C 言語と IRAF を活用した解析ツールの開発

松永典之 (東京大学 天文学教育研究センター 木曽観測所)

2011 年 5 月 18 日

1　本講習の目的

本講習では、C 言語で書いたプログラムから IRAF のタスクを呼んで解析を行います。これまでの IRAF 講習会では、cl スクリプトを書くか、python, perl などのスクリプト言語から IRAF を利用することが主に議論されてきたようです。一方、C 言語はコンパイルを必要とするプログラミング言語です。C 言語の長所・短所について、教科書の紹介などを見ると

- もっとも普及しているプログラミング言語の一つ。
- 多くの OS 自身が C 言語で書かれていて、ハードウェアを扱うための低水準な処理もできるため、応用できる範囲が広い。
- 自由度が高くどんな処理でもできる代わりに、ソースの書き方によっては安全性が低くなる。

などと書かれています。私自身はあまり多くの言語を使って試したことがないので、他の言語とあまり比較できるわけではありませんが、C 言語で特に不便や不都合を感じたことはありません。cl スクリプトはあまり書きやすくないように思いますし、超常現象が起こる可能性があるので安心して使えないという欠点があります。

なお、C 言語から IRAF タスクを呼ぶといっても、実際にこれから紹介するのは、`system` 関数と `popen` 関数というものを使って、シェルに IRAF のタスクを実行させるだけです。それでは C 言語でもシェルスクリプトでも変わらないじゃないかと言われると、まあそうかもしれませんが、C 言語と組み合わせる方がより複雑な作業を一緒にできるのではないかと思います。

今回の講習では、IRAF のタスクを組み合わせて解析するツールを、実際に C 言語のプログラムとして作っていただきます。解析の内容としては、バイアスを引いて、フラットで割るといった基本的な処理から、画像に座標系 (WCS) を入れて 3 色カラー合成を作るということも行います。

1.1 サンプルプログラムとデータ

本講習で利用するサンプルプログラムとデータは、
`new-r14:/data/matsunaga/sample.tar`
にあります。各自作業をするディレクトリにコピーして解凍してください。

```
cp -p /data/matsunaag/sample.tar .
tar xvf sample.tar
cd sample
```

sample ディレクトリの中には、以下のディレクトリがあります。

- sample1 : サンプルプログラム 1 (3 ページ)
- sample2and3 : サンプルプログラム 2 (4 ページ)、および 3 (4 ページ)
- sample4 : サンプルプログラム 4 (5 ページ)
- data1 : サンプルプログラム 2–4、および課題 1 (6 ページ) で利用するデータ
- data2 : 課題 2 (7 ページ) で利用するデータ
- kadai1, kadai2 : 課題 1 (6 ページ)、課題 2 (7 ページ) でプログラムを作る作業はここで行ってください。data1, data2 のデータは、このディレクトリにコピーしても、あるいは実行時に与える引数などでデータのあるディレクトリを参照するようにしても構いません。

C 言語のプログラムをコンパイルする場合には、

```
gcc -Wall sample2.c -o sample2
```

などのように警告も出すオプションをつけて行いましょう。-o オプションをつけることで、出力する実行形式のファイル名を指定できます。

2 IRAF のタスクをシェルから動かす

まず、IRAF を起動させてインタラクティブに作業するのではなく、外部から IRAF のタスクを呼んで実行させる方法を見てみましょう。

今回 IRAF を呼ぶために用いる方法は、本質的には 2010 年度第 3 回の IRAF 講習会で板由房氏 (東北大) が紹介しているのと同じ方法です。つまり、IRAF のプロンプトに対して打ち込むようなコマンドの内容を書き並べておいて、それをシェル上で cl というコマンドへ渡してやります。

cl_sample1

cl_sample1 というファイルを見てみてください。

```
images
imutil
imarith ../data1/raw1.fits - ../data1/bias.fits out.fits
logout
```

images, imutil, logout と書かれた行を除けば、何のことはない imarith のタスクを実行するだけですね。images, imutil はそれぞれ images パッケージ、imutil パッケージをロードさせている部分です。imarith というタスクは、images.imutil というパッケージに含まれているタスクなので、それを実行する前にパッケージをロードしておきます。最後に終了するときには logout します。これらを実行するには、コマンドラインで

```
(*)         cl -old < cl_sample1 > log_sample1
```

