_' + str('{:02d}'.format(i)) + '.fits'
41     fits.writeto('./out/' + file_name, sub_dark, overwrite=True)
42
43
44 # flat
45 ## create the flat images
46 ### light on
47 light_on = np.empty((0, 1024, 1024))
48 light_on_fits = sorted(glob.glob('./flat/*.fits'), key=
   natural_keys)
49 for data in light_on_fits:
50     light_on_image = fits.getdata(data)
51     light_on = np.append(light_on, light_on_image[np.newaxis, :], 
   axis=0)
52
53 median_light_on = np.median(light_on, axis=0)
54
55 ### identify the median dark image as the flat off image
56 flat_image = median_light_on - median_dark
57
58 ## normalize the flat image
59 median = np.median(flat_image)
60 stddev = np.std(flat_image)
61 normalized = np.where(flat_image/median < 0.001, 9999, flat_image
   /median)
62 fits.writeto('./out/flat.fits', normalized, overwrite=True)
63
64 ## flat the images
65 dir_images = sorted(glob.glob('./out/dir_*.fits'), key=
```

```
    natural_keys)
66 print(dir_images)
67 for i, data in enumerate(dir_images):
68     img = fits.getdata(data)
69     flat = img/normalized
70     file_name = 'fdir_' + str('{:02d}'.format(i)) + '.fits'
71     fits.writeto('./out/' + file_name, flat, overwrite=True)
72
73
74 # sky
75 ## create the sky image
76 sky_images = np.empty((0, 1024, 1024))
77 fdir_images = sorted(glob.glob('./out/fdir_*.fits'), key=
    natural_keys)
78 for data in fdir_images:
79     sky = fits.getdata(data)
80     sky_images = np.append(sky_images, sky[np.newaxis, :], axis=0)
81
82 median_sky = np.median(sky_images, axis=0)
83 fits.writeto('./out/sky.fits', median_sky, overwrite=True)
84
85 ## subtract the sky data from fdir data
86 for i, data in enumerate(fdir_images):
87     img = fits.getdata(data)
88     sub_sky = img - median_sky
89     file_name = 'sfdir_' + str('{:02d}'.format(i)) + '.fits'
90     fits.writeto('./out/' + file_name, sub_sky, overwrite=True)
91
92 # visualize the image
93 my_makedirs('./fig')
94 sfdir_images = sorted(glob.glob('./out/sfdir_*.fits'), key=
    natural_keys)
95 for i,data in enumerate(sfdir_images):
96     img = fits.open(data)
97     pic = aplpy.FITSFigure(img)
98     pic.show_grayscale()
99     img_name = 'fig_' + str('{:02d}'.format(i)) + '.pdf'
100    pic.save('./fig/' + img_name)
```

測光処理（2次処理）

5.2 分光データ解析

第 6 章

天文データ考察の基本

第 7 章

終わりに

参考文献

- [1] NOAO. Iraf - image reduction and analysis facility, 2019. [http://ast.noao.edu/
data/software](http://ast.noao.edu/data/software).

付録

A プログラミング環境の構築

