GammaFinder: a Java application to find galaxies in astronomical spectral line datacubes.

Peter James Boyce

Project Supervisor: Professor Antonia J. Jones

MSc in Computing

School of Computer Science, Cardiff University

16th September 2003

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed …….. (candidate)

Date ……..

STATEMENT 1

This dissertation is being submitted in partial fulfilment of the requirements for the degree of MSc.

Signed …….. (candidate)

Date ……..

STATEMENT 2

This dissertation is the result of my own independent work / investigation, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed …….. (candidate)

Date ……..

STATEMENT 3

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed …….. (candidate)

Date ……..

Abstract

This dissertation describes GammaFinder, a Java application which automatically searches for galaxies in the 3-dimensional datacubes produced by two major astronomical surveys: the HI Parkes All Sky Survey (HIPASS) and the HI Jodrell All Sky Survey (HIJASS). These surveys are mapping the entire sky, searching for galaxies by their emission in the 21-cm spectral line of neutral atomic hydrogen (HI). The core functionality of GammaFinder is based upon the Gamma Test, a near-neighbour data analysis routine that estimates the variance of the noise in continuous data. The Gamma Test is shown to also be a highly effective way of detecting discontinuities in otherwise continuous functions (in the presence of noise). This makes it an excellent way of selecting possible galaxies from HI spectra since a galaxy's HI emission line forms a discontinuity in an otherwise smoothly varying, but noisy, spectrum. GammaFinder searches for discontinuities in the 21-cm spectra of which a HIJASS/HIPASS datacube is comprised. It returns a list of possible sources along with a measurement of the strength of the discontinuity. GammaFinder is a GUI-based application, utilising Java Swing components. The results of the search are displayed on the GUI where they can be edited, annotated and saved to a text file. The spectra can also be examined on the GUI. The requirements specification, design and implementation of GammaFinder are described. In a preliminary evaluation, GammaFinder found at least an extra 50% more galaxies over those currently included in the published HIJASS catalogue. GammaFinder can select galaxies with significantly lower HI flux values than can previously used galaxy detection techniques. Hence, the use of GammaFinder could lead to a large increase in the number of galaxies which can be found in HIPASS and HIJASS data and significantly boost the scientific exploitation of these two major surveys.

Acknowledgments

I would like to thank my supervisor Professor Antonia J. Jones for suggesting applying the Gamma test to astronomy data and for her help and advice during the project.

Particular thanks are also due to Dr. Dafydd Evans of the School of Computing for his help and advice with the Gamma test and to Dr. Robert Minchin of the School of Physics and Astronomy who made many useful comments on the astronomy aspects of the project.

It has been a privilege and a pleasure to be involved in the HI Jodrell All Sky Survey and the HI Parkes All Sky Survey. I would like to thank my colleagues in both survey teams. Particular thanks are due to Virginia Kilborn, Christine Jordan, Mike Disney, Robert Minchin (again), Robert Lang, Diego Garcia and Marco Grossi, without whose superhuman efforts HIJASS would never had got off the ground.

I would also like to thank all my fellow students on the MSc computing course for sharing what has been an engaging, if exhausting, year.

Special thanks are due to my wife, Sarah, and our children, James and Caroline, for putting up with a largely absent husband / father over the summer holidays.

I gratefully acknowledge the financial support of Newport & Gwent Enterprise throughout the duration of the MSc course.

This research has made use of the NASA/IPAC Extragalactic Database (NED) which is operated by the Jet Propulsion Laboratory, Caltech, under agreement with the National Aeronautics and Space Administration.

This research has also made use of the Digitised Sky Survey, produced by the Space Telescope Science Institute under US Government Grant NAG W-2166.

This research makes extensive use of data from the HI Jodrell All Sky Survey (HIJASS) and the HI Parkes All Sky Survey (HIPASS). HIJASS is funded by the Particle Physics and Astronomy Research Council. HIPASS was funded by the Australia Telescope National Facility and the Australian Research Council.

Some aspects of the analysis presented in Chapter 3 of this report made use of the winGamma application, a ‘non-linear analysis workbench’ developed in the Department of Computer Science, Cardiff University (Durant 2001).

To open and read ‘FITS’ format data, the GammaFinder application makes use of the org.eso.fits Java package written by Preben Grosbol of the European Southern Observatory (available under the GNU General Public Licence).

Table of Contents

Table of Figures

1. Introduction

1.1 GammaFinder

This dissertation describes a new Java application, GammaFinder, which automatically searches for galaxies in the 3-dimensional datacubes produced by two major astronomical surveys: the HI Parkes All Sky Survey (HIPASS) and the HI Jodrell All Sky Survey (HIJASS). The core functionality of GammaFinder is based upon the Gamma Test (Jones et al. 2002a).

1.2 HIPASS and HIJASS

The author is a member of the teams undertaking these two surveys which, between them, will map the entire sky, searching for galaxies by their emission in the 21-cm line of neutral atomic hydrogen (HI).

HIPASS and HIJASS have created a huge dataset of 3-dimensional 'datacubes'. A datacube can be considered to be a collection of spectra. Each (x, y) pair represents a spatial position on the sky. The 'z'-axis represents the 21-cm spectrum at that position. Thus far, two methods have been employed to find galaxies within this data. Firstly, the datacubes have been visually inspected by astronomers. Secondly, automatic finding algorithms have been run over the datacubes. The automatic finding algorithms have proved to be no more effective than visual inspection. If an automatic finder could be devised which could find fainter galaxies (in HI flux) than visual inspection, this would have a big impact on the scientific exploitation of the two surveys, effectively making the surveys more sensitive. This was the main impetus for developing GammaFinder.

1.3 The Gamma Test

The Gamma Test (Stefansson et al. 1997) is a near-neighbour data analysis routine that estimates the variance of the noise in continuous data. Though developed as a tool to assist in the construction of data-derived models, it has also been applied to problems in control theory (Koncar 1997), feature selection (Chuzhanova et al. 1998; Durrant 2001), secure communications (de Oliveira 1999) and controlling chaotic systems (Tsui et al. 2002; Jones et al. 2002b).

The original aim in applying the Gamma test to HIPASS and HIJASS datacubes was to obtain an accurate estimate of the variance of the noise on the data. However, as described in this dissertation, the Gamma Test has also proved a highly effective way of detecting discontinuities in otherwise continuous functions. This makes it an excellent basis for an automatic finding algorithm.

1.4 Structure of the dissertation

Some background information about the nature and purpose of HIJASS and HIPASS is presented in Chapter 2, along with an explanation of the currently used techniques for detecting galaxies in HIJASS/HIPASS datacubes.

Chapter 3 contains a description of the Gamma Test and a discussion of the way in which the Gamma Test can be utilised to analyse spectral line data. This chapter concludes with three procedures which utilise the Gamma Test to analyse properties of a HIJASS / HIPASS datacube. These (a) measure the average noise in the datacube; (b) identify frequencies in the spectra which are affected by local interference; (c) select potential sources from the datacube. GammaFinder implements these three procedures.

The requirements specification and design of GammaFinder are described in Chapter 4. A description of the implementation of the application is given in Chapter 5.

Chapter 6 contains the results of an evaluation of GammaFinder, both in terms of its ability to detect real galaxies in HIJASS/HIPASS data and in terms of its computational efficiency.

Chapter 7 contains a summary of the main results and some suggestions for further work.

2. Background

2.1 Introduction

This chapter presents some introductory information about HI spectral line astronomy. The purpose of HI surveys is discussed in Section 2.2. Section 2.3 contains an overview of the techniques of spectral line observing and a description of the main features of an HI spectrum. The information about a galaxy obtainable from its HI emission line is described in Section 2.4. The HIPASS and HIJASS surveys are described in Section 2.5, whilst in Section 2.6 the characteristics of the datacubes produced by these surveys are discussed. The techniques and problems of detecting galaxies in HIPASS / HIJASS data are described in Section 2.7. Section 2.8 presents some concluding remarks.

2.2 The purpose of HI surveys

We require a complete and bias-free census of the local Universe to fully understand a range of astrophysical problems (e.g. galaxy formation and evolution, the mass density of the Universe). However, our knowledge of the contents of the Universe is primarily based on surveys made at optical wavelengths. Such surveys identify galaxies from the light emitted by the stars they contain. Hence, the catalogues of galaxies compiled from these surveys will preferentially contain those morphological types of galaxies that are optically bright. There are known to be classes of extragalactic objects (e.g. dwarf galaxies, low surface brightness galaxies) that are relatively faint in optical light. These types of object will tend to be under-represented or missed altogether from optically-selected catalogues.

Besides their stellar content, many galaxies also contain large amounts of neutral atomic hydrogen (HI). This provides an alternative way of searching for galaxies since neutral atomic hydrogen has a spectral emission line (at a wavelength of 21-cm) that can be detected by radio telescopes. Searches for galaxies based on their HI content may provide a very different perspective on galaxy populations compared to optical surveys since there is no reason to assume that a galaxy's HI content will be directly correlated with its optical brightness.

HIPASS and HIJASS are the first HI surveys to search for galaxies over a large area of sky. These surveys will be described in Section 2.5.

2.3 The 21-cm spectral line of neutral atomic hydrogen

The 21-cm spectral line of HI results from a magnetic dipole transition between the two levels that characterise the ground state of the hydrogen atom. Depending on whether the electronic and nuclear spin vectors are parallel or anti-parallel, a slightly higher (triplet) or lower (singlet) hyperfine level is obtained. Spontaneous transitions from the triplet to the singlet level occur with a transition probability of 2.868×10^{-15} s^{-1} (see e.g. Rohlfs 1986). The vast reservoirs of hydrogen in many galaxies means that sufficient numbers of transitions are happening per second for the HI line to be detectable.

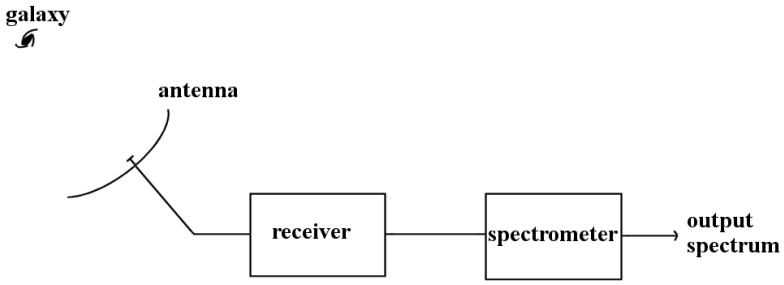


Figure 2.1. The main stages of observing a 21-cm spectrum.

Figure 2.1 shows a simplified schematic diagram of the process of observing a 21-cm emission line from a galaxy. A radio telescope antenna collects the electromagnetic radiation arriving from the sky in the direction to which it is pointed. The receiver is tuned (by filters) to select radiation from a relatively narrow frequency 'bandpass' around the rest frequency of the HI line. The radiation within this bandpass, collected by the antenna, is converted into an AC signal by the receiver. This signal is then passed to the spectrometer. The spectrometer samples the received signal at a number of discrete, equally spaced frequencies within the bandpass. The output from a spectrometer is a 'spectrum', i.e. the detected power of the received electromagnetic radiation at a set of discrete frequencies (known as 'channels') within the bandpass.

An example of the resulting HI spectrum for a galaxy is shown in Figure 2.2. The main features of this spectrum are described below.

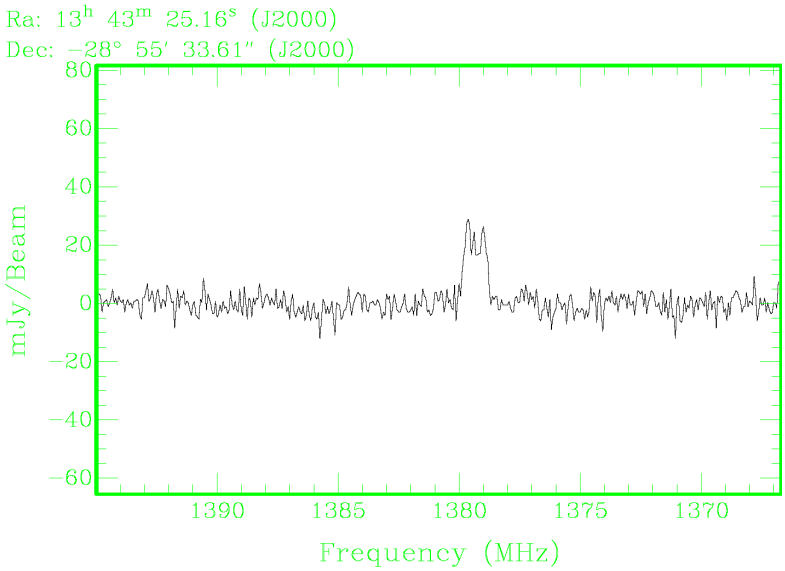


Figure 2.2. A HI spectrum resulting from the observing process shown in Figure 2.1.

2.3.1 The axes on the spectrum

A spectrum is a plot of the intensity of observed radiation (y-axis) against the frequency of that radiation (x-axis). Note that this is really a plot of discrete (x,y) pairs since the output from the spectrometer measures the intensity in a set of discrete frequency 'channels'. It is conventional to connect each adjacent (x,y) pair (i.e. adjacent in x) together to aid the interpretation of the data.

The y-axis is labelled in a unit called the 'Janksy' ('Jy'). The units W m^{-2} Hz^{-1} describe the power arriving per square metre of the antenna surface per unit frequency width. However, such small amounts of radiation are received from astronomical sources that the Jy is a more convenient unit. By definition 1 Jy = 10^{-26} W m^{-2} Hz^{-1} .

The 'Ra' and 'Dec' labels refer to the 'Right Ascension' and 'Declination' of the position on the sky to which the telescope was pointing. Right Ascension and Declination describe the angular coordinates of a position on the sky. They are analogous to the longitude and latitude coordinate system on the Earth (see e.g. Duffet-Smith 1988).

2.3.2. The detected HI emission line

The prominent peak in the spectrum in Figure 2.2 is due to HI emission from a galaxy. The galaxy is emitting over a range of frequencies and, hence, has been detected in

several frequency channels. If all the gas in the galaxy were stationary relative to the centre of that galaxy, then we would expect a very narrow emission line. However, in many galaxies, the gas is undergoing some kind of rotation about the centre of the galaxy. Some gas is moving towards us and some away from us (relative to the general motion of the galaxy). Due to the Doppler Effect (see Section 2.4.1), the HI emission from the gas moving towards us has its frequency shifted to higher frequencies and the gas away from us to lower frequencies. Hence, the observed HI profile is broadened by the rotation. This issue is discussed further in Section 2.4.4.

2.3.3 The noise on the spectrum

Outside the frequency range occupied by the galaxy, we expect zero radiation to be detected. However, the measured spectrum clearly deviates above and below the zero intensity level. This deviation from the expected function is known as 'noise'.

During an observation of an HI spectral line, the radiation associated with the emission line is generally a tiny fraction of the total power received by the antenna. At each frequency channel within the band, a large amount of thermal radiation is also detected. This comes mostly from the antenna and receiving system. During the data reduction process an attempt is made to remove this thermal radiation from the spectrum, leaving just the residual radiation from the galaxy. However, this cannot be done perfectly since the noise associated with the thermal radiation cannot be removed. This noise results from the fundamental uncertainty in the number of photons emitted by a thermal source in a given period of time. The probability function of the number of photons emitted in a given time interval forms a Poisson distribution about the mean number of photons emitted in that time.

The following equation describes the root-mean-squared (rms) noise per channel on a HI spectrum:

$$\sigma_{noise} = \frac{Tsys}{\sqrt{BW * t}} \qquad (2.1)$$

where Tsys is the 'system temperature' (a measure of the total power in the bandpass), BW is the width in frequency of a single channel and t is the integration (observing)

time (see e.g. Rohlfs 1986). The Poissonian nature of the noise means that the noise on the final spectrum only decreases as $\sqrt{t}$. This places a strong constraint on the sensitivity which can be obtained by increasing integration time: to reduce the noise on a spectrum by a factor of 2, we have to increase the integration time by a factor of 4.

2.4 Information in an HI Spectrum

An HI spectrum can provide a great deal of information about the galaxy detected. This information is summarised in this section.

2.4.1 The distance to the galaxy.

All external galaxies with measured HI emission lines are found to have central frequencies of the 21-cm line shifted from the rest frequency of the line. The standard interpretation is that this is due to the Doppler effect, i.e. it occurs because the galaxy is moving either towards us (in a few cases) or (in most cases) away from us. The 'redshift', z, of a galaxy is defined as

$$z = \frac{\lambda obs - \lambda rest}{\lambda rest} \qquad (2.2)$$

where ②$_{obs}$ is the observed wavelength of the 21-cm line and ②$_{rest}$ is the rest wavelength of the line. We can calculate the recession velocity, v, causing this Doppler effect via the equation

$$v = cz \qquad (2.3)$$

where c is the velocity of light (see e.g. Burke & Graham-Smith 2002).

The Universe is believed to be expanding such that, on a large scale, all points within it are expanding away from all other points. The speed of this expansion is proportional to the distance between the two points. Hence, the further away from each other two galaxies are, the faster they are receding from each other. This is the generally held explanation for the fact that nearly all galaxies have redshifts that imply they are receding from our own.

It is standard practice to label the frequency axis of HI spectra in units of cz (km s^{-1}) rather than frequency (MHz). This gives a useful indicator of the distance to a galaxy. According to Hubble's law, the velocity of recession of a galaxy, v, is related to its distant, d (in Mpc), via the equation

$$v = cz = H_o\, d \qquad (2.4)$$

where H_o is Hubble's constant. Current measurements of H_o place it in the region 60 – 80 km s^{-1} Mpc^{-1} (see e.g. Branch 1998). It is standard practice to assume a value of H_o=75 km s^{-1} Mpc^{-1}. For example, if we have a galaxy with a cz = 6000 km s^{-1}, then we can estimate its distance to be about 80 Mpc.

2.4.2 Derived parameters from an HI profile

Figure 2.3 shows a spectrum from the spiral galaxy UGC 02855. Note that the frequency axis is now labelled in units of Velocity (i.e. cz). This spectrum has been annotated to illustrate several parameters which can be derived from a spectrum.