と入力してみてください。out.fits という画像ができるはずです。imarith が実行されたのです。ただし、実行するワーキングディレクトリに login.cl が必要です。(ディレクトリに login.cl が無いときは、他の場所からコピーするか、mkiraf してください。)

以上の呼び出しは、どのように働いているのでしょうか。cl のコマンドは、インタラクティブに IRAF を起動させるときにも使いますね。-old オプションについては、バージョン 2.14 以降の IRAF が ECL(Enhanced CL=履歴の応用機能などを使える進化したインターフェース) をデフォルトで使うことになって入出力の形式が変わったことから、ECL を使わない単純な入力を与えるために、このオプションを加えるようになりました。いずれにせよ、cl コマンドは IRAF を起動させます。ためしに、cl -old とコマンドラインで打ってみてください。IRAF が起動するはずです。ここで、cl_sample1 の内容をタイピングするとどうなりますか。普段、インタラクティブに作業するのと同じように、imarith が実行され、さらに logout したところで IRAF が終了しますね。上で、入力リダイレクト (<) を使って cl_sample1 を cl に与えたのはそれと同じことをしているのです。

したがって、IRAF でどういう作業をすればよいかわかっていれば、それを単に書き並べたファイルを作って cl に与えてやればよいのです。IRAF の使い方そのものに関しては、以上がこの講習会で紹介をする全てです。あとは、C 言語にせよ、シェルスクリプトにせよ、状況に応じて実行すべきタスクをファイルに書き並べて、それを適切に実行していくようなプログラムを書けば解析ツールの出来上がりです。残されたのは、いろいろな条件の判断や使うべきタスク・パラメータの決定などを行う部分をどのようにして書くかという問題で、本講習では C 言語を利用してそれを行います。

なお、他にも外部から IRAF のタスクを呼んで実行させる方法はいくつかあります。過去の IRAF 講習会のテキストなどを調べてみてください。

3 C 言語のプログラムからシェルのコマンドを呼ぶ

いろいろな方法がありますが、system と popen という関数を利用します。これらの関数は、stdio と stdlib という基本のライブラリに入っていて、使い方も簡単です。

```
int system(const char *command);
FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

sample2.c

system はコマンドを書いた文字列を渡して、それを実行させるというものです。一方、popen は、与えたコマンドの実行結果をファイル型へのポインタ (FILE *) で受けることができます。これによって、実行結果をそのままプログラムの中で読み取って、次の計算などに使うことが出来ます。sample2.c を見てください。ディレクトリ内にある FITS ファイルを ls で表示させて、そのファイルにたいして imarith で 2 倍するというコマンドを作成しています（ただし、ここでは imarith を実行させてはいません）。

```
cd sample2and3/
gcc -Wall sample2.c -o sample2
./sample2
```

sample3.c

このままでは、imarith に与える FITS ファイルと出力する FITS ファイルの名前が同じです。imarith の動作としてはこれでも問題はないのですが、入力ファイルを上書きすることになります。生データはなるべく残しておきたいので、出力する名前を変更できるようにしましょう。.fits の直前に 2 という文字をいれることにします。sample3.c では、string という文字列操作のライブラリにある strcpy, strstr, sprintf という関数を使っています。

```
char *strcpy(char *dest, const char *src);
char *strstr(const char haystack, const char *needle);
int sprintf(char *str, const char *format, ...);
```

```
cd sample2and3/
gcc -Wall sample3.c -o sample3
./sample3
```

4　C言語のプログラムからIRAFのタスクを動かす

sample4.c

それでは、imarithを実際にC言語のプログラムから呼んで、計算を実行させましょう。まず、2節で見たcl_sample1のようなものを実行させることを考えます。勘のよい方は、3ページの(*)のコマンドをsystem関数で呼んでしまえばよいと想像がつきますね。

sample4.cは、コマンドラインからimarithに必要な引数を与えて、それに対応してIRAFをプログラム中から起動・操作して、imarithを実行します。C言語で、引数を与えるには、main(int argc, char *argv[])のようにmain関数を使えばオーケーです。