もし天に限らず、データ解析にはプログラミングが必須となります。まず環境がなければ解析を行うことはできません。ひと口にプログラミングと言っても、様々な言語が存在し、それぞれに特色があります。例えば Python の実行環境には Anaconda 社提供の「Anaconda」^{*1}や Google 社提供の「Google Colaboratory」^{*2}などの有名なものがありますが、ここでは統合開発環境をおすすめしたいと思います。Anaconda や Google Colaboratory は Python に特化した環境になっていますが、統合開発環境は自分で拡張機能を追加することによって、自分好みのスタイルで Python だけでなく、C/C++ や HTML/CSS、LATEX などの様々な言語を扱うことができるものです。統合開発環境にも同じく様々なものがありますが、その中でも特に有名と言える、Microsoft 社提供の「Visual Studio Code」通称「VSCode」^{*3}をおすすめします。VSCode は 2015 年が最初のリリースと、非常に新しいエディタですが、さすが Microsoft というべきか、今では非常に多くの人に使われています。個人的には Github 社提供の「Atom」という統合開発環境が好きでずっと使っていたのですが、Github 社が Microsoft 社に買収された影響もあって、2022 年の 12 月で開発終了にまで追い込まれてしまいました。

A.1 VSCode のインストール

まずはインストールしてみましょう。脚注の URL で検索すると、インストール画面が出てきます。図 A.1 がその画面です。どういう仕組みなのかはわからないのですが、自分の PC 環境にあった VSCode のバージョンが「Code editing. Redefined.」の下に出ているはずです。私の環境は intel チップの Mac なのでこの表示になっていますが、自分の PC 環境に本当に合っているのかどうか確かめてからインストールしましょう。Mac ではない場

^{*1} <https://www.anaconda.com/products/individual>

^{*2} <https://colab.research.google.com>

^{*3} <https://code.visualstudio.com/>

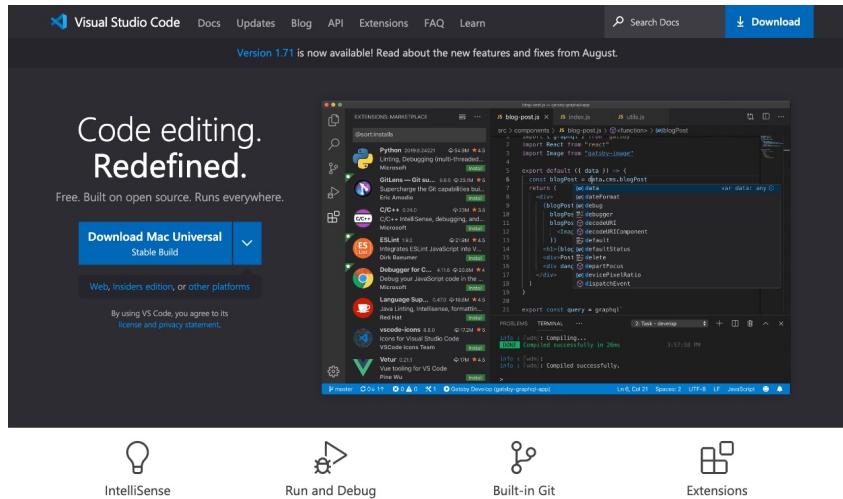


図 A.1 VSCode のインストール画面

合は異なる表示になっていると思います。

ダウンロードが終わるとダウンロードフォルダに `VSCode-darwin-universal.zip` みたいな zip ファイルがあるはずです。これをクリックすると VSCode というアプリケーションが解凍されます。これをアプリケーションフォルダに移動すれば、インストール完了です。

A.2 各種設定

次に、使用に便利な一般的な拡張機能を入れてみましょう。ブラウザで「VSCode 拡張機能 おすすめ」とか調べるととても参考になる記事が多いですが、そんな中から筆者が実際に使っているものをピックアップしていくつか紹介します。

vscode-icons

`vscode-icons` は VSCode で表示されるファイル類のアイコンをおしゃれにしてくれる機能です。アイコンがあると作業の際に視覚的にファイルを区別できて便利だと思います。

indent-rainbow

`indent-rainbow` はコード内のインデントを色分けして表示してくれる機能です。コーディングの際にはインデントを使って視認性を高くすることが重要です。また、Python ではインデントがあることが `for` 文や `if` 文などで重要となります。そのインデントのレベルをわかりやすくしてくれます。

```
22 median_dark = np.median(dark_images, axis=0)
23 fits.writeto('./out/dark.fits', median_dark, overwrite=True) Define a constant instead of duplicating this literal './out/dark.fits' 3 times. [+2 locations]
24
25 ## subtract the dark data from raw data
26 dark = fits.getdata('./out/dark.fits')
27 raw_fits = glob.glob('./raw/*.fits')
28 for i,data in enumerate(raw_fits):
29     raw_image = fits.getdata(data)
30     sub_dark = raw_image - dark
31     file_name = 'dir_{:02}'.format(i) + '.fits' Define a constant instead of duplicating this literal '.fits' 3 times. [+2 locations]
32     fits.writeto('./out/' + file_name, sub_dark, overwrite=True) Define a constant instead of duplicating this literal './out/' 3 times. [+2 locations]
33
34
```

図 A.2 Error Lens によって表示された警告メッセージ

Code Spell Checker

Code Spell Checker は、コードのスペルミスを指摘してくれる機能です。若干曲者で、Python の正しいコマンド名なのにスペルミスを指摘されることがあったり、固有名詞に対してスペルミスを指摘してきたりしますが、ユーザー辞書みたいなのに追加すれば対処できます。コードエラーの原因は些細なスペルミスだったりするので、便利な機能です。

Error Lens

Error Lens は警告などをわかりやすく表示してくれる機能です。例えば図 A.2 のような感じで警告を表示してくれます。

Path Intellisense

Path Intellisense はディレクトリ名の補完をしてくれます。Python に限らず、コード内で相対的にディレクトリ指定を行うことは非常に多いので

Prettier

Prettier

SonarLint

SonarLint

- B Python 実行環境の構築
- C IRAF 実行環境の構築
- D バグ取りの翁
- E Makali'i 実習
- F SAOImage DS9 実習