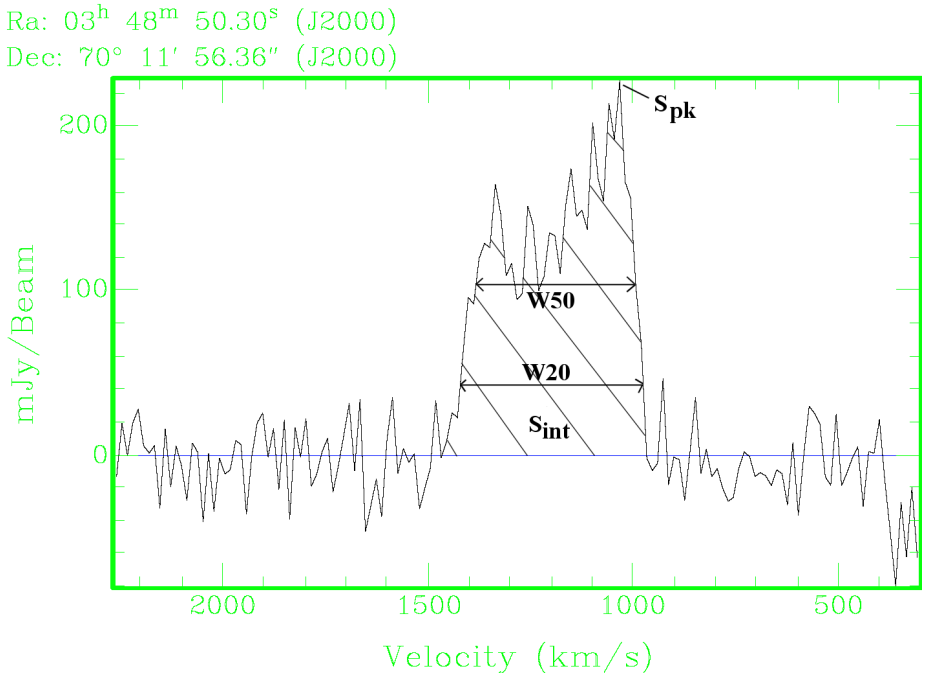


Figure 2.3. An HI spectrum of UGC 02855, annotated to show the parameters measured from an HI spectrum.

The *Peak Flux*, S_{pk}, (measured in Jy) is the largest measured flux value in any of the channels in which the galaxy is detected.

The *Integrated Flux*, S_{int}, (measured in Jy km s^{-1}) is the sum of the fluxes in all the channels in which the galaxy is detected.

The *Signal-To-Noise Ratio* of a detection refers to the flux value of the detection compared to the rms noise in the data. If an emission line has a S_{pk} = 60 mJy and the rms noise per channel in the spectrum is 15 mJy, then this detection has a 'signal-to-noise ratio' of 4.

The *velocity-width* of a profile is the frequency range over which the line is observed, expressed in cz units. This is commonly expressed as the velocity-width at half the S_{pk} value (W50) or at 20% of the S_{pk} value (W20).

2.4.3 The mass of HI in a galaxy.

This can be found using the equation

$$M_{HI} = 2.356 \times 10^5 \, d^2 \, S_{int} \qquad (2.5)$$

where M_{HI} is measured in M_o (1 M_o = 2 x 10^{30} kg), d is measured in Mpc and S_{int} in Jy km s^{-1} (see e.g. Rohlfs 1986).

2.4.4 The internal dynamics of the gas in the galaxy

The velocity-width and profile shape of the HI emission line from a galaxy depend on both the intrinsic gas motions within the galaxy and the galaxy's inclination to the line of sight. This is illustrated in Figure 2.4 which shows images of 3 galaxies along with their HI emission line profiles. The images were taken from the Digital Sky Survey (DSS). The DSS is a set of digitised photographic survey plates covering the whole sky. The DSS data from any position in the sky can be downloaded from http://archive.stsci.edu/cgi-bin/dss_form. This forms an invaluable resource to the community and will be utilised further in Chapter 6.

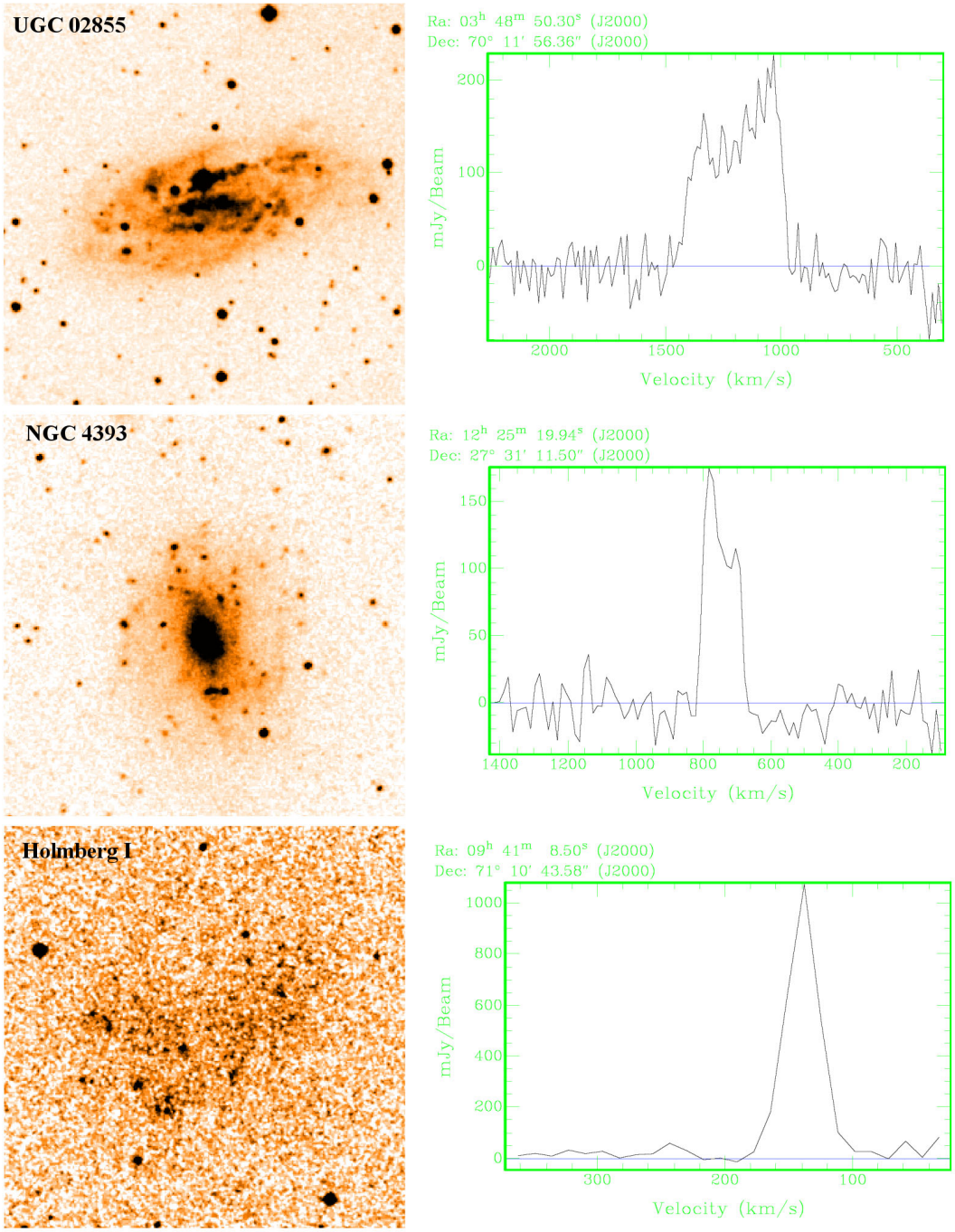


Figure 2.4. Images from the Digital Sky Survey of three galaxies along with their HI spectra (from HIJASS).

UGC 02855 (top panel) is a spiral galaxy, which is significantly inclined to the line of sight. This has a broad, double-peaked emission line profile with W20 $\not<$ 450 km s^{-1}. This 'double-peaked' profile is typical of that observed from spiral galaxies which are inclined to the line of sight. The HI in a spiral galaxy is usually rotating about the centre of the galaxy. However, it is only that component of this motion along the line of sight which will cause a Doppler shift in the frequency of the observed radiation. If a

galaxy is inclined to the line of sight then that component of the HI moving towards us will be blueshifted (relative to the general redshift of the whole galaxy) and that component moving away from us will be redshifted. Hence, we see two peaks in the emission, each displaced from the canonical redshift of the galaxy.

NGC 4393 (middle panel) is also a spiral galaxy. However, its spectrum has W20 $\not<$ 130 km s^{-1}, much narrower than that of UGC 02855, although we still a double-peaked profile. As the image makes clear, NGC 4393 is much more face-on than UGC 02855. This makes the profile of NGC 4393 much narrower since the gas is moving at much smaller projected velocities along the line of sight (relative to the recession velocity of the galaxy).

Holmberg I (bottom panel) is a dwarf galaxy in the nearby M81 Group. This has no disk or spiral structure and only turbulent motions in its internal gas. Hence, compared to the spiral galaxies, it has a very narrow velocity-width (W20 $\not<$ 60 km s^{-1}).

2.5 HIPASS and HIJASS

HIPASS was commenced in February 1997 and concluded in March 2002. HIPASS surveyed the entire southern sky, as well as the northern sky up to Declination = $+25^{o}$. A cz range of $-1280 < cz < 12700$ km s^{-1} was surveyed. This survey was made feasible by the fitting of a 13-beam 'multibeam' receiving system to the Parkes Radio Telescope in New South Wales (see Figure 2.5). This enables the telescope to observe 13 closely-spaced positions on the sky simultaneously. A full explanation of the survey technique and data reduction process can be found in Barnes et al. (2001).

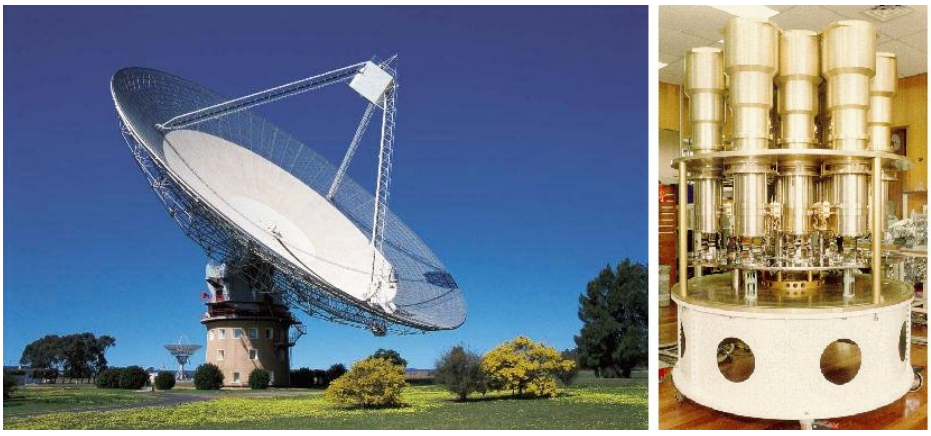


Figure 2.5. The Parkes Radio Telescope and the 13-beam multibeam receiver (pictures courtesy of CSIRO).

HIJASS is surveying the northern sky above Declination = +22^{o}, to similar sensitivity to HIPASS, using the Multibeam 4-beam receiver mounted on the 76-m Lovell Telescope at the Jodrell Bank Observatory, Cheshire (see Figure 2.6). HIJASS was begun in 2000.

Figure 2.6. The 76m Lovell telescope at Jodrell Bank.

As expected, HIPASS and HIJASS have added significantly to the census of the local extragalactic population. Highlights from HIPASS include the discovery of 10 new members to the nearby Cen A group of galaxies (Banks et al. 1999) and the most sensitive determination of the HI mass function yet published (Zwaan et al. 2003a). HIJASS highlights include the discovery of a new member of the nearby M81 group of galaxies (Boyce et al. 2001). A catalogue of confirmed sources from the HIPASS survey, 'HICAT', has been submitted for publication (Meyer et al. 2003, Zwaan et al. 2003b). Lang et al. (2003) have published a catalogue of confirmed sources from the first part of the HIJASS survey.

2.6 Characteristics of a HIPASS/HIJASS datacube.

The final result of the survey and data reduction process with both HIPASS and HIJASS (see Barnes et al. 2001) is a 3-dimensional 'datacube'. The characteristics of these datacubes are described in this section.

The datacube can be considered to be a collection of spectra. Each (x,y) coordinate pair represents a spatial position on the sky (measured in Right Ascension and Declination).

The ‘z’ axes represents the measured 21-cm spectrum at that position. HIJASS/HIPASS datacubes are stored as 3-dimensional ‘FITS’ (Flexible Image Transfer System) images. FITS is an image format designed to enable the transfer of astronomical images between different software packages.

Pixels in multi-dimensional astronomical images are generally referred to as ‘voxels’. Each voxel along the x and y axes of a datacube represents 4 arcmin on the sky. A typical datacube has 150 voxels in the Right Ascension (x) dimension and 150 in the Declination (y) dimension (equivalent to about 8^o x 8^o area of sky). The z axis has 1024 voxels corresponding to the 1024 channels of the spectra produced by the multibeam spectrometers. The frequency width of each z-channel is 13.4 km s^{-1}.

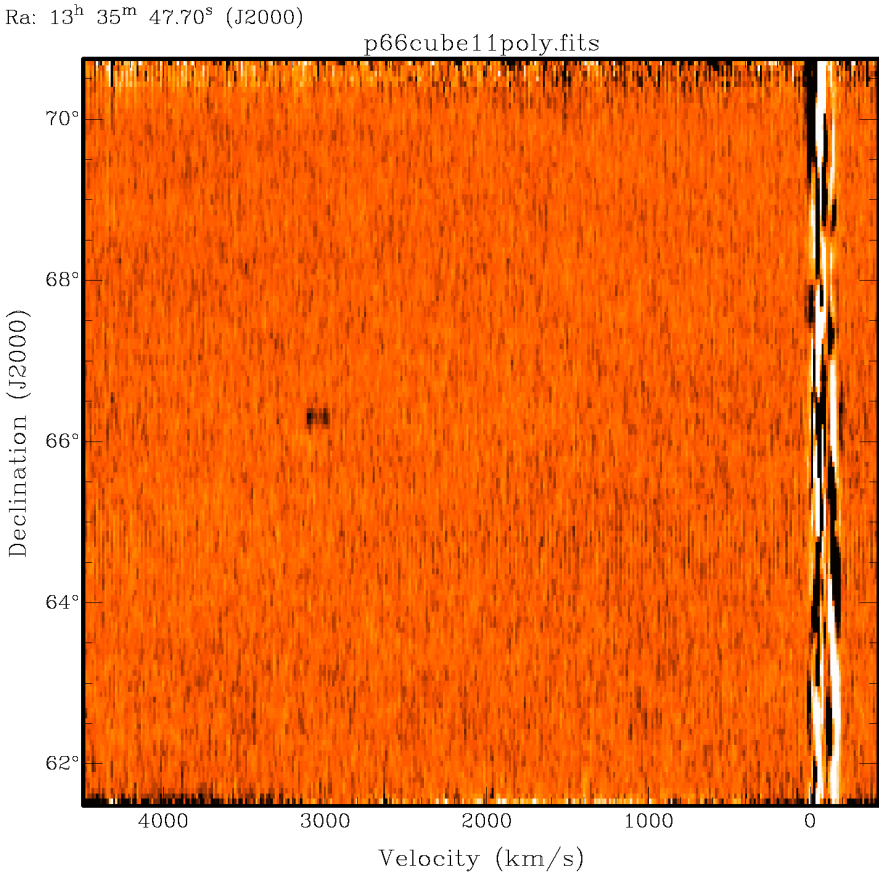


Figure 2.7. A ‘slice’ though a HIJASS datacube showing Declination against Velocity (cz) for a single Right Ascension plane.

Reducing the data into datacube format makes the process of analysing the data easier than if the final dataset simply consisted of a set of spectra. One can still view the spectrum at any (x,y) position but, in addition, the data can be viewed as a set of 2-

dimension ‘slices’ through the datacube. Figure 2.7 is a plot of Declination against cz for a single Right Ascension (‘x’) plane. The source HIJASS J1335+6615 can be seen at Declination=66.25^{o}, cz = 3050 km s^{-1}. This has been identified as the spiral galaxy UGC08604. There is also a large signal close to 0 km/s at all Declinations. This is due to HI emission from our own Galaxy (around cz = 0 km s^{-1}). The standard data reduction process does not fully remove this radiation from the final spectra and its residue is seen on the plot.

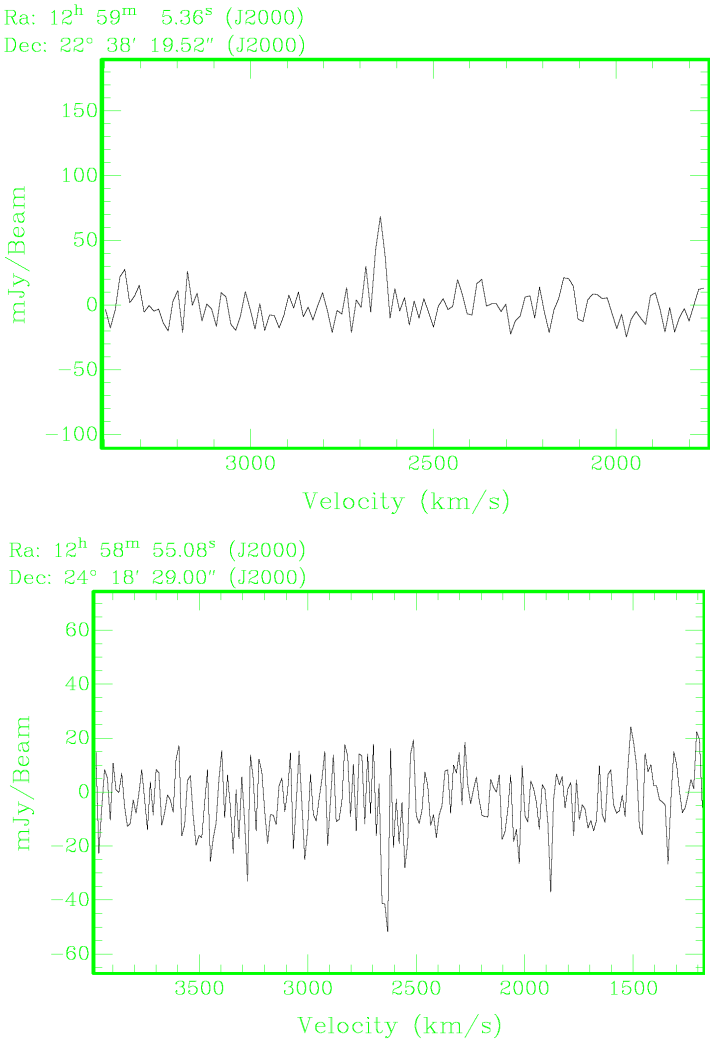


Figure 2.8. Spectra from different (x,y) positions of the same HIJASS datacube showing the differing effects of narrow-line RFI emission at cz = 2640 km s^{-1}.