```
cd sample4/
vi sample4.c　(コンパイルする前にLOGIN_CLを自分の使うファイル名に直してください)
gcc -Wall sample4.c -o sample4
./sample4  ../data1/raw1.fits  -  ../data1/bias.fits  out.fits
```

と実行すれば、imarithで引き算を行って、out.fitsが作成されます。

sample4.cでは、すこし応用がきくように

```
imarith  input.fits - inputb.fits output.fits
```

というタスクを実行させるような関数を作ってあります。do_imarithという関数がそれです。このような関数を作っておくことで、他の画像ファイルについて同じ作業を行うときにも、同じ関数を使い回すことができます。また、login.clをコピーするための関数copy_login_clも作っています。IRAFのタスクを実行するためには、ワーキングディレクトリにlogin.clが必要なので、これを決められたディレクトリからコピーします。login.clの所在については、定数マクロLOGIN_CLで定義してあります。

なお、copy_login_clの中で、commandが指す配列のサイズをあらかじめ与えずにmalloc関数を使って動的にメモリを確保していますね。これは、たとえば後でLOGIN_CLの定義が非常に長い文字列になっても使えるようにしておく工夫です。文字列を使い終わったところで、free関数でメモリを開放しています。いろいろな解析をしている最中で、非常に長いファイル名を使うことになってもプログラムが勝手に対応させるための書き方です。argv

の中身を、あらかじめメモリ領域を確保した配列に strcpy などでコピーするのではなく、すでに文字列を格納してくれている argv[1], argv[2] ... などへのポインタを char *変数に代入して使うというのも、余計な文字列長の心配をしなくてよいための工夫です。

動的なメモリの使い方については、知っていると便利ですが、ここではどうしても必要というわけではありません。ただ、C 言語の特徴として、何でもできる反面、セキュリティ面で弱いということの大きな原因のひとつが、文字列の長さに関するものだと思いますので、このような使い方をマスターしておくとよいのではないかと思います。たとえば、sample3.c では、80 文字分しかメモリを確保していない file へ、ls -1 から返ってくる何文字分かわからないファイル名を読み込もうとするので、バッファオーバーランの起こる可能性があります。その問題点や解決方法については (他人に使われるソフトを作る上では非常に重要ですね…)、ウェブ上の文書等を調べて見てください。

5 課題 1: ディレクトリ内の画像の一次処理を行う

さて、sample3.c と sample4.c を改良して、ディレクトリ内にある FITS 画像の一次処理を行うというプログラムを作ってみましょう。(kadai1 のディレクトリに、kadai1.c という名前のソースファイルを書いてください。)

ここでは、ディレクトリの中にある FITS ファイルに対して、バイアス画像を引いて、フラット画像を割るという演算をさせることにします。このために、バイアス画像とフラット画像は引数で名前を指定できるようにしましょう。さらにバイアス画像とフラット画像自身にはそのような演算を行わないことにします。また、できればディレクトリの名前も指定できるようにして、

```
./kadai1  ../data1  ../bias.fits  ../flat.fits
```

のように実行すれば、../data1 の中にある bias.fits flat.fits 以外の*.fits に対しバイアス引き算、フラット割り算するようなプログラムにしましょう。いろいろな状況を考え出すとバイアスかフラットかの判断など難しい部分もありますが、sample3.c, sample4.c を参考にしながら、各自の C 言語の習熟度に応じて可能な範囲でプログラミングしてください。(たとえば、バイアスとフラットは必ず bias.fits, flat.fits という名前で、それ以外の FITS ファイルには bias または flat という文字列が出てこないという条件を決めるとだいぶ難易度が下がります。さらに、sample3.c のように末尾を b.fits と変えたときに他のファイルと名前が重複してしまうことがないものと仮定しても良いです。C 言語に慣れている方は、そのような条件がなくても動くようなプログラムを考えてみてください。)

なお、いろいろな状況で使えるツールとするためには、エラーが起こったときの対処をできるようなプログラムであることが必要です。たとえば、`sample2.c` などで `popen` して開くはずのファイルポインタが得られない場合には、エラーメッセージを出して強制終了するようになっています。この課題のプログラムで引数として指定されたバイアス画像やフラット画像が無い場合に、`imarith` を呼ばずに終了させるためにはどうしたらよいでしょうか。

6 課題 2: 3 バンドの画像に WCS を入れて、3 色合成を行う

せっかくなので、なにか楽しいことのできるツールを作ってみましょう。通常、光赤外線天文学で取得する画像は何らかの波長フィルターを通して露光したもので、それぞれの画像は単色です。そこで、複数の波長でとった画像を重ね合わせて擬似的なカラー画像を作成することがよく行われます。この課題では、近赤外線 3 バンド (JHKs) でとられた画像からカラー画像を合成してみましょう。`data2` のディレクトリには、`j.fits, h.fits, k.fits` という 3 枚の FITS ファイルがあります。それぞれ、J バンド ($\sim 1.25\mu$m) H バンド ($\sim 1.63\mu$m), Ks バンド ($\sim 2.14\mu$m) のフィルターでとられた画像です。また、`catalog.xym` と `param` はあとで座標系を求める時に使うファイルです。

きれいに 3 枚の画像を合成するためには、互いにどのようにずれているかを調べてそのずれをシフトした上で重ねないと、同じ星がずれて見えてしまいます。そのために、ここでは画像の WCS(World Coordinate System=世界座標系) を求めてやって、その情報を使って `ds9` にきれいに重ねた状態で表示させるということを行います。WCS は、その画像が天球座標上でどちらの方向をどのように投影させて 2 次元画像としているかという情報を表します。これを求めるためには、画像に映っている星を検出し、それを天球座標のわかっている星のリストと比較をして、どういう天球座標の星がどのピクセルに映っているかを調べる必要があります。以下のように作業を進めます。

- 各画像に映っている星を検出してリストを作成する。
- 観測した視野についてすでに得られている天体のリストとの同定を行う。
- 同定した結果から、画像がどういう WCS を持っているかを調べ、それを FITS ファイルに入れる。

ここでは、まず上の作業を 1 ステップずつ行ってみて、その後、C 言語のプログラムからそれらを実行できるように、ひとつのプログラムへ統合するところを課題としましょう。

6.1 星の検出

画像の星を検出するのには、apphot パッケージの daofind タスクを用います。daofind タスクを実行する上で重要なパラメータは、画像に映っている星の FWHM、バックグラウンドの標準偏差、何シグマ以上の星を検出するかというくらいですね。このうち、FWHM はちょっと面倒なので最初からわかっているものとしましょう。data2 ディレクトリにある 3 つの画像はいずれも FWHM が 2.5 pixel くらいです (星を検出するだけなので正確な値は必要ありません)。バックグラウンドの標準偏差は、せっかくなので自分たちで求めることにします。

```
imstatistics j.fits nclip=2 lsigma=3. usigma=3.
```

とすれば、$\pm 3\,\sigma$ 以上平均からはなれたカウントをもつピクセルを計算から除くというクリッピングを 2 回行った上で、平均などの統計値を求めてくれるので、星の影響などもおおよそ取り除いた上でバックグラウンドの値が求められます。あとで、C 言語から結果を読み取るときには、出力の形式もシンプルであった方が良いので、

```
imstatistics j.fits fields="stddev" format- nclip=2 lsigma=3. usigma=3.
```

とオプションを増やして実行してみてください。求めた標準偏差だけが出力されます。

次に、daofind で使うパラメータの設定を行います。とりあえず、

```
digiphot   (まず digiphot, apphot をロード)
apphot
unlearn daofind
unlearn datapars
unlearn findpars
```

というコマンドを実行してください。unlearn は、パラメータを初期化するためのタスクで、daofind のパラメータと、daofind が間接的に参照することになる datapars, findpars というパラメータセットを初期化しました。さて、eparam などで datapars のパラメータを見てみると、

```
PACKAGE = apphot
   TASK = datapars
(scale  =                   1.) Image scale in units per pixel
(fwhmpsf=                  2.5) FWHM of the PSF in scale units
(emissio=                  yes) Features are positive ?
(sigma  =                INDEF) Standard deviation of background in counts
(datamin=                INDEF) Minimum good data value
```