There are two main problems which beset HIPASS and HIJASS data. Firstly, several z-channels in a HIPASS / HIJASS datacube can be affected by locally-based radio frequency interference (RFI). Figure 2.8 shows two spectra from different (x,y) positions in the same HIJASS datacube. This datacube suffers from RFI at cz = 2640

km s^{-1}. In the top panel the RFI is producing a peak in the spectrum which could easily be mistaken for a narrow velocity-width galaxy. In the bottom panel, however, the RFI is producing a negative peak in the spectrum. The intensity of the peaks and troughs produced by the interference vary throughout the datacube.

The second problem, known as "baseline ripple" is illustrated in Figure 2.9. Instead of a flat spectrum (with noise), this spectrum has a baseline with a periodic rise and fall. This results when standing waves are formed between the focus box and dish of the telescope. Figure 2.9 represents one of the worst examples of baseline ripple. Many cubes are hardly affected.

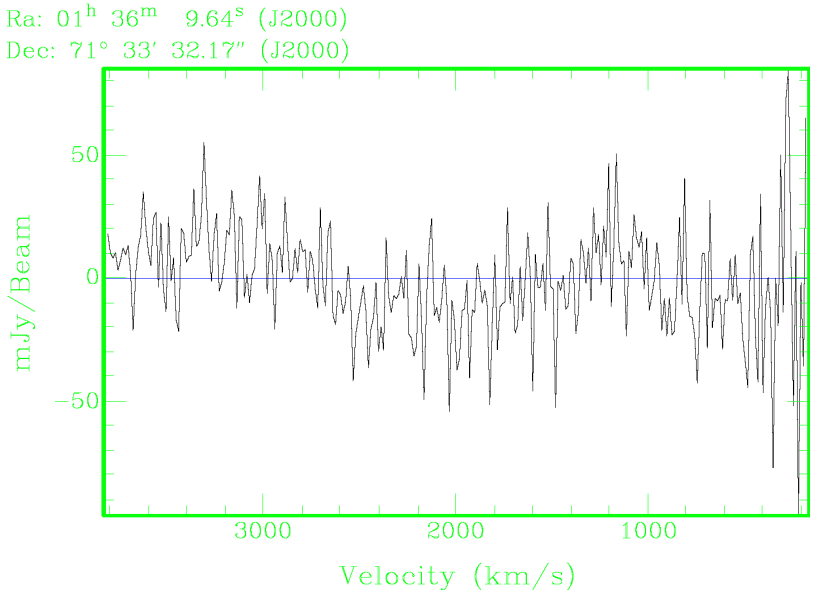


Figure 2.9. A HIJASS spectrum which suffers from 'baseline ripple'.

2.7 Detecting galaxies in HIPASS / HIJASS datacubes

The process of finding galaxies in HIPASS / HIJASS datacubes is outlined in this section. This includes a discussion of the contribution which automatic finding algorithms can make to this process.

The basic process of finding galaxies in a HIPASS / HIJASS datacube involves an astronomer undertaking a 'visual inspection' of the datacube by examining each Declination-Velocity slice (see Figure 2.7) and also the set of Declination-Right Ascension slices for each cz channel. The astronomer is essentially looking for places where the signal is significantly above the noise level in several contiguous z-channels.

Figures 2.8 and 2.9 illustrate 2 problems which the astronomer has to overcome. Firstly, local RFI can easily be confused with a weak narrow velocity-width galaxy. Secondly, the effects of baseline ripple can act so as to mask real galaxies in the data, but extenuate random fluctuations in the noise and make them appear like real galaxies.

One advantage an astronomer has is that a real galaxy will be detected in several adjacent (x, y) positions. This is because the beam widths of the telescope (14 arcmin for Parkes and 10 arcmin for the Lovell) are broader than the spatial (i.e. Right Ascension and Declination) voxels (4 arcmin) so even a source not resolved by the telescope (i.e. having a smaller projected size on the sky than the telescope beam) occupies around 3 x 3 spatial voxels. Nearly all sources are actually unresolved by the beam.

The end result of such a visual inspection is a list of identified sources. These are graded either as 'definite' (i.e. those which the astronomer considers undoubtedly real) or 'possible' detections (generally sources with lower peak or integrated flux values or which the astronomer suspects may be the result of RFI or baseline ripple). The 'possible' sources have to re-observed in order to determine if they are real or not. This is done by pointing a radio telescope directly at each source, using a smaller bandpass (to improve velocity resolution) but longer integration time (to improve signal-to-noise) than in the survey data. This process, known as 'narrow-band follow-up' is an essential part of creating a catalogue of confirmed sources from HIPASS / HIJASS data.

Obviously, the astronomer wants to find the faintest sources possible within the data. However, the fainter an astronomer tries to delve into the data, the more likely it is that random configurations of the noise will be misidentified as galaxies. The human eye tends to select galaxies on the basis of their peak flux, S_{pk}. Hence, the sample produced by a visual search will be "peak-flux limited", i.e. all galaxies with S_{pk} above some peak flux limit will be included in the sample. This peak flux limit (the 'completeness limit') is usually expressed in terms of the rms noise per channel in the data, $⑨_{noise}$. For example, if the rms noise in the data is 15 mJy per channel and the selection process finds all galaxies with $S_{pk} > 75$ mJy, then we can say that the sample has a 'completeness limit' of $5⑨_{noise}$. Some galaxies will still be found with S_{pk} below the

completeness limit, but at S_{pk} values below this limit, the human eye is not finding all the galaxies in the data. Below some S_{pk} value no galaxies at all are detected. This is known as the 'detection limit'.

The sample of galaxies selected by visual inspection is not complete above any integrated-flux limit. For example, 2 galaxies may the same integrated flux, S_{int}, but one of them have a much broader velocity-width and hence lower S_{pk}. One of these galaxies may be included in a visually selected sample and the other not. Ideally, we would like to create a sample that included all galaxies above a given HI mass (at a given distance). A galaxy's HI mass is related to its integrated flux (Equation 2.5). Hence, we would require an integrated flux limited sample (i.e. where all galaxies above a given S_{int} are included) rather than a peak flux limited sample.

Whilst an astronomer must have overall control of the process of finding galaxies in a HIPASS / HIJASS datacube, there is significant scope for an automatic finding algorithm to assist in this process. To be of value an automatic finding algorithm must be capable of fulfilling one or both of two functions:

1. Find galaxies that are missed by a visual inspection. If an algorithm could detect sources which are not easily seen in a visual inspection, then it would be highly valuable.
2. Save an astronomer the task of visual inspection at all. To be useful in such a context, the algorithm must be capable of finding all those sources that would have been found in a visual inspection.

Three automatic finding algorithms have thus far been written. MultiFind (Kilborn 2001) and PolyFind (Minchin 2001) are both based around the idea of searching each spectrum in a datacube for peaks in the emission which are greater than a specified peak flux density, expressed in terms of an estimated value for the rms noise in the spectrum. MultiFind (Meyer et al. 2003) cross-correlates spectra with top-hat profiles of various scales: a feature is detected if it rises above a threshold value (determined by the variance of the noise) in the convolved spectrum.

The published HIJASS catalogue (Lang et al. 2003) was created by astronomers visually searching the datacubes and then subjecting 'possible' sources to narrow-band

follow-up to confirm or reject them as galaxies. The PolyFind algorithm was also run over every datacube. PolyFind actually missed a significant fraction of the sources found by a visual search, but did identify a small number of sources not found by the visual search. However, PolyFind also returned large numbers of spurious sources.

The HIPASS catalogue (HICAT: Meyer et al. 2003) made use of MultiFind and TopHat. A main candidate list was created by running each of these finders over all the datacubes. All the candidates were then examined visually by astronomers. Both finders returned a large number of spurious candidates which had to be rejected at this stage. Those candidates graded as 'possible' by the astromomers were subjected to narrow-band follow-up to confirm or reject them as galaxies.

The resulting HIJASS and HIPASS catalogues are both peak flux limited. The HIJASS catalogue has a completeness limit at $S_{pk} \not<5⑨_{noise}$ and has a detection limit at $S_{pk} \not<3⑨_{noise}$ (Lang et al. 2003). HICAT has similar completeness and detection limits (Zwaan et al. 2003b), suggesting that MultiFind and TopHat do not find galaxies to significantly fainter flux limits than does a visual search.

Clearly, an automatic finding algorithm which could find galaxies to significantly lower limiting HI flux values than can a visual search would be extremely valuable. None of the present finders is able to.

2.8 Concluding remarks

This chapter has provided some background information necessary to understand the need for, and the role of, an application which can automatically detect possible sources from HIPASS / HIJASS data. In Chapter 3, the application of the Gamma Test to this problem is investigated.

3. Applying the Gamma Test to HIPASS and HIJASS Data

3.1 Introduction

The possible applications of the Gamma Test to the analysis of HIPASS / HIJASS datacubes are discussed in this chapter. The Gamma Test itself is described in Section 3.2. Its utility for measuring the noise in HI spectra and for detecting discontinuities in the spectra is considered in Section 3.3. The ability of the Gamma Test to find z-channels suffering from RFI is considered in Section 3.4. The ability of the Gamma Test to select possible galaxies from HIPASS / HIJASS data is described in Section 3.5. Three procedures which utilize the Gamma Test to analyse properties of HIPASS / HIJASS datacubes are presented in Section 3.6. These (a) measure the average noise; (b) find z-channels affected by RFI; (c) select potential sources. Some concluding remarks are made in Section 3.7.

3.2 The Gamma Test

A full description of the Gamma Test can be found in Jones et al. (2002a) and a proof in Evans and Jones (2002). A brief description is given in this section.

Suppose we are given a set of input-output data of the form $\{ (\boldsymbol{x}_i, y_i) \mid 1 \le i \le M \}$ where we can think of the vector $\boldsymbol{x} \in \mathbb{R}^m$ as the input (confined to some closed bounded set $C \subset \mathbb{R}^m$) and the corresponding scalar $y \in \mathbb{R}$ as the output, i.e. we assume that, in some sense, y is determined by $\boldsymbol{x}$. Suppose that the underlying relationship is of the form

$$y = f(\boldsymbol{x}) + r \tag{3.1}$$

where f is a smooth function representing the system and r is a random variable representing noise. The expected value of the noise variable r may be assumed to be zero, since any constant bias can be absorbed into the unknown function f. Despite the fact that f is unknown, subject to the condition that f has bounded first and second partial derivatives over the input space C, the Gamma Test provides an estimate for the noise variance, $Var(r)$. This estimate is known as the *Gamma Statistic*, denoted by Γ.

Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be any two points of the input space C. The corresponding outputs will be

$$y = f(\boldsymbol{x}) + r \tag{3.2}$$

and

$$y' = f(\boldsymbol{x}') + r' \tag{3.3}$$

Hence:

$$½(y' - y)^2 = ½((r' - r) + (f(\boldsymbol{x}') - f(\boldsymbol{x})))^2 \tag{3.4}$$

The continuity of f implies that:

$$| f(\boldsymbol{x}') - f(\boldsymbol{x}) | \to 0 \quad \text{as} \quad | \boldsymbol{x}' - \boldsymbol{x} | \to 0 \tag{3.5}$$

Therefore

$$½(y' - y)^2 \to ½(r' - r)^2 \quad \text{as} \quad | \boldsymbol{x}' - \boldsymbol{x} | \to 0 \tag{3.6}$$

Taking the expectation of both sides of Equation 3.6

$$\varepsilon (½ (y' - y)^2) \to Var(r) \quad \text{as} \quad | \boldsymbol{x}' - \boldsymbol{x} | \to 0 \tag{3.7}$$

However, given any finite data set we cannot evaluate this limit directly. Instead the Gamma Test computes an estimate of $Var(r)$ by comparing the relationship between pairs of *nearest neighbour* input values and the corresponding output values.

For any ordered sets of points T= $\{\boldsymbol{x}_1,..., \boldsymbol{x}_M\}$, the q-th nearest neighbour of any point $\boldsymbol{x}_i$ is uniquely defined as follows. The first nearest neighbour of $\boldsymbol{x}_i$ is that point $\boldsymbol{x}_{j1} \in$ T $\backslash\{\boldsymbol{x}_i\}$ having minimal distance from $\boldsymbol{x}_i$ and minimal index $j1$. The second nearest neighbour of $\boldsymbol{x}_i$ is that point $\boldsymbol{x}_{j2} \in$ T $\backslash\{\boldsymbol{x}_i , \boldsymbol{x}_{j1}\}$ having minimal distance from $\boldsymbol{x}_i$ and minimal index $j2$. In general, the q-th nearest neighbour of $\boldsymbol{x}_i$ is that point $\boldsymbol{x}_{jq} \in$ T $\backslash\{\boldsymbol{x}_i , \boldsymbol{x}_{j1, ..., } \boldsymbol{x}_{jq-1}\}$ having minimal distance from $\boldsymbol{x}_i$ and minimal index jq. Ordering equidistant points (e.g. the channels in a spectrum) by their indices in this way means that every point $\boldsymbol{x}_i$ has a uniquely defined q-th nearest neighbour.

Let $\boldsymbol{x}_{N[i,q]}$ denote the q-th nearest neighbour of the point $\boldsymbol{x}_i$ in the set $\{\boldsymbol{x}_1,..., \boldsymbol{x}_M\}$ and let $y_{N[i,q]}$ denote the associated output value. We can compute a sequence of estimates for $\varepsilon (½ (y' - y)^2)$ by the sample means:

$$\gamma_M(q) = \frac{1}{2M} \sum_{i=1}^{M} |y_{N[i,q]} - y_i|^2 \qquad (3.14)$$

where $|.|$ denotes the Euclidean distance. In each case, we can obtain an indication of the "error" by computing the mean squared distance between the q-th nearest neighbours, defined by:

$$\delta_M(q) = \frac{1}{M} \sum_{i=1}^{M} |\boldsymbol{x}_{N[i,q]} - \boldsymbol{x}_i|^2 \qquad (3.15)$$

It can be shown (Evans and Jones 2002) that there exists some constant A such that

$$\gamma_M(q) \approx Var(r) + A\,\delta_M(q) + O(\delta_M(q)) \qquad as \qquad \delta_M(q) \to 0 \qquad (3.16)$$

The Gamma Test is, therefore, conducted by computing the pairs $(\gamma_M(q),\delta_M(q))$ for $1 \leq q \leq p$ and performing linear regression on these points. This process is illustrated in Figure 3.1. The Gamma Statistic Γ is the constant term of this regression line.

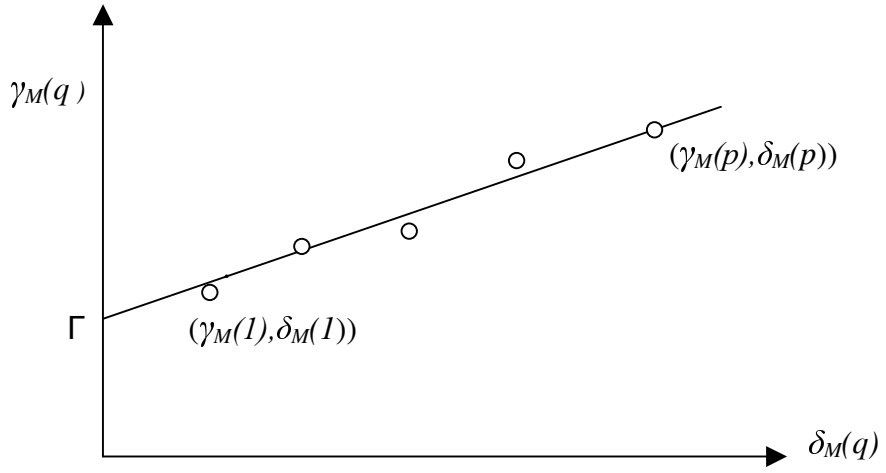


Figure 3.1. The Gamma Test regression plot.

3.3 Applying the Gamma Test to HIPASS / HIJASS spectra

The initial impetus for applying the Gamma Test to HI spectra was to test whether it provided an accurate way of determining the variance of the noise on the spectra. This section describes the results of tests of its utility for this purpose. However, as also illustrated in this section, it was found that the Gamma Test is also an effective way of detecting discontinuities in a spectrum.

A HI spectrum, at a given (RA, Dec) position, in a HIJASS / HIPASS datacube consists of a set of z-channels (each representing a particular cz). Each z-channel has a corresponding intensity value (in mJy). We can regard the z-channel numbers as the input data ($\boldsymbol{x}$) and the intensity values as the output values (y). In this case, each input $\boldsymbol{x}$ vector consists of only one component, the z-channel number. Assuming that the output is related to the input via an underlying relationship of the form

$$y = f(\boldsymbol{x}) + r \tag{3.18}$$

where f is a smooth function and r is a stochastic variable which represents noise, then we should be able to apply the Gamma Test to the data and, hence, be able to estimate the variance of the noise on the spectrum. The conventional way to express the noise level in an HI spectrum is as the rms noise per channel. Hence, the input values ($\boldsymbol{x}$) must be expressed in units of z-channel number (not in MHz or km s^{-1}) and the output values (y) in units of Jy.

Figure 3.2(a) shows an artificial spectrum generated using the Mathematica software package. The spectrum was generated with a noise variance of 0.000225 Jy, equivalent to an rms noise of 15 mJy per channel (typical of HIPASS / HIJASS spectra). The spectrum has 300 channels. A Gamma Test was run on this spectrum using the *winGamma* application (Durant 2001). A returned value of $\Gamma = 0.000233$ was obtained when running the Gamma Test on all 300 channels (using the default maximum number of nearest neighbours, p=10). The rms noise obtained by taking the square root of this is 15.3 mJy, very close to the actual value.

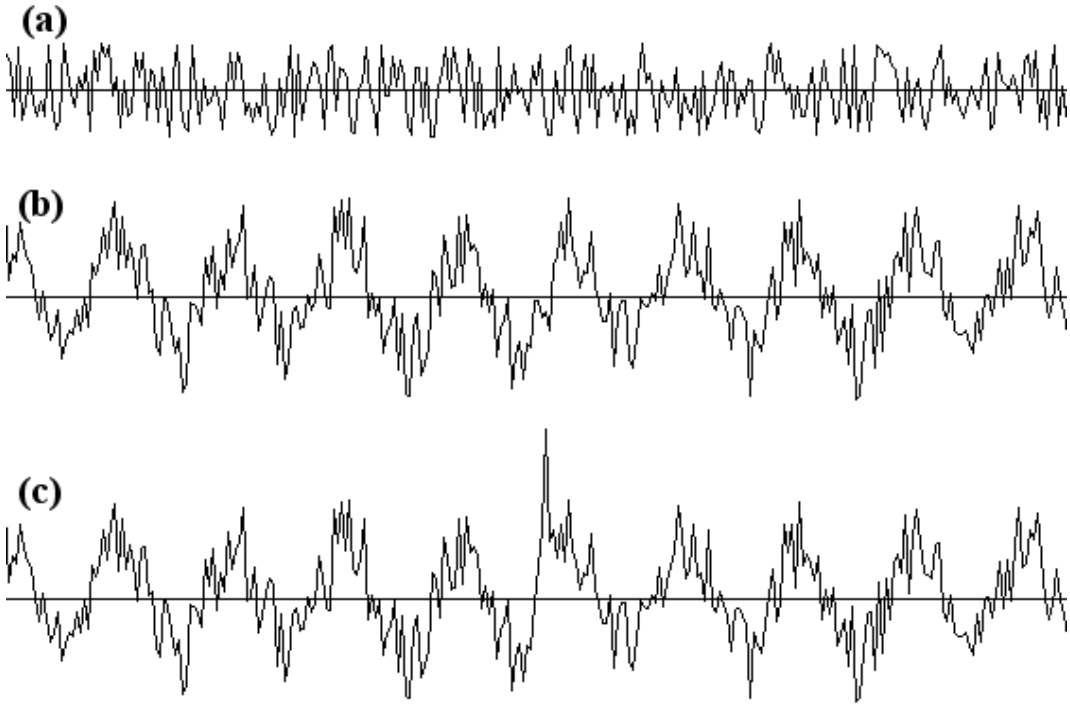


Figure 3.2(a) an artificial spectrum with an rms noise per channel of 15 mJy; **(b)** as in (a) but with a baseline ripple with period = 150 channels and peak to peak intensity !30 mJy. **(c)** as in (b) but containing a narrow emission line with W20 = 60 km s^{-1} and S_{pk} = 100 mJy.

Figure 3.2(b) shows an artificial spectrum with the same rms noise as in Figure 3.2(a). However, in this example a simulated baseline ripple with a period of 150 channels has been added, typical of that seen in HIPASS / HIJASS datacubes. The peak to peak intensity variation of this ripple is !30 mJy, among the worse values found in real HIPASS / HIJASS data. The Gamma Test was also run on this spectrum. A returned value of Γ = 0.000248 was obtained, equivalent to an rms noise of 15.7 mJy per channel. Because the baseline ripple can be described by a smooth function, the Gamma Test still returns a reasonably accurate estimate of the noise.

Figure 3.2(c) shows the same spectrum as in Figure 3.2(b) except that a simulated HI emission line has been added with W20=60 km s^{-1} and S_{pk}=100 mJy. The presence of the emission line on a spectrum may cause the assumption of a smoothly varying function to be broken and the Gamma Test to fail to return an accurate estimate of the noise. Running the Gamma Test over all channels on this spectrum returns a Γ = 0.000310, leading to an estimate of the rms noise of 17.6 mJy. This invalidates the

utility of the Gamma Test (used in this way) to measure the noise on the spectrum. However, it does suggest that the Gamma Test could be used to detect the presence of a discontinuity in a spectrum, since the Γ value from a section of the spectrum containing the discontinuity may be much higher than the Γ from any section not containing the discontinuity.

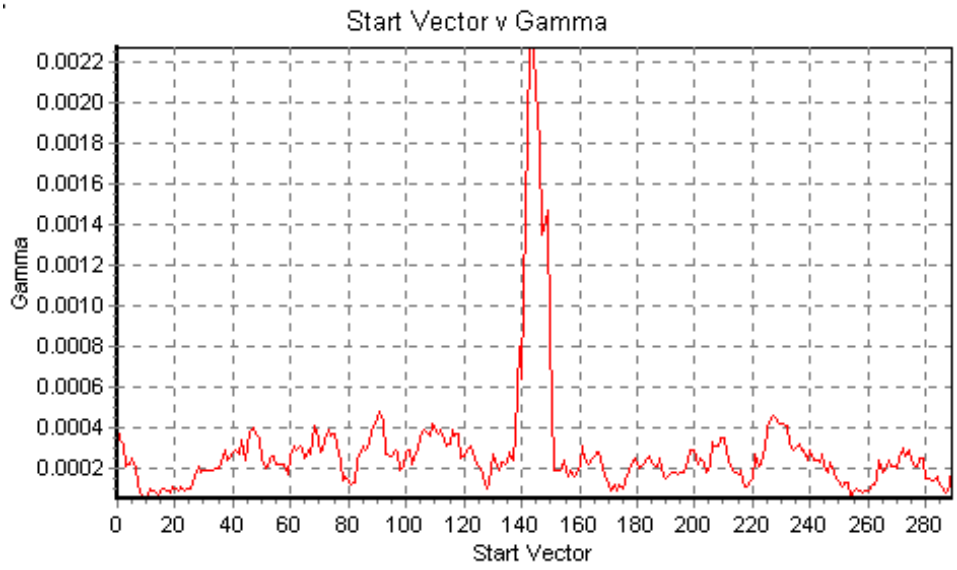


Figure 3.3. Result of running a 'moving-window' Gamma Test (window size=11 channels, p=10), using *winGamma*, on the spectrum shown in Figure 3.2(c).

This effect is illustrated in Figure 3.3 which shows the result of running a 'moving-window' Gamma Test on the spectrum in Figure 3.2(c). The 'moving-window' Gamma Test undertakes a Gamma Test centered on each channel in the input spectrum. Each Gamma Test is conducted inside a 'window' of a fixed number of channels. In Figure 3.3, a window size of 11 channels was used and a maximum number of nearest neighbours, p=10. Figure 3.3 shows the resulting Γ returned for each channel of the input spectrum.

There is a sharp peak in Γ (0.002272) at the position of the spectral line. Elsewhere, the returned Γ values are distributed around the input variance of 0.000225. In fact, the window size of 11 was chosen because this window size was found to produce the largest peak in Γ at the position of the spectral line compared to the returned Γ values for the rest of the spectrum. A comparison of Figures 3.2(c) and 3.3 illustrates that using a moving-window Gamma Test can be a highly effective way of detecting discontinuities in a spectrum.

Central to the operation of the Gamma Test is the comparison of the output (y) values from neighbouring input ($\boldsymbol{x}$) points. Since the Gamma Test assumes a smooth underlying function, if the y values between neighbouring $\boldsymbol{x}$ positions are widely divergent, this is considered to be due to noise. The sharp peak in Figures 3.2(c) is due to an emission line, not noise. However, the Gamma Test in effect is still trying to calculate the variance of the noise, but the whole of the signal caused by the discontinuity is being treated as an instance of noise. A very high Γ is returned since 3 of the channels in the window have widely divergent output (y) values compared to those of the other channels.

A moving-window Gamma Test can also be used to return a reasonably accurate estimate of the variance of the noise, despite the presence of a spectral line. Each Gamma Test is estimating the noise within its 11-channel window. Clearly those Gamma Tests which have the spectral line within their window will give very inaccurate measures of the noise. Nonetheless, by taking the median of the returned Γ values, a typical value of Γ for the spectrum can be derived. The median value of Γ in the 'gamma-spectrum' of Figure 3.3 is 0.000251, equivalent to an rms noise of 15.8 mJy. This is very similar to the returned Γ from running a Gamma Test using all 300 channels on spectrum in Figure 3.2(b) (i.e. without the spectral line).

3.4 Using the Gamma Test to detect RFI in HIPASS / HIJASS data

The utility of the Gamma Test to detect discontinuities in the spectra was demonstrated in Section 3.3. The application of this to detecting RFI in spectra is considered in this section.

Figure 3.4(a) shows a section of a HIPASS spectrum. The narrow 'spike' seen to positive intensities is due to local RFI and could potentially be confused with a narrow velocity-width galaxy. Figure 3.4(b) shows the result of running a 'moving window' Gamma Test over this spectrum using a window size of 11 channels. The maximum number of nearest neighbours used is p=15 (as opposed to *winGamma*'s default value of 10) since this was found to give the optimum peak in the returned Γ value at the position of the RFI 'spike'. (Note that Figure 3.4 and subsequent ones were produced using a prototype of GammaFinder, since *winGamma* does not permit a p value larger

than the window size – see Section 6.2.) The spectrum in the lower panel of Figure 3.4(b) shows the resulting Γ from each test (henceforth referred to as the 'gamma spectrum'). There is a sharp peak in the gamma spectrum at the position of the RFI 'spike'.

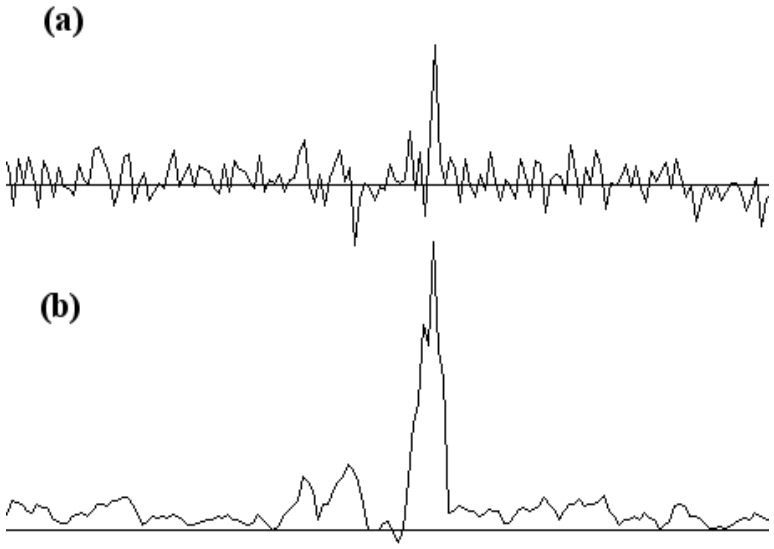


Figure 3.4. (a) 160 channel-wide (2100 km s^{-1}) section of a HIPASS spectrum with a 'spike' to positive intensities due to local RFI; **(b)** The 'gamma spectrum' resulting from running a moving-window Gamma Test (window size = 11 channels, p=15) over the spectrum in (a).

Figure 3.5(a) shows part of another spectrum from a different (x,y) position of the same HIPASS cube shown in Figure 3.4(a). This spectrum also contains a 'spike', this time seen to negative intensities. This spike is due to the same RFI as that seen in Figure 3.4(a) and, consequently, is at the same frequency. Figure 3.5(b) shows the result of running a moving window Gamma Test over the spectrum in (a). Again, the gamma spectrum shows a very pronounced peak at the frequency of the RFI.

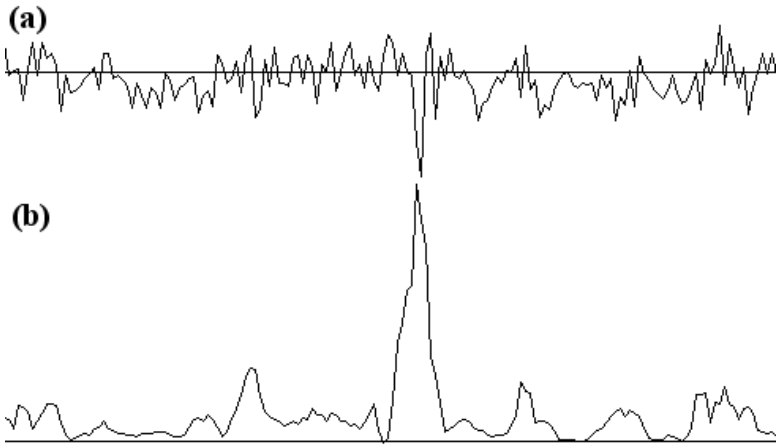


Figure 3.5. (a) 160 channel-wide (2100 km s^{-1}) section of a HIPASS spectrum with a 'spike' to negative intensities due to local RFI; **(b)** The 'gamma spectrum' resulting from running an 11-channel wide (p=15) moving-window Gamma Test over the spectrum in (a).

The peak fluxes of the 'spikes' seen in Figures 3.4(a) and 3.5(a) are only about a factor 2.5 above the rms noise in the spectra. However, the peak Γ in the gamma spectra in Figures 3.4(b) and 3.5(b) are about a factor of 7 above the rms variation in the gamma spectra. Hence, it will be easier to detect RFI from the gamma spectra than the raw data, especially since the peak in the gamma spectrum is always to positive intensities.

3.5 Using the Gamma Test to detect galaxies in HIPASS / HIJASS data

The utility of the Gamma Test to detect discontinuities in the spectra was demonstrated in Section 3.3. The application of this to detecting galaxies in the data is considered in this section.

Figure 3.6(a) shows part of a spectrum from a HIJASS cube. This spectrum includes the HI emission line from the galaxy UGC 05701. This galaxy has one of the smallest peak flux values (S_{pk} = 56 mJy) and narrowest velocity-widths (W20 = 70 km s^{-1}) in the HIJASS Catalogue (see Lang et al. 2003).

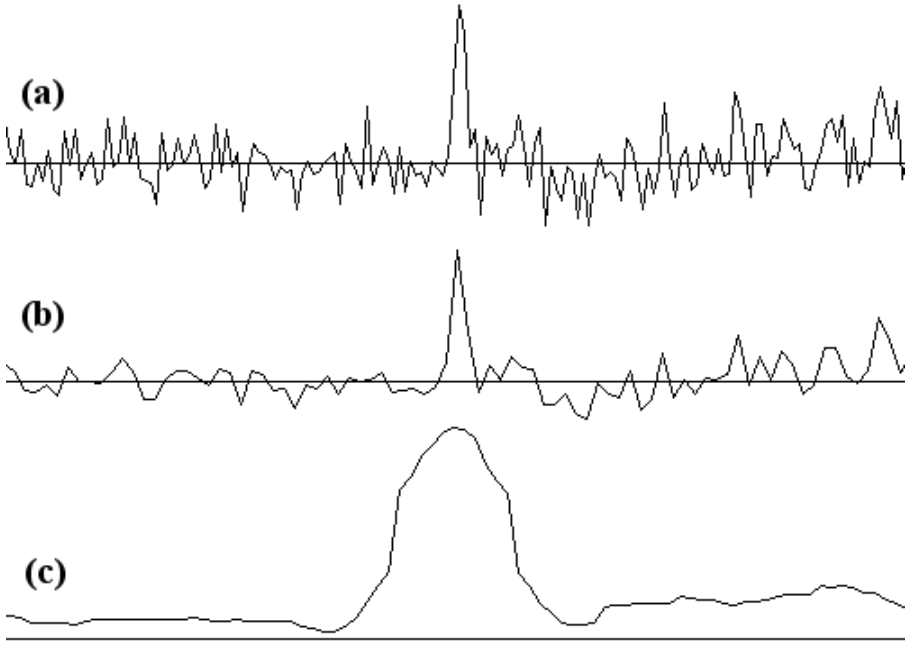


Figure 3.6. (a) 170-channel wide (2200 km s^{-1}) section of a HIJASS spectrum showing the emission line from UGC05701; **(b)** The spectrum shown in (a) after smoothing by a Hanning function with FWHM = 2 channels; **(c)** The gamma spectrum derived from the Hanning smoothed spectrum in (b) (11-channel window, p=15).

Figure 3.6(b) shows the spectrum which results from smoothing the spectrum in Figure 3.6(a) by a Hanning function with a FWHM of 2 channels. The technique of "Hanning smoothing" (see e.g. Blackman and Tukey 1959) is commonly used on radio spectral line data in order to reduce the level of noise on the data. The Hanning function is given by A(x) = ½ [1 + cos(⑥x/a)] where a is the FWHM. The intensity value of a channel in a Hanning smoothed spectrum (with FWHM = 2 channels) is dependent on the value of that channel and of the 2 adjacent channels in the unsmoothed spectrum. In order to produce a smoothed spectrum in which the channels are still independent, it is usual to include in the output spectrum only every other channel from the original.

If the noise is Poissonian in nature, then Hanning smoothing with FWHM = 2 channels should reduce the noise by a factor of 1.414. The smoothing process reduces the frequency resolution to twice the original channel width, i.e. to 26 km s^{-1} for HIPASS and HIJASS data. Hence, whilst Hanning smoothing may improve the detectability of a source (by reducing the noise) details in the original spectrum may be lost. It is also

possible that a very narrow velocity-width source may actually have its peak flux significantly reduced as it is averaged with surrounding pixels. However, most galaxies have intrinsic HI velocity-widths much larger than 26 km s^{-1} and there is usually little danger as far as detectability goes in applying a Hanning filter with FWHM of 2 channels.

Figure 3.6(c) shows the result of running a moving window Gamma Test over the Hanning smoothed spectrum shown in Figure 3.6(b). Exactly the same window size (11 channels) and maximum number of nearest neighbours (p=15) were used as in Figures 3.4 and 3.5. Note that both the Hanning smoothed spectrum and the gamma spectrum have only half the number of channels as the raw spectrum. However, all three have been plotted on the same cz scale to allow comparison.

A very strong peak is seen in the gamma spectrum at the position of the HI emission line. Whilst the peak flux of the emission line on the raw spectrum is about a factor of 3 above the rms noise level, the peak in the gamma spectrum is about a factor of 10 above the rms variation in the gamma spectrum. A moving window Gamma Test is clearly a highly effective way of finding narrow velocity-width galaxies.

Figure 3.7(a) shows a spectrum from a HIJASS cube showing the galaxy UGC 06552. This also has a small peak flux value (S_{pk} = 62 mJy) but has a broader velocity-width (W20 = 273 km s^{-1}) than UGC05701. Figure 3.7(b) shows the result of smoothing this spectrum with a Hanning function with FWHM = 2 channels. Figure 3.7(c) shows the resulting gamma spectrum from running a moving window Gamma Test over this Hanning smoothed spectrum. In this example, no significant peak is seen in the gamma spectrum to mark the position of the galaxy's HI line.

A moving window Gamma Test is far more effective at detecting a discontinuity in a spectrum if that discontinuity only occupies a small number of the channels in the window. In the Hanning-smoothed spectrum of Figure 3.7(b), UGC 06552 occupies 6 of the 11 channels in the Gamma Test window. Whilst it has a similar peak flux to UGC 05701, this latter source occupied only 3 of the 11 channels in the Gamma Test window (Figure 3.6(b)). That the Gamma Test was far more effective at detecting UGC 05701 is not surprising since the Gamma Test is effectively measuring an average of

the difference between the intensity values between neighbouring channels. If more of the pixels within the window are actually part of the galaxy then this average difference will be smaller. This implies that the Gamma Test is less effective at detecting broader velocity-width galaxies. Fortunately, this problem can be overcome.