```
(datamax=                INDEF) Maximum good data value
(noise  =              poisson) Noise model
(ccdread=                    ) CCD readout noise image header keyword
(gain   =                    ) CCD gain image header keyword
(readnoi=                   0.) CCD readout noise in electrons
(epadu  =                   1.) Gain in electrons per count
(exposur=                    ) Exposure time image header keyword
(airmass=                    ) Airmass image header keyword
(filter =                    ) Filter image header keyword
(obstime=                    ) Time of observation image header keyword
(itime  =                   1.) Exposure time
(xairmas=                INDEF) Airmass
(ifilter=                INDEF) Filter
(otime  =                INDEF) Time of observation
(mode   =                   ql)
```

などと出てきます。ほとんどのパラメータはデフォルトのままでとりあえず問題ありませんが、fwhmpsf と sigma は実際のパラメータに変更しましょう。eparam でインタラクティブに変更するだけでは、外部のプログラムから作業ができないので、IRAF のコマンドラインで

```
datapars.fwhmpsf=2.5
datapars.sigma=30.
```

のようにタイプしてみてください。ちゃんと datapars のパラメータが変更されます。findpars の中では、threshold を適当な値に変更したいので、

```
findpars.threshold=5.
```

としてください。こちらもパラメータが変更できます。

パラメータを変更したら、daofind を実行します。インタラクティブにパラメータの確認を行うこともできますが (デフォルトはこちら)、ここでは verify=no というオプションをつけてください。また、出力するファイルの名前も指定します。

```
daofind j.fits output="j.coo" verify=no
```

ただし、IRAF の一般的な規則として、すでに同じ名前のファイルが存在する場合は、上書きをせずにそこでエラーとなってタスクが終了します。プログラミングなどをするときには、ファイル名の衝突などが起こらないように注意してください。

さて、daofind で星を検出したら、ピクセル座標と等級だけのリストに直しましょう。次の作業ではこのリストを使います。

```
pdump j.coo xcen,ycen,mag yes > j.xym
```

JHK の 3 枚の画像に対して、以上の作業を行って、ピクセル座標と等級のリスト (j.xym, h.xym, k.xym) を作ってください。

6.2 カタログとの同定と WCS の埋め込み

カタログとの同定と WCS の埋め込みを行うために、OPM というソフトを利用します。このソフトは筆者が開発したもので、Optimistic Patter Matching (Tabur, 2007, PASA, 24, 180) というアルゴリズムを用いて、位置座標と等級が求められている 2 つのカタログの同定と座標の変換式を求めるものです。http://www.ioa.s.u-tokyo.ac.jp/ nmatsuna/Japanese/software/OPM.html で公開しているので、ソフトに興味のある方はそちらをご覧ください。

catalog.xym というファイルは、赤経 17:47:11.13, 赤緯-26:49:10.4 という座標の周りで 2MASS サーベイが同定した天体の位置 (東西南北に何秒離れているか) と H バンドの等級をリストしたものです。OPM ソフトを使って、このリストと前節で作った JHK バンドでの画像に写っている星のリストとの同定を行います。

```
/data/matsunaga/OPM/opm  j.xym  catalog.xym  j.match  param
```

第 1 引数と第 2 引数が同定をさせるリストの名前で、第 1 引数のリストの座標系にどのような変換を加えると第 2 引数のリストの座標系へできるかを調べます。ただし、現バージョン (Ver. 1.4) で加える変換はシフトと回転で 1 次元の変形です。第 3 引数 (j.match) は出力するファイル名、第 4 引数 (param) は OPM の計算に与えるパラメータ一覧です。上記のコマンドを実行すると、座標系の変換式と、どの天体が 2 つのリストで共通しているかというリストが、j.match というファイルに出力されるはずです。

catalog.xym は、原点とする座標を中心として、天球面上のどの座標に何等の星が写っているという情報をもっています。そのリストと同定することができ、座標の変換式も得られたので、JHK の画像がどのような座標系で投影されたものなのかわかったことになります。そこで、