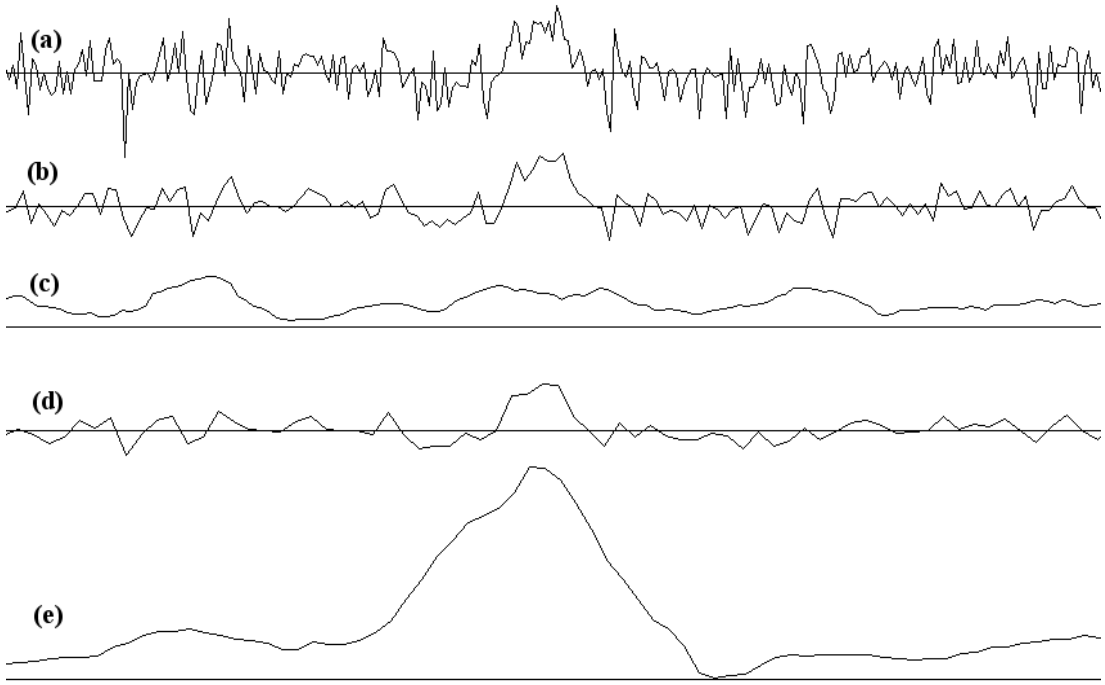


Figure 3.7. **(a)** 290-channel (3800 km s^{-1}) wide section of a HIJASS spectrum including the emission line from UGC 06552; **(b)** The spectrum shown in (a) after smoothing by a Hanning function with FWHM = 2 channels; **(c)** The gamma spectrum derived from the Hanning smoothed spectrum shown in (b) (11-channel window, p=15); **(d)** The spectrum shown in (a) after smoothing by a Hanning function with FWHM = 4 channels; **(e)** The gamma spectrum derived from the Hanning smoothed spectrum shown in (d) (11-channel window, p=15).

Figure 3.7(d) shows the result of smoothing the original spectrum by a Hanning Function with a FWHM of 4 channels. In such a smoothed spectrum, the value of every channel is dependent on the value of that channel and the 6 nearest channels in the original spectrum. Therefore, only every 4^{th} channel is retained in the smoothed spectrum. Smoothing by FWHM = 4 channels will reduce the frequency resolution to 52 km s^{-1}, but should reduce the noise by a factor of 2. However, whereas the galaxy occupied 6 channels in the FWHM = 2 channel Hanning smoothed spectrum, it occupies only 3 channels in the FWHM = 4 channel Hanning smoothed spectrum. Figure 3.7(e) shows the result of running a moving-window Gamma Test over the

spectrum in Figure 3.7(d). The galaxy is now very clearly detected.

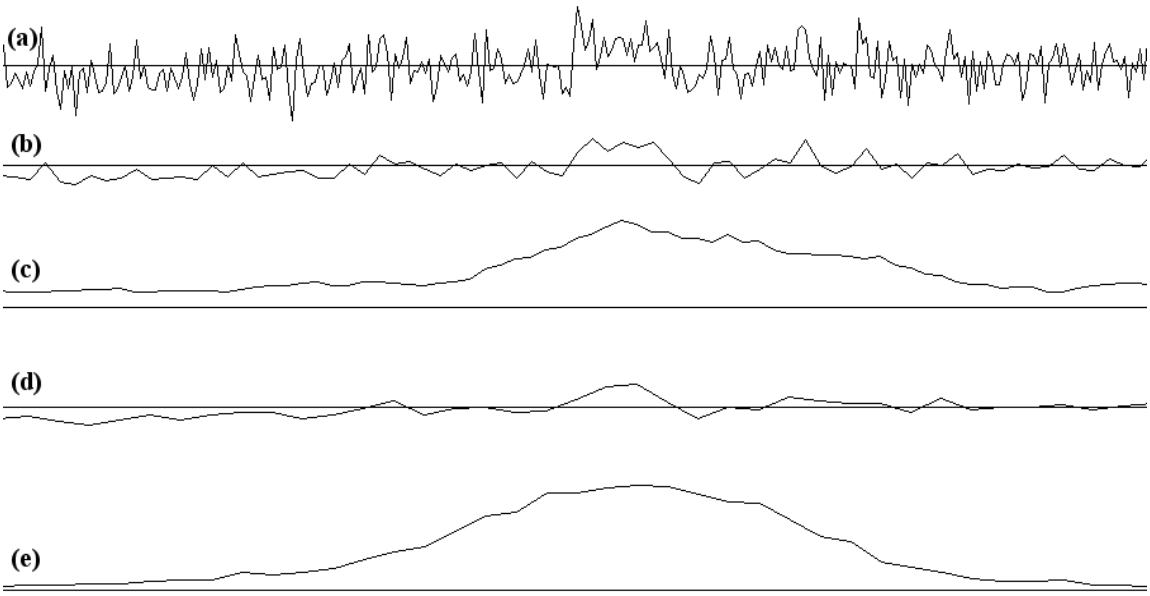


Figure 3.8. (a) 320-channel (4200 km s^{-1}) wide section of a HIJASS spectrum containing the emission line from the galaxy UGC6434; **(b)** The spectrum shown in (a) smoothed by a Hanning function with FWHM of 4 channels; **(c)** The gamma spectrum derived from the Hanning smoothed spectrum shown in (b) (11-channel window, p=15); **(d)** The spectrum shown in (a) smoothed by a Hanning function with FWHM 8 channels; **(e)** The gamma spectrum derived from the Hanning smoothed spectrum shown in (d) (11-channel window, p=15).

This technique can be extended further. Figure 3.8(a) shows the HIJASS spectrum of UGC 6434. This galaxy also has one of the lowest peak flux values (S_{pk} = 61 mJy) in the HIJASS catalogue, but has one of broadest velocity-widths (W20=340 km s^{-1}). Figure 3.8(b) shows the spectrum after being smoothing by a Hanning function with FWHM = 4 channels. Figure 3.8(c) shows the gamma spectrum derived from this spectrum. Only a very modest peak is seen in Γ at the position of the emission line. For such a broad velocity-width, even in the FWHM = 4 channels Hanning smoothed spectrum, the galaxy is occupying 6 channels. However, Fig 3.8(d) shows the result of Hanning smoothing the original spectrum with a FWHM = 8 channels. The value of a channel in the smoothed spectrum is dependent on the value of that channel and the 14 nearest channels in the original spectrum. Hence, only every 8^{th} channel is retained in the smoothed spectrum. The frequency resolution is reduced to 104 km s^{-1}. However, the number of channels in which the galaxy is detected is only 3: it represents a

significant discontinuity in the smoothed spectrum.

Figure 3.8(e) shows the gamma spectrum derived from Figure 3.8(d). The very broad velocity-width source creates a very high peak in Γ. The apparent broadness of the peak is slightly misleading. It should be remembered that this spectrum has only 1/8 of the pixels of a raw spectrum. Also, only about 38% part of the spectrum is being plotted. The rest of the spectrum contains sufficient data points to provide a baseline against which the peak can be detected.

This section has presented a discussion of how the Gamma Test can be used to detect 3 galaxies. These galaxies were chosen from those with the lowest S_{pk} values in the HIJASS catalogue, but represent the full range of velocity-widths of galaxies in the catalogue. By Hanning smoothing the data by an appropriate FWHM, all three could be easily detected by running a moving-window Gamma Test over the smoothed spectrum. The use of three smoothing FWHMs (2, 4 and 8) enables a search to be made for galaxies over a velocity-width range from about 50 km s^{-1} to 400 km s^{-1}, more or less the range measured for known galaxies.

3.6 Procedures for analysing HIPASS / HIJASS datacube

As explained in Sections 3.3, 3.4 and 3.5, there are three main types of analysis of a HIPASS / HIJASS datacube for which the Gamma Test can be utilized: measuring the variance of the noise, finding z-channels affected by RFI and finding possible sources. In this section three procedures are presented to undertake each of these types of analysis.

The `findNoise` procedure estimates the average noise in a datacube. The procedure runs a moving-window Gamma Test (11-channel window, p=15) over the z-channel spectrum at every spatial position in the datacube. It then finds the average variance of the noise in the datacube by taking the median value of all the returned Γ values. Choosing the median value means that a 'typical' value of variance will be returned, i.e. not one derived from a channel affected by RFI , an emission line or residue Galactic emission. The average (rms) noise is derived by taking the square root of the variance.

```
Procedure findNoise( cube(xpix,ypix,zpix) )

for i= 1 to xpix do
   for j = 1 to ypix do
      for k = 1 to zpix do

      gamStat(i,j,k) ← 🏛 from cube(i, j, k-5 → k+5), p =15

     end for
   end for
end for

averageVariance ← median [ gamStat(i, j, k) ]

averageNoise = sqrt( averageVariance )

return averageNoise
```

The `findBadZChans` procedure finds those z-channels affected by RFI in a datacube. For each z-channel, a median value is found of the Γ resulting from running a Gamma Test centred at every (i,j) position in that channel (window size of 11, p=15). If the z-channel suffers from RFI then this value will be significantly larger than if the channel does not suffer from RFI. The median Γ for each z-channel is copied to the array `medGamInChan`. The median value from within this array, `medGam`, is then found. Finally, the value of `medGamInChan` for each channel is compared with medGam. If the former is less or equal to a factor of 1.5 times the latter, than the channel is considered to be free of RFI and the appropriate value of the Boolean array zChanQual is set to true (else it is set to false).

```
Procedure findBadZChans( cube(xpix,ypix,zpix) )

zChanQual(zpix) ← Boolean array to note which z-Channels are
                  affected by RFI. All values set to 'false'

for k = 1 to zpix do
l ← 0
   for i= 1 to xpix do
      for j = 1 to ypix do

      gamStat(l) ← 🖷 from cube(i, j, k-5 → k+5), p=15
      l++

     end for
   end for

   medGamInChan(k) ← median( gamStat )

end for

medGam ← median( medGamInChan )


for k = 1 to zpix do

   if medGamInChan(k) <= medGam * 1.5
      zChanQual(k) = true

end for

return zChanQual
```

The `findGals` procedure finds possible sources in a datacube. This has three main stages.

Firstly, three Hanning smoothed spectra are derived from the spectrum at each (i,j) position, using FWHMs of 2, 4 and 8 channels respectively.

Secondly, at each (i,j) position, a moving window Gamma Test (11-channels, p=15) is run on each of these 3 smoothed spectra, resulting in 3 'gamma spectra'.

```
Procedure findGals( cube(xpix,ypix,zpix) )

gamThres ← a number used to determine if a peak in a gamma
           spectrum is significant or not

for i= 1 to xpix do
   for j = 1 to ypix do

      h2Spec(i,j) ← spectrum at cube(i,j) smoothed by
                    Hanning function with FWHM = 2 channels

      h4Spec(i,j) ← spectrum at cube(i,j) smoothed by
                    Hanning function with FWHM = 4 channels

      h8Spec(i,j) ← spectrum at cube(i,j) smoothed by
                    Hanning function with FWHM = 8 channels
   end for
end for

for i= 1 to xpix do
   for j = 1 to ypix do

      g2Spec(i,j) ← 'gamma spectrum' from 11-ch, p=15 moving-
                     window Gamma Test on h2Spec(i,j)

      g4Spec(i,j) ← 'gamma spectrum' from 11-ch, p-15 moving-
                    window Gamma Test on h4Spec(i,j)

      g8Spec(i,j) ← 'gamma spectrum' from 11-ch, p=15 moving-
                    window Gamma Test on h8Spec(i,j)
   end for
end for

for i = 1 to xpix do
   for j = 1 to ypix do
      for k = 1 to zpix do

         if ( g2Spec(i,j,k) > gamThres*median( g2Spec(i,j) )
             && g2Spec(i,j,k) is a peak )  )
            Output i, j, k, g2Spec(i,j,k)

         if ( g4Spec(i,j,k) > gamThres*median( g4Spec(i,j) )
             && g4Spec(i,j,k) is a peak )  )
            Output i, j, k, g4Spec(i,j,k)

         if ( g8Spec(i,j,k) > gamThres*median( g8Spec(i,j) )
             && g8Spec(i,j,k) is a peak )  )
            Output i, j, k, g8Spec(i,j,k)

      end for
   end for
end for
```

Thirdly, the gamma spectra are searched for positions where Γ has a value larger than a factor '`gamThres`' times the median value for Γ in that gamma spectrum. If one of these positions is also at a peak in the distribution of Γ values (in all 3 dimensions) then the position and the value of Γ at that position are output as a possible sources. This procedure is run on all 3 sets of gamma spectra (since these are sensitive to galaxies of different velocity-widths).

Throughout the rest of this dissertation, we refer to the ratio of Γ at a particular (i,j,k) position to the median value for Γ in the equivalent gamma spectrum, at (i, j), as the 'Gamma S/N' (by analogy to the signal-to-noise ratio used to express the significance of flux detections - see Section 2.4.2). This ratio measures how far above a 'typical' Γ for that gamma spectrum the particular Γ value is. To be considered a possible source, a voxel must have a Gamma S/N μ `gamThres`. Hence, the factor '`gamThres`' can be thought of as representing the limiting Gamma S/N at which a peak in the Γ values is considered significant. It is important that the median value of Γ in a gamma spectrum is used in this calculation. This prevents the reference used to test significance being biased by emission lines or RFI.

3.7 Concluding remarks

This Chapter has described how the Gamma Test can be a useful analytical technique when applied to HIPASS / HIJASS datacubes. Three procedures have been described which use the Gamma Test to (a) determine the average noise in a datacube; (b) find z-channels affected by RFI in a datacube; (c) find potential sources in a datacube. The next two chapters describe the design and implementation of a Java application (GammaFinder) which implements these procedures.

4. Specification and Design of GammaFinder

4.1 Introduction

Section 3.5 presented three procedures which utilise the Gamma Test to undertake an aspect of the analysis of HIPASS / HIJASS datacube: `findNoise` estimates the average noise in a datacube; `findBadZChans` finds those z-channels in a datacube affected by RFI; and `findGals` find possible sources in a datacube. This chapter presents the specification and design of a software application, GammaFinder, which puts these procedures into operation. The specification of the application is described in Section 4.2. Some general design issues are discussed in Section 4.3. Section 4.4 contains a description of the Java classes of the application. In Section 4.5 a description of the sequence of operation of a typical GammaFinder session is given.

4.2 Requirements specification

4.2.1 Functional requirements

Figure 4.1 is a top level UML Use Case Diagram which represents the main functionality that the application was designed to provide.

`Use Case 1. Determine the average noise in a datacube using the findNoise procedure.`

Use Case 2. Determine the bad z-channels in a datacube using the findBadZChans procedure.

Use Case 3. Find possible sources in a datacube using the findGals procedure.

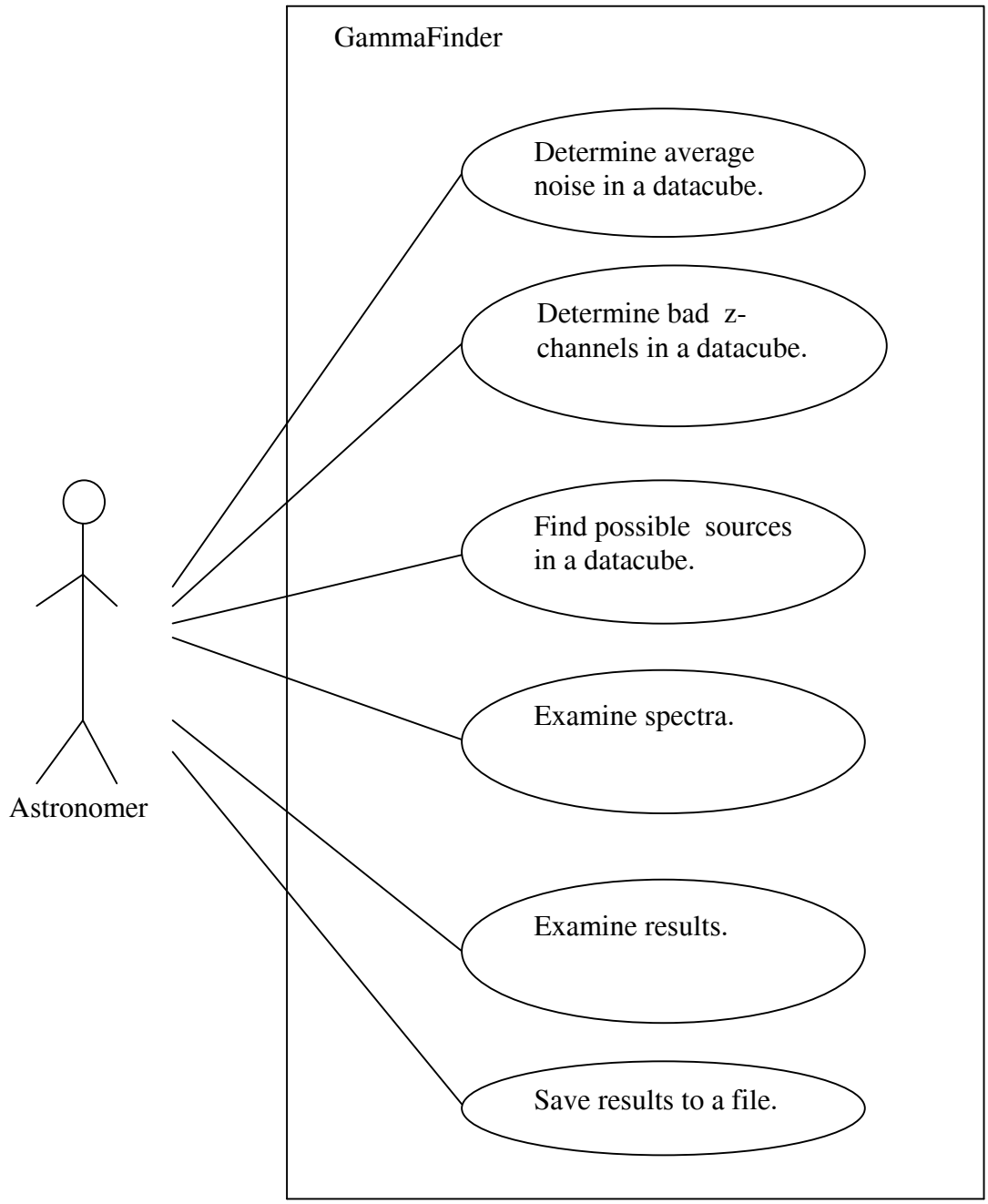


Figure 4.1. A top level UML Use Case Diagram representing the main functionality that GammaFinder was designed to provide.