```
/data/matsunaga/OPM/put_wcs_header j.fits j.match 17:47:11.13 -26:49:10.4
```

という OPM ソフトのコマンドを実行すると、座標系の情報（つまり WCS）を FITS ファイルに埋め込むことができます。JHK のそれぞれの場合について、このような作業を行って、3 枚の FITS 画像に WCS を入れてください。

撮像観測をするときには、ターゲットが視野内に入るように望遠鏡をポインティングしますが、特定の星がどのピクセルに入るかというところまで制御することはほとんどありません。したがって、3 つのフィルターでの画像をとったら、ある星が全く同じピクセル座標上にある保証はなく、実際ずれ

ていることが多いでしょう。しかし、(高速に天球上を動いていく天体でない限り) 天球座標系でみれば、同じ星は同じ座標にあるはずです。したがって、WCS を入れた画像であれば、その座標系を参照して画像を並べることで、同じ星を同じ場所に写すことができます。ds9 は、WCS で画像を並べて、さらに RGB カラー合成を行って表示することができます。

```
ds9  -zscale  -rgb  -blue j.fits  -green h.fits  -red k.fits
```

と実行して試してみてください。

6.3 課題

きれいな 3 色合成画像はできましたか。(同じような色の星しかなくて、あまりきれいな視野ではないですね。どうもすみません。。。) それでは、これらの一連の作業を自動で行うようなプログラムを作ってみましょう。

```
./kadai2 j.fits h.fits k.fits catalog.xym 17:47:11.13 -26:49:10.4 param
```

というふうに、3 つの画像の名前と、参照するカタログのファイル名、その基準座標 (RA, DEC)、OPM で利用するパラメータファイルの名前を引数として与えたら、3 つの画像に put_wcs_header を使って WCS を埋め込むところまで自動で行うようにしてください。途中、各画像の星の検出は daofind で行うこととして、そのとき使う fwhmpsf は 2.5 pixel の固定で構いませんが、sigma は imstat で計測してください。imstat を適当な名前の中間ファイルに出力させて、それを読み込むのが簡単でよいでしょう。

```
imstat j.fits  fields="stddev" format- nclip=2 lsigma=3. usigma=3. > imstat.tmp
```

C 言語のプログラミングに慣れている人は（そうでない人も)、(1) imstat で標準偏差を測るところ、(2) daofind で星を検出するところ、(3) OPM の同定を行うところ、(4) FITS に WCS を埋め込むところ、などはそれぞれ個別の関数としてまとめた方が、作成・デバック・管理のやりやすいプログラムになるでしょう。

本当は、fwhmpsf の測定や、catalog.xym と param の作成なども、自動で行いたいところです。興味のある人は、考えてみてください。そこまでいけば、たいていどんな画像がきてもすぐに WCS を入れてカラー合成ができるようになるだろうと思います。

7 おまけ：分割コンパイルと make の利用

少しまとまったプログラムを作ろうとするときには、ひとつの長いソースファイルを書くよりも、いくつかのソースファイルに分割するべきです。そ

の大きな理由は、複数のプログラムで何度も利用されるような関数は何度も書かずに一回だけ記述をして、それをいろいろなプログラムから呼ぶ方が良いからです。たとえば、上の例で `login.cl` をコピーしたり、`imarith` を実行するといった作業は、いろいろなプログラムで何度も行うことになります。もしも同じ作業をさせるための関数が複数のファイルに記述されていると、その関数をデバッグまたは改良したいと思ったときに全てのファイルにもれなく直さないといけなくなってしまいます。

また、複数のファイルを使ってツールを作成するときには、`make` を利用しましょう。`make` はいろいろなソースファイルを統合してコンパイルしたりするときに、複数のファイルの間の依存関係に従って必要最低限のコンパイルを実行して、目的を果たします。詳しい説明は省略しますが、コーディング・デバッグなどするときにも効率よく、混乱することなしに作業することができます。

筆者が C 言語の学習をする上で、以下の教科書が大変参考になりましたので紹介しておきます。

「C 言語 入門書の次に読む本」 坂井弘亮著（技術評論社、2003 年）