`Use Case 4. Examine the spectra associated with each spatial position in a datacube.`

The user can study each of the 3 types of spectra created by the `findGals` procedure. This is represented in the expanded Use Case Diagram in Figure 4.2.

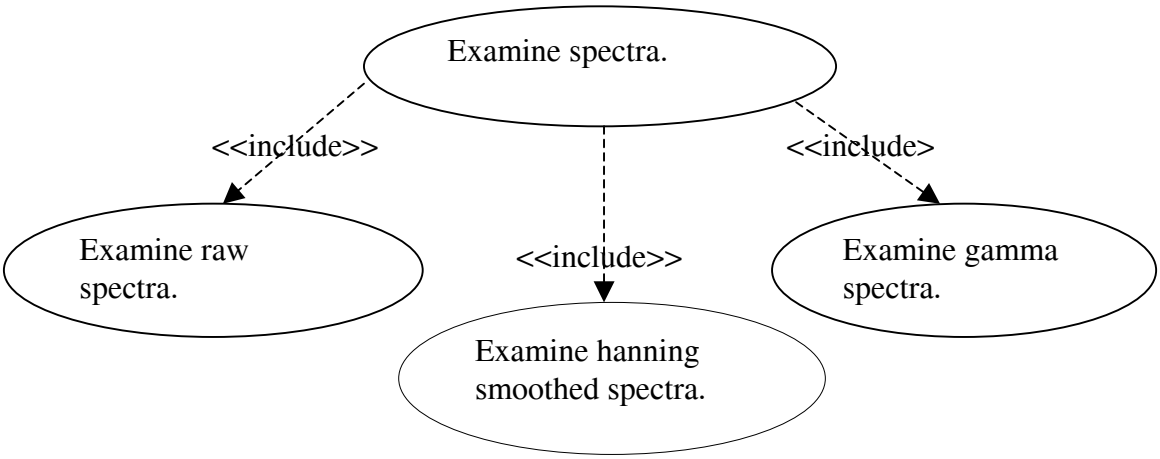


Figure 4.2. Expanded Use Case Diagram for Use Case 4.

`Use Case 5. Examine results.`

An astronomer is able to examine the results from each of Use Cases 1-3. There are three aspects to this use case, each of which is represented by the expanded Use Case Diagram in Figure 4.3.

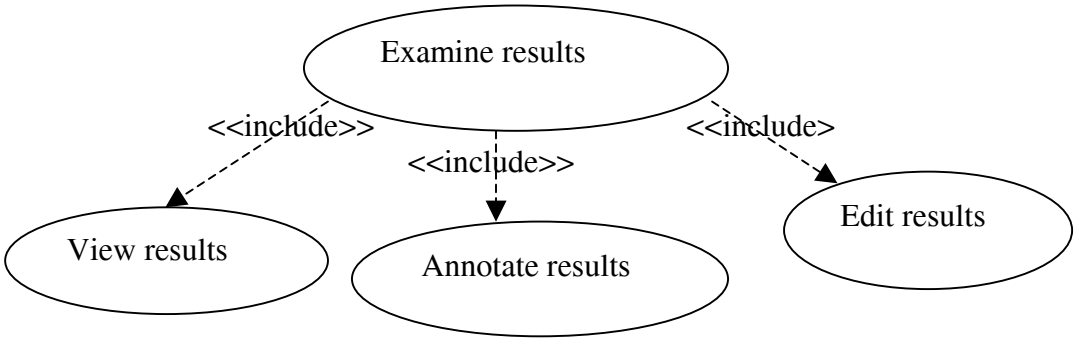


Figure 4.3. Expanded Use Case Diagram For Use Case 5.

`Use Case 6. Save Results to a file.`

4.2.2 Non-functional requirements

GammaFinder has been designed to meet the following non-functional requirements.

1. Portability. The intention is that GammaFinder will be used by astronomers throughout the world.
2. Straightforward to use. The prime functions of GammaFinder should be easily accessed and performed.
3. Make efficient use of memory and CPU time.
4. Work with the standard astronomical data format, 'FITS'.

4.3 General Design Issues

It was decided to build GammaFinder as a GUI-based application using Java Swing. The use of Java will ensure that the requirement of portability is met. The use of a GUI assists the requirement that the application be straightforward to use. Results and spectra can be easily examined and edited on a GUI. In the interests of the efficient use of memory and CPU time, the 'float' data type was chosen to represent all real numbers used within the application. No significant loss of accuracy results from using this data type as opposed to 'double'.

4.4 Explanation of the Java classes in GammaFinder

Figure 4.4 presents a UML Class Diagram of GammaFinder. An understanding of the data structures and methodology of GammaFinder can be obtained by a consideration of each of these classes and their relation to each other. These are discussed below.

4.4.1 `public class Voxel`

This class was designed to represent a single voxel in an HI datacube. Whilst a voxel in the original datacube has only an (x, y, z) position and an intensity value associated with it, the `Voxel` object also has associated with it the values derived by running the Hanning smoothing functions (with FWHM=2, 4 and 8 channels) over the spectrum defined by the z-channels at the (x, y) position of the `Voxel` object. Also associated with the `Voxel` object are the Gamma Statistic, Γ, values derived by running a moving window Gamma Test (window size 11 channels, p=15) over the raw spectrum defined by the z-channels at the (x, y) position of the `Voxel` object and the Γ values derived from the equivalent 3 Hanning smoothed spectra. Hence, the instance variables of a `Voxel` object represent the 8 pieces of information which need to be associated with a voxel in order to implement the three procedures described in Section 3.6. This data

structure was chosen since it is conceptually easy to understand and models the real world situation well.

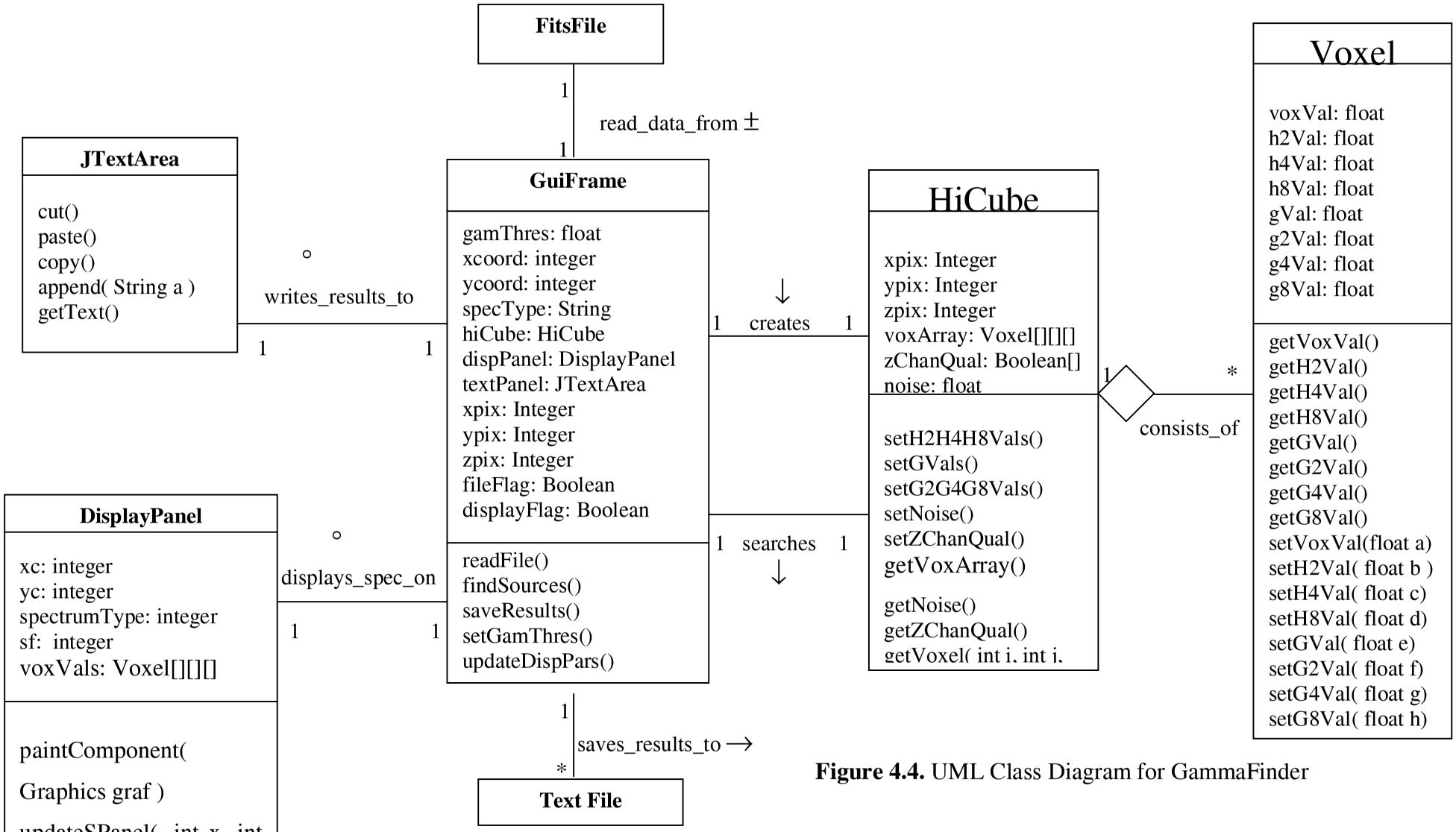


Figure 4.4. UML Class Diagram for GammaFinder

The instance variables of the class are:

voxVal: the intensity of a voxel.

h2Val, h4Val, h8Val: the intensities of a voxel after its z-channel spectrum has been smoothed by Hanning functions with FWHMs = 2, 4 and 8 channels respectively.

gVal: Γ value of a voxel derived from a moving window Gamma Test over the raw z-channel spectrum.

g2Val, g4Val, g8Val: Γ values of a voxel derived from a moving window Gamma Test over the 3 Hanning smoothed z-channel spectra of that voxel.

The constructor for the class creates a `Voxel` object by setting the value of its `voxVal` instance variable. All other instance variables can be set by specific methods within the class. Methods are also defined for returning the value of the each of the instance variables of the class. These are utilised by methods in the `HiCube` and `GuiFrame` classes.

4.4.2 **public class HiCube**

This class represents those aspects of an HI datacube of relevance to the analysis undertaken by GammaFinder. The instance variables of the `HiCube` class are:

xpix, ypix, zpix: integers representing the number of voxels in the Right Ascension, Declination and cz dimensions respectively.

voxArray: a three dimensional array of `Voxel` objects. Each `Voxel` object in this array represents the properties of a particular voxel in the HI datacube.

noise: a 'float' representing the average noise in the datacube.

zChanQual: a Boolean array of dimension `zpix`. Any 'bad' z-channels (i.e. seriously affected by RFI) in the datacube are represented by a value of "false" in this array. Good z-channels are represented by a value of "true".

There is a single constructor for this class. A 3-dimensional float array is passed to the constructor, representing the raw pixel values of the datacube. Using this array, the values of `xpix`, `ypix` and `zpix` are determined. A 3-dimensional array of `Voxel`

objects is created of the same dimensions as the input float array. The `voxVal` instance variable of each `Voxel` in the array is set to be equal to the value of the equivalent voxel in the input float array.

The following methods set the values of the instance variables of a `HiCube` object:

`setGVals()`: This method runs a moving window Gamma Test (window size 11-channels, p = 15) over each z-channel spectrum in the `HiCube` object. The result of this at each (i, j, k) position is stored as the `gVal` instance variable in the `Voxel` object at the relevant `voxArray` position.

`setH2H4H8Vals()`: This method runs three Hanning smoothing functions over each z-channel spectrum in the datacube. These functions have FWHMs of 2, 4 and 8 voxels respectively. The resultant intensity at each (i, j, k) position is stored as the `h2Val`, `h4Val` and `h8Val` instance variables in the `Voxel` object at the relevant `voxArray` position. Since the smoothed spectra have less channels that the raw spectra (see Section 3.5), these instance variables are left 'null' where that channel is excluded from the Hanning smoothed spectrum. This method implements the first part of the `findGals` procedure.

`setG2G4G8Vals()`: This method runs a moving window Gamma Test (window size = 11 channels, p = 15) over each Hanning smoothed spectrum in the datacube (i.e. the relevant set of `h2Val`, `h4Val` and `h8Val` instance variables at each (i, j) position). The resultant Γ at each (i, j, k) position is stored as the `g2Val`,`g4Val` and `g8Val` instance variables in the `Voxel` object at the relevant `voxArray` position. These values are left as 'null' where those of the equivalent Hanning smoothed spectrum are null. This method implements the second part of the `findGals` procedure.

`setNoise()`: This method implements the `findNoise` procedure to find the average noise in the datacube and sets the value of the `noise` instance variable.

`setzChanQual()`: This method implements the `findBadZChans` procedure to find those z-channels in the cube which suffer from significant RFI. It sets the values of the `zChanQual` instance variable to be "false" for these channels and "true" for all other channels.

The following methods return the values of instance variables of a `HiCube` object:

`getZChanQual()`: returns the Boolean array `zChanQual`.

`getNoise()`: returns the value of the `noise` instance variable.

`getVoxel( int i, int j, int j )`: return the `Voxel` object at position (i,j,k) in the `voxArray` instance variable.

`getVoxArray()`: return the `VoxArray` instance variable.

4.4.3 `public class GuiFrame`

This class defines the properties of the GUI (the design of the GUI is discussed in Chapter 5). This class also runs the application, utilising the other classes to derive and present the required results.

The class has many instance variables relating to element of the GUI. It also has the following instance variables:

`gamThres`: a float representing the 'Gamma S/N' limit (see Section 3.6).

`textPanel`: an object of `JTextPanel` class. All results are written to this object.

`dispPanel`: an object of class `DisplayPanel`, used for displaying spectra.

`hiCube`: an object of `HiCube` class. This represents the current datacube being analysed.

`xpix,ypix,zpix`: integers representing the dimension of the current `HiCube` object.

`xcoord, ycoord`: integers representing the position of the spectrum to be displayed on `dispPanel`.

`specType`: a String representing the type of spectrum to be displayed on `dispPanel.`

`fileFlag`: Boolean used to indicate if the file has been read in correctly.

`displayFlag`: Boolean used to indicate if `xcoord` and `ycoord` are set to permissible values.

The following methods are defined within the class.

`readFile( )`: This method opens a FITS datacube (selected by the user using the `JFileChooser` class) and creates a new `HiCube` object (`hiCube`) by passing the 3-D float array to the constructor of the `HiCube` class. To open and read the FITS data, it makes use of the org.eso.fits Java package written by Preben Grosbol of the European Southern Observatory.

`setGamThres():` This method reads the user supplied value of the `gamThres` instance variable from the GUI.

`findSources()`: This method searches the `HiCube` object, `hiCube`, for possible detections and returns the results of this search as a list of possible sources, along with the 'Gamma S/N' onto `textPanel`. Hence, it implements the third part of the `findGals` procedure.

`updateDispPars()`: This method reads the user supplied values for `xcoord`, `ycoord` and `specType` from the relevant input fields of the GUI.

`saveResults()`: This method saves the current contents of `textPanel` to an ascii text file (selected by the user using the `JFileChooser` class).

4.4.4 `public class DisplayPanel`

This class extends the `JFrame` class to provide a panel for displaying spectra requested by the user. An instance of this class, `dispPanel` is displayed on the GUI. The class has the following instance variables.

`voxVals`: the 3-dimensional array of `Voxel` objects from which spectra to be displayed are selected.

`xc`, **`yc`**: Integers defining the (i, j) position of the spectrum to be displayed.

`spectrumType`: an integer flag noting the type of spectrum to be displayed.

`sf`: an integer representing an appropriate scaling factor to be applied to the spectrum to be displayed.

The following methods are defined:

updateSPanel(int xc, int yc, Voxel[][][]voxVals, String specType): The method sets the instance variables to suitable values for the spectrum requested and then calls the repaint() method.

paintComponent(Graphics2D graf): this method redraws the display.

4.4.5 `class JTextArea`

This class is a component of the Swing package used for displaying and editing text. The `GuiFrame` object creates a single instance of this class, `textPanel`, to which it writes all the results of its other activities. The following methods are used:

copy(), cut(), paste(): standard text editing methods.

append(String aString): appends a string to the `JTextArea` object.

getText(): copies text on the `JTextArea` object to a string.

4.4.6 `public class FitsFile`

This class represents a FITS format image. The methods and classes used by the `GuiFrame` class to open and read data from a `FitsFile` object are provided by the `org.eso.fits` package.

4.4.7 `public class Stats`

This contains a static method, `modMean()`, called by the `findSources()` method of `GuiFrame` class to calculate a modal value in a `g8Val` gamma spectra (described in Section 5.9).

4.4.8 `class DisplayFormat`

This contains four static methods, called by the `findSources()` method of the `GuiFrame` class, and used to format the displayed results.

4.5 The sequence of operation of GammaFinder

Figure 4.5 is a UML Sequence Diagram which illustrates the standard sequence of operation of GammaFinder. This sequence is described below with reference to the diagram.

1. The application is started by an astronomer. An object of the `GuiFrame` class is created (`guiFrame`). The GUI is displayed.

2. The astronomer chooses to open a file. This leads to a sequence of operations which open a FITS file and create a `HiCube` object based upon its contents.

2.1 `guiFrame` calls the `readFile()` method. The astronomer chooses the `FitsFile` object to be opened using the `JFileChooser` class. The method opens the chosen file and returns a 3-dimensional float array of its contents.

2.2 The `readFile()` method then creates a new `HiCube` object (`hiCube`) passing `dataArray` to the constructor of the `HiCube` class.

2.3 `guiFrame` applies the `setGVals()` method of `HiCube` to `hiCube`. This performs the moving window Gamma Test on every spectrum in `hiCube` and copies the results into the `gVal` instance variable of each `Voxel` object in the `voxArray` instance variable of `hiCube`.

2.4 `guiFrame` applies the `setZChanQual()` method of `HiCube` to `hiCube`. This determines which z channels in `hiCube` suffer from RFI and sets the values of the `zChanQual` instance variable of `hiCube` to reflect this. These values are then excluded from the subsequent reduction and analysis process.

2.5 `guiFrame` applies the `setH2H4H8Vals()` method of `HiCube` to `hiCube`. This performs three Hanning smoothing operations on every spectrum in `hiCube` and copies the results into the `h2Val`, `h4Val` and `h8Val` instance variables of each `Voxel` object in the `voxArray` instance variable of `hiCube`.

2.6 `guiFrame` applies the `setG2G4G8Vals()` method of `HiCube` to `hiCube`. This performs a moving window Gamma Test on each channel of each of the Hanning smoothed (of FWHM = 2, 4 and 8 channels) spectra in `hiCube` and copies the resulting Γ values into the `g2Val`, `g4Val` and `g8Val` instance variables of each `Voxel` object in the `voxArray` instance variable of `hiCube`.

2.7 guiFrame applies the setNoise() method of HiCube to hiCube. This calculates the average noise in the datacube and copies the result to the instance variable noise.

2.8 guiFrame applies the getNoise() method of HiCube to hiCube. This return the value of the noise instance variable to GuiFrame.

2.9 guiFrame applies the getZChanQual() method of HiCube to hiCube. This returns the Boolean array zChanQual to guiFrame.

2.10 guiFrame applies the append() method of JTextArea to copy the value of noise and a list of the z-channels affected by RFI to textPanel.

3. The astronomer selects the 'Find' option from the GUI.

3.1 guiFrame calls its findSources() method. This actually makes many calls to the various values stored in the array of Voxel objects in hiCube, utilising the getVoxel(i, j, k) method of HiCube and the various 'get' methods of the Voxel class.

3.2 guiFrame applies the append() method to copy the results from the finding process to textPanel.

4. The astronomer selects an (i, j) position and a spectrum type and selects the 'Display Spectrum' JButton on the GUI.

4.1 guiFrame applies its setDispPars() method which reads the user input.

4.2 guiFrame calls the updateSPanel() method of the DisplayPanel class and supplies the user-selected xcoord, ycoord, specType values, along with the voxVals array.

4.2.1 dispPanel calls its paintComponent() method and the panel is re-drawn with the requested spectrum.

5. The astronomer directly edits the results on textPanel.

6. The astronomer selects the 'Save' option on the GUI.

6.1 `guiFrame` calls its `saveResults()` method. Firstly, this enables the user to select a name for an output text file (using the `JFileChooser` class).

6.2 The `saveResults()` method then applies the `getText()` method to `textPanel`. This method returns the current contents of `textPanel` as a String.

6.3 The `saveResults()` method then copies the returned results to the output file.

7. The astronomer selects the 'Exit' option on the GUI and `GammaFinder` is closed.

4.6 Concluding remarks

This chapter has presented a description of the main functional and non-functional requirements for GammaFinder and discussed the top level design of the application. The next chapter will give a more detailed description of the implementation of the design.

5. Implementation

5.1 Introduction

The specification and design of GammaFinder were described in Chapter 4. In this Chapter the implementation details of the most important aspects of the application are described. In Section 5.2 the main features of the GUI are discussed. Subsequent sections then describe the implementation of the main methods. These are described in the order in which they would be called during a typical sequence of operation (see Figure 4.5). Section 5.12 presents some concluding remarks.

5.2 The Graphical User Interface

The properties of the GUI are defined by the `GuiFrame` class. The GUI was designed using components of the Java Swing package since these give greater flexibility and sophistication that the standard Java awt package. Figure 5.1 shows the GUI as it appears when the application is started.

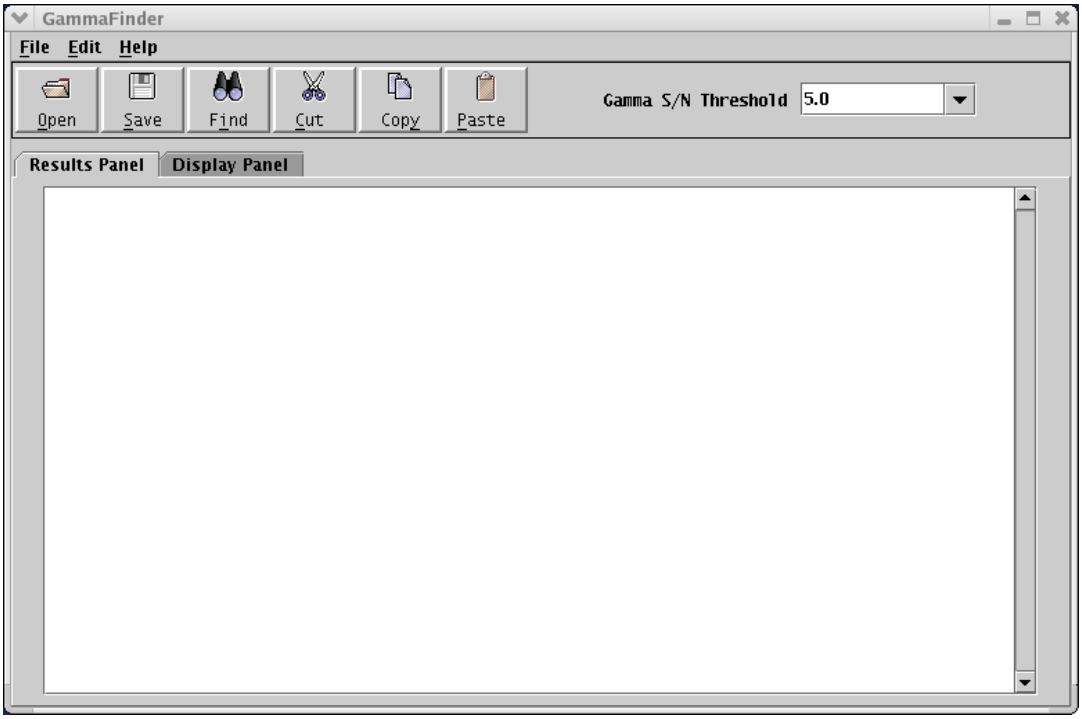


Figure 5.1. The GammaFinder GUI as it appears when the application is started.

The `GuiFrame` class extends the `JFrame` class. A 'cross-platform' look and feel is specified to help meet the requirement for portability.

Most of the functionality of the application can be accessed from the menu bar. The menu bar consists of a `JMenuBar` object which has added to it three `JMenu` objects, corresponding to the three menu headings 'File', 'Edit' and 'Help'. Each of the `JMenu` objects has added to it several `JMenuItem` objects, corresponding to the actual items on the menu. A keyboard shortcut is specified for each `JMenu` and `JMenuItem` object using the `setMnemonic()` method. An event listener is added to each `JMenu` and `JMenuItem` object using the `addActionListener()` method. Figure 5.2 shows the GUI with the 'File' menu activated.

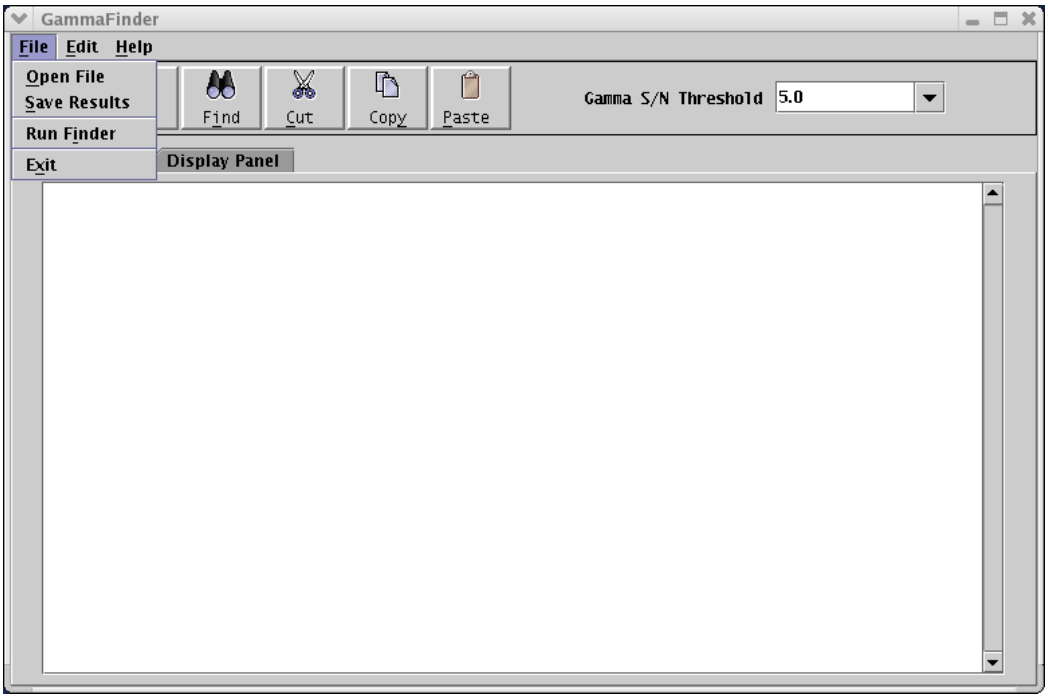


Figure 5.2. The GUI showing the 'File' menu activated.

Most of the functionality of the application can also be accessed from the tool bar. This consists of a `JPanel` object on which are placed 6 `JButton` objects, each of which corresponds to one of the main tasks of the application. A keyboard shortcut and an event listener are added to each `JButton`. The user is able to select the `gamThres` instance variable via a `JComboBox` object.

The lower part of the GUI consists of a `JTabbedPane` object to which are added 2 `JPanel` objects. The first of these contains a `JTextArea` object (`textPanel`) set in a `JScrollPane`. This forms the "Results panel" to which results are written. The

second contains a `DisplayPanel` object (`dispPanel`) on which spectra can be displayed. It also contains a `JComboBox` object from which the type of spectrum (`specType`) to be displayed can be selected and 2 `JTextField` objects, via which the user can specify the `xcoord` and `ycoord` values of the spectrum to be displayed. A `JButton` object is selected to display the spectrum. Figure 5.3 shows the GUI with the "Display Panel" selected.

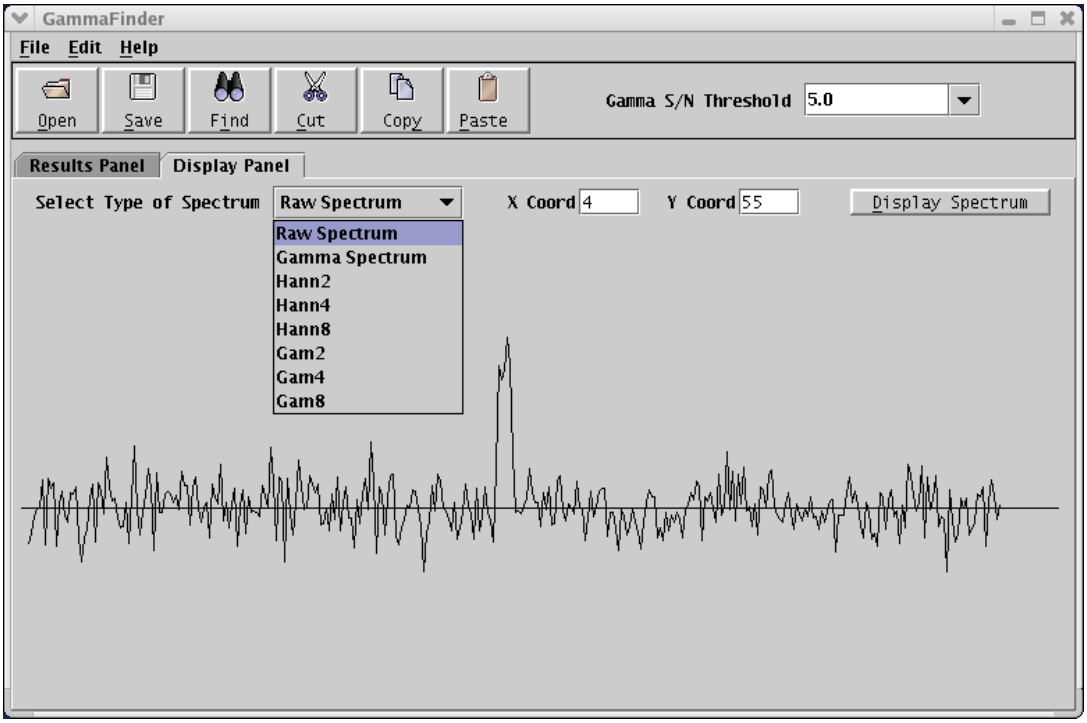


Figure 5.3. The GUI with the "Display Panel" visible.

An '`actionPerformed`' inner class within the `GuiFrame` class specifies the actions to be performed when one of the interactive elements of the GUI is selected. The following code extract shows what happens when the 'Save' button or the 'Save Results' menu item is selected.

```
else if( e.getSource() == j2 || e.getSource() == button2 )
{
   saveResults();
}
```

5.3 The `readFile()` method

When the user selects the 'Open' button or the 'Open File' option on the 'File' menu, a sequence of operations is activated. Firstly, the `readFile()` method of the

`GuiFrame` class uses the `JFileChooser` class to enable the user to select a file with a '.fits' extension (see Figure 5.4).

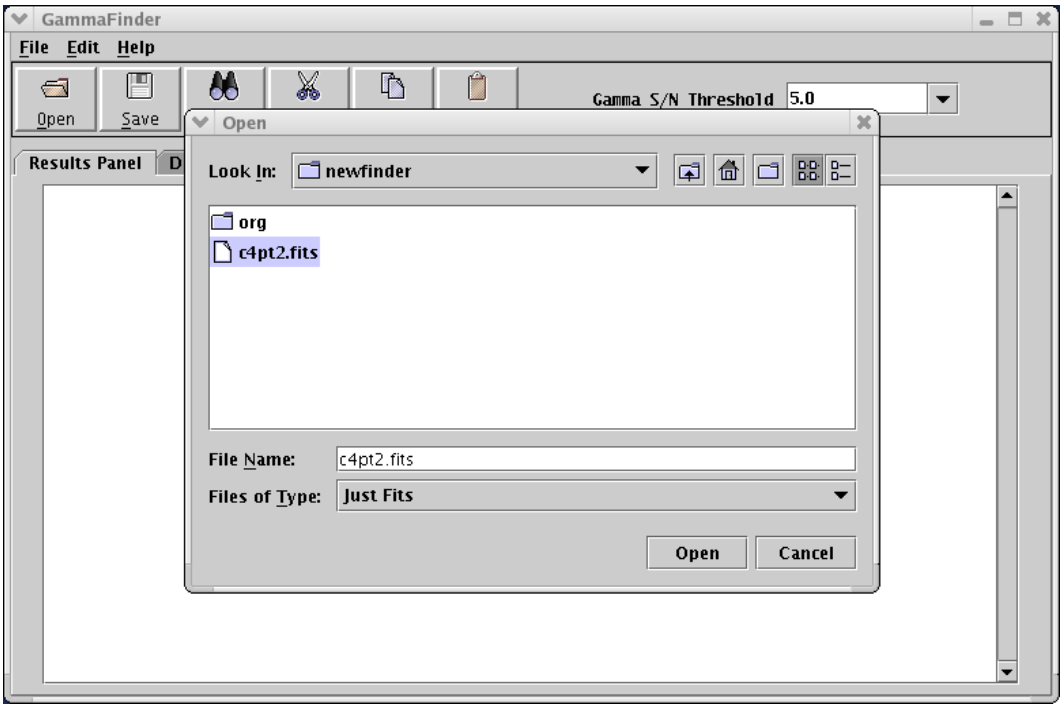


Figure 5.4. A user selecting a FITS file.

If the file selected is not a FITS file or cannot be read then the relevant exception is caught and the user is notified via a `JOptionPane` object (Figure 5.5).

Provided that the selected file can be opened, the `readFile()` method uses several methods from the org.eso.java package to (a) determine that the FITS file contains a 3-D datacube; (b) find the dimensions (`xpix, ypix, zpix`) of that cube; (c) read the intensity values at each (`xpix, ypix, zpix`) and copy them to a 3-D float array, `inFileData`. Any exceptions thrown during this process are caught and the user informed via a `JOptionPane`.

Finally, the readFile() method creates a new object of the HiCube class:

```
hiCube = new HiCube( inFileData );
```

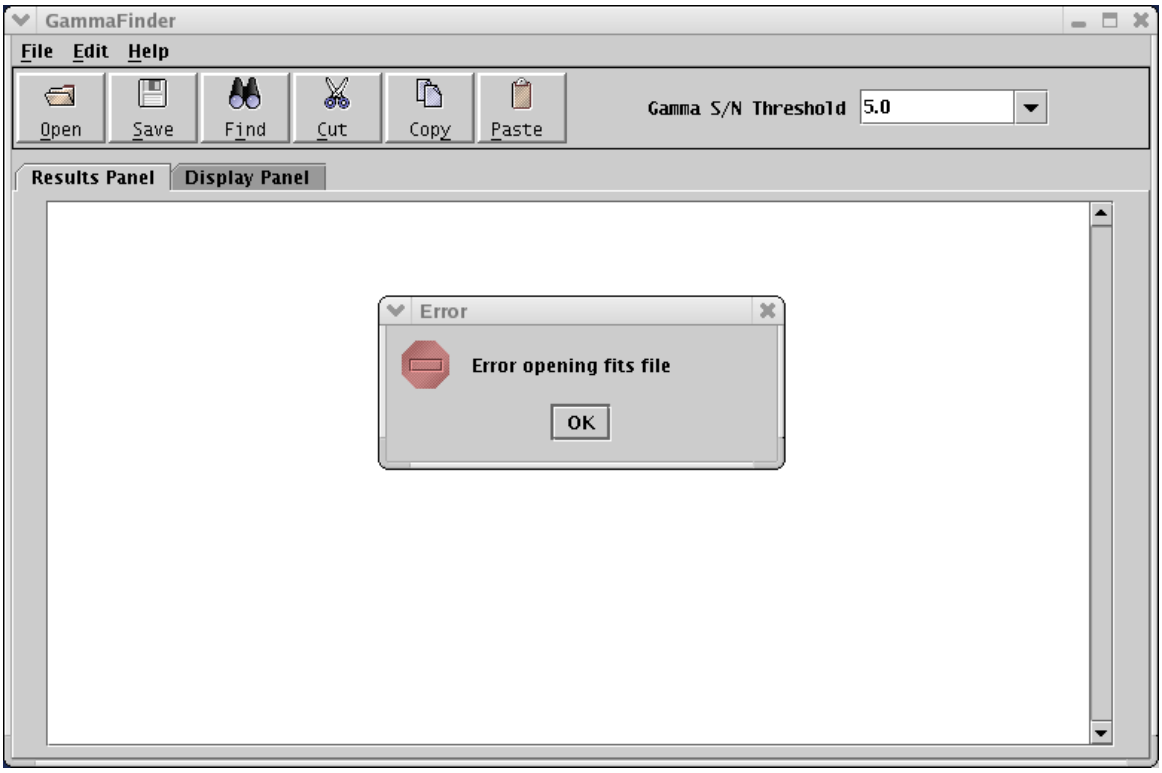


Figure 5.5. A JOptionPane informing the user that the file could not be opened.

The user is then informed of the successful completion of these activities via a `JOptionPane`.

5.4 The `setGVals()` method

Once a `HiCube` object (`hiCube`) has been successfully created, the `GuiFrame` object calls the `setGVals()` method to set the `gVal` instance variable of all the `Voxel` objects in the `voxArray` of `hiCube`.

This method is based within a triple nested loop in x, y and z voxels (using i, j and k as indices). Within this loop, the `gVal` instance variable is calculated for the `Voxel` object at the (i,j,k) position of the `voxArray` instance variable of `hiCube`. The following code fragment illustrates this.

```
for( int i = 0; i < xpix; i++)
   for( int j =0; j < ypix; j++ )
      for( int k = 0; k < zpix; l++)
         //code to set the gVal in voxArray[i][j][k]
```

The `gVal` at a given (i,j,k) position is, of course, the Gamma Statistic resulting from running a Gamma Test on the z-channel spectrum at that (i,j) position, using an 11 channel-wide window (centred at the k-position) and having a maximum number of nearest neighbours $p = 15$. The code to calculate this comprises two stages.

In the first stage, the values of $\gamma_M(q)$ and $\delta_M(q)$ are calculated for $1 \le q \le 15$. The following code fragment accomplishes this process.

```
for( int q = 1; q <= 15; q++ )
{
   deltaSum = 0;
   gammaSum = 0;

   for( int l = k - 5; l <= k + 5; l++)
   {
      if( q % 2 == 1 )
      {
         deltaSum += ( (float) ( ( q + 1 ) * ( q + 1 ) / 4 );
         posq = l + (q+1)/2;
         posl = l;
      }
      else
      {
         deltaSum += (q * q / 4);
         posk = l + q/2;
         posl = l;
      }

      if( posq < 0 )
         posq += zpix;
      if( posq >= zpix )
         posq -= zpix;
      if( posl < 0 )
         posl += zpix ;
      if( posl >= zpix )
         posl -=  zpix;

      gammaSum += (float)( (voxArray[i][j][posq].getVoxVal()
                            - voxArray[i][j][posl].getVoxVal() ) *
                         ( (voxArray[i][j][posq].getVoxVal()
                            - voxArray[i][j][posl].getVoxVal() ) );
   }
   deltaQ[q] = deltaSum / 11;
   gammaQ[q] = gammaSum / 22;
}
```

The outer loop is run 15 times, once for each nearest neighbour distance ($1 \le q \le 15$). At the end of each loop the calculated $\gamma_M(q)$ and $\delta_M(q)$ values are copied to the q-th elements of the `deltaQ` and `gammaQ` arrays. The inner loop is run 11 times, once for

each channel in the 11-channel wide window. The purpose of the inner loop is to calculate the

$$\sum_{i=1}^{M} |\boldsymbol{x}_{N[i,q]} - \boldsymbol{x}_i|^2 \quad \text{and} \quad \sum_{i=1}^{M} |y_{N[i,q]} - y_i|^2$$

values within the 11-channel window. The first of these is stored in the variable '`deltaSum`', the second in the variable '`gammaSum`'. There are two complicating factors in this summation:

1. By definition (see Section 3.2), $|\boldsymbol{x}_{N[i,q]} - \boldsymbol{x}_i|$ has value 1, 1, 2, 2, 3, 3, 4, 4 etc, with increasing q value since the 1st nearest neighbour channel and the second actually lie at the same distance (on either side of the input channel). Hence, $|\boldsymbol{x}_{N[i,q]} - \boldsymbol{x}_i|$ has to be determined using a different equation for odd and even q values.
2. If the centre channel in the current window is closer than 6 channels from either end of the spectrum, or if the current input position within the window is within 7 channels of either end of the spectrum then there are not enough channels to compute the Gamma Test within the whole window and/or to do the whole of the nearest neighbour calculation. The solution adopted is to 'wrap around' to the other end of the spectrum. The two indexes '`posl`' and '`posq`' implement this.

In the second stage, a linear regression is performed on the $(\gamma_M(q), \delta_M(q))$ pairs as stored in the float arrays `deltaQ` and `gammaQ`. The result of this regression is the Gamma Statistic. This is copied into the `gVal` instance variable of the `Voxel` object at the (i, j, k) position of the `voxArray` instance variable of `hiCube`. The following code extract performs these operations.

```
for ( int q = 1; q <= 15; q++)
{
   sumX += deltaK[q];
   sumX2 += ( deltaK[q] * deltaK[q] );
   sumY += gammaK[q];
   sumXY += ( deltaK[q] * gammaK[q] );
}

float gradient = (sumXY - (sumX * sumY / 15) ) / ( (sumX2) - ( sumX *
                                                    sumX / 15 )   );

float intercept = ( sumY - ( gradient * sumX ) ) / 15;

voxArray[i][j][k].setGVal( intercept );
```

The winGamma application uses a k-d tree data structure (Friedman et al. 1979) to implement the Gamma Test. This requires a single computation of *O(MlogM)* to compute the ($\gamma_M(q)$, $\delta_M(q)$) pairs. The method used to calculate these pairs in setGalVals() has $O(M^2)$. However, since we are using a small value of M (11) and the input data only has a dimensionality of 1, a satisfactory performance is still obtained (see Section 6.3).

5.5 The `setZChanQual()` method

Once the `setGVals()` method has been run, the `GuiFrame` object calls the `setZChanQual()` method to set the `zChanQual` Boolean array instance variable of `hiCube`. This method is a straightforward Java implementation of the `findBadZChans` prodecure (Section 3.6).

5.6 The `setH2H4H8Vals()` method

Once the `setZChanQual()` method has been run, the `GuiFrame` object calls the `setH2H4H8Vals()` method to set the `h2Val`, `h4Val` and `h8Val` instance variables of all the `Voxel` objects in the `voxArray` of `hiCube`.

The method is based within a double nested loop in x and y voxels. Within this loop, the `h2Val`, `h4Val` and `h8Val` instance variables are calculated for all the `Voxel` objects at the (i,j) specified by the loop. The following code fragment illustrates this.

```
for i = 0 to i < xpix; i++)
{
   for( int j = 0; j < ypix; j++ )
   {
      //code to set the h2Vals at this (i,j) position
      //code to set the h4Vals at this (i,j) position
      //code to set the h8Vals at this (i,j) position
   }
}
```

For example, the code within this double loop which calculates the `h2Val` instance variables is:

```
for(int k = 1; k < (zpix - 1) ; k = k + 2 )
{
```

```
    temp = (0.5 * voxArray[i][j][k].getVoxVal() ) +
           ( 0.25 * ( voxArray[i][j][k-1].getVoxVal() +
                                voxArray[i][j][k+1].getVoxVal() )  );
    voxArray[i][j][k].setH2Val( (float) temp );
}
```

5.7 The `setG2G4G8Vals()` method

Once the `setH2H4H8Vals()` method has been run, the `GuiFrame` object calls the `setG2G4G8Vals()` method to set the `g2Val`, `g4Val` and `g8Val` instance variables of all the `Voxel` objects in the `voxArray` instance variable of `hiCube`.

The method is based within a double nested loop in x and y voxels. Within this loop, the `g2Val`, `g4Val` and `g8Val` instance variables are calculated for all the `Voxel` objects at the (x,y) specified by the loop. Each of these is calculated in a similar way to that used by the `setGVals()` method, except that the relevant `h2Val`, `h4Val` and `h8Val` instances variables are used as appropriate.

5.8 The `setNoise()` method

Once the `setG2G4G8Vals()` method has been run, the `GuiFrame` object calls the `setNoise()` method to set the `noise` instance variable of `hiCube`. This method is a straightforward Java implementation of the `findNoise` procedure (Section 3.6).

Following the completion of this method, a `JoptionPane` informs the user that the `HiCube` object has been successfully created (Figure 5.6).

Once this modal dialog window is dismissed, the dimensions of `hiCube`, the rms noise and the channels numbers of the bad-channels are written to `textPanel`. This process makes use of the `getNoise` and `getZChanQual()` methods. This completes the sequence of operations activated by the user selected the ‘Open’ button or the ‘Open File’ menu item.

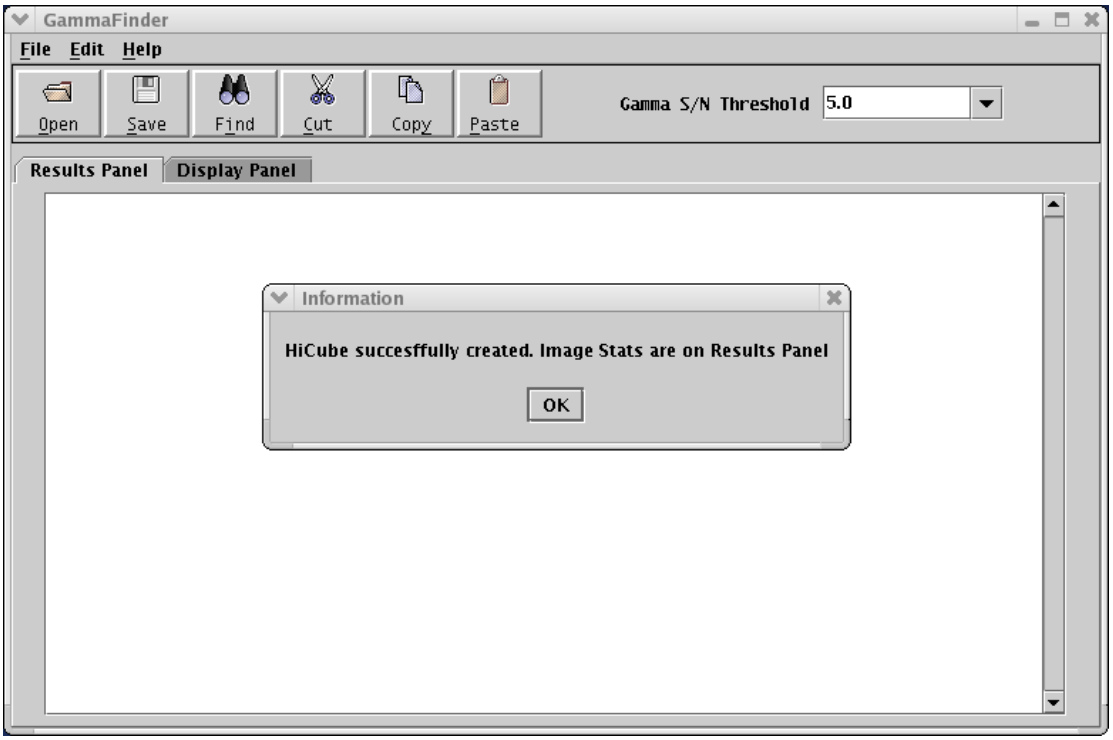


Figure 5.6. The GUI after the completion of the `setNoise()` method.

5.9 The `findSources()` method

When the user selects the 'Find' button or the 'Run Finder' menu item, the `findSources()` method of the `GuiFrame` class is called. The method firstly calculates a '`worsevar`' value defined to a factor of 5 greater than the square of the '`noise`' instance variable of `hiCube`.

The heart of this method is based within a double nested loop in the x and y voxels (using indices i and j). Within this loop, the z-channel gamma spectra (represented by the `g2Val`, `g4Val` and `g8Val` instance variables of the `Voxel` objects in the `voxArray` of `hiCube`) defined by the current (i, j) pair are searched for significant peaks in the $\emptyset$ values.

Firstly, the noise variance in the spectrum at the current (i, j) is evaluated by finding the median of the `gVal` instance variable at this (i, j). This is done by copying the set of `gVal` values into a float array, sorting this into ascending order (using the `sort()` method of the `Arrays` class) and choosing the middle value. The returned value is then compared to the `worsevar` value. Only if the median variance is less

than worsevar is a search conducted for sources at this (i, j) position. This test is done to exclude very noisy spectra from the galaxy finding process.

If the above test is met then a median value is found for each of the g2Val, g4Val and g8Val sets of values for this (i, j) position. These median values, denoted medianG2, medianG4 and medianG8 represent 'typical' values in each type of gamma spectrum and, hence, should be relatively unaffected by RFI, emission lines, etc. in the data.

A search is then made along each of the 3 types of gamma spectra at this (i, j) position. The following code extract illustrates this search along the gamma spectrum of g2Val values.

```
for( int k = 3; k < (zpix - 3) ; k = k + 2 )
{

if( ( hiCube.getVoxel(i,j,k).getG2Val() > (gamThres * medianG2) ) &&
(hiCube.getVoxel(i,j,k).getG2Val() > hiCube.getVoxel(i,j,k+2).getG2Val()) &&
(hiCube.getVoxel(i,j,k).getG2Val() > hiCube.getVoxel(i,j,k-2).getG2Val()) &&
(hiCube.getVoxel(i,j,k).getG2Val() > hiCube.getVoxel(i+1,j,k).getG2Val()) &&
(hiCube.getVoxel(i,j,k).getG2Val() > hiCube.getVoxel(i-1,j,k).getG2Val()) &&
(hiCube.getVoxel(i,j,k).getG2Val() > hiCube.getVoxel(i,j+1,k).getG2Val()) &&
(hiCube.getVoxel(i,j,k).getG2Val() > hiCube.getVoxel(i,j-1,k).getG2Val()) &&
(hiCube.getVoxel( i, j, k ).getH2Val() > 0) )
```

The g2Val at each k position (with a g2Val value) is examined. To be selected as the location of a possible source three criteria must be met. Firstly the g2Val must be a factor gamThres greater than the medianG2 value (i.e. have Gamma S/N gamThres). Secondly, the g2Val in this position must represent a peak in the g2Val distribution in all three dimensions (i, j, and k). Thirdly, the corresponding h2Val for this pixel must have a value > 0. This last condition eradicates the possibility of returning as a potential source a peak in the g2Val distribution which results from local RFI with negative intensity.

A similar process is used to find potential sources in the g4Val and g8Val data. A 3-dimensional Boolean array is used to note sources found in each of the sets of data such that sources found in the g2Val data will not also be returned from the g4Val data. Similarly, sources found in the g4Val data will not also be returned in the g8Val data.

Due to the smaller number of channels with a g8Val there is a correspondingly greater chance of the median value in a gamma spectrum being biased by a source in the data. A modal value (modmeanG8) is also derived from this data, using the ModMean static method of the Stats class. This ensures that a more reliable 'typical' g8Val is returned if a source is included in the data. To be selected as a possible source a g8Val value can be more than a factor of gamThres above the medianG8 or the modmeanG8 value.

All potential sources are written to textPanel along with the type of hanning spectrum in which the possible source was detected and the 'Gamma S/N'.

5.10 Viewing spectra

If the user selects the "Display Spectrum" button on the "Display Panel" then, firstly, the updateDispPars() method of the GUIFrame class is called. This method reads the values in the JTextField objects on dispPanel and copies these values to the xcoord and ycoord instance variables. If either or both of the input values are not in integer format or are outside the dimensions of the HiCube object, the user is informed via a modal dialog window and the displayFlag instance variable is set to 'false'.

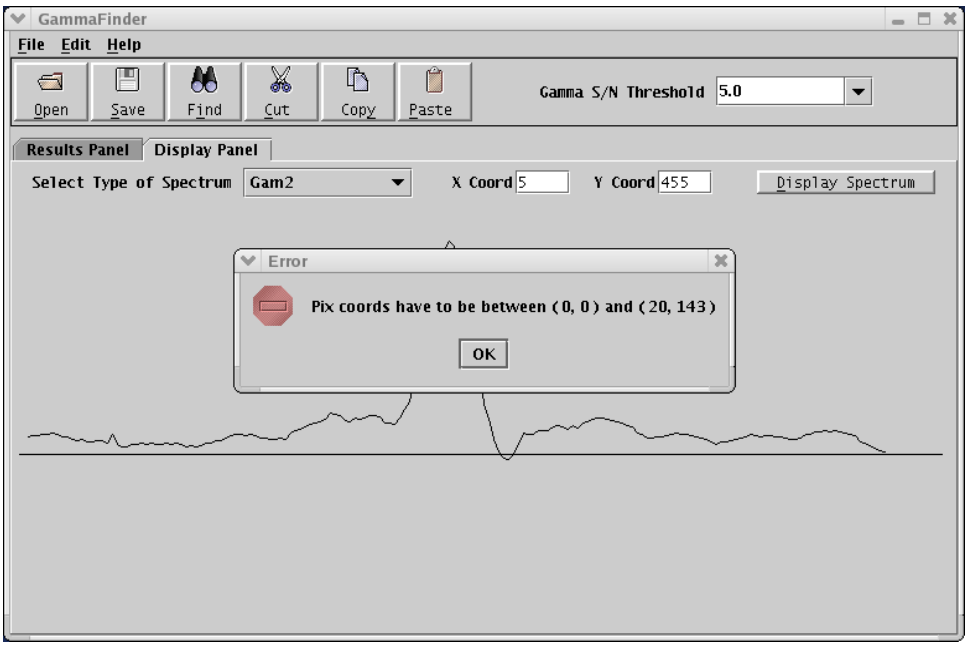


Figure 5.7. The GUI showing the JOptionPane displayed if the user-supplied X and/or Y Coord is outside the dimension of the current HiCube object.

This is illustrated in Figure 5.7. If these values are valid, then the method reads the user-selected spectrum type and updates the `specType` instance variable accordingly.

The `updateSPanel()` of the `DisplayPanel` class is then run on `dispPanel`. The current `voxArray`, `xcoord`, `ycoord` and `specType` values are supplied as parameters. This method displays the requested spectrum with an appropriate scaling on `dispPanel`.

5.11 Editing and saving results

The "Results Panel" comprises a `JTextArea` object (`textPanel`) on a `JScrollPane`. The contents of this can be edited, with the assistance of `cut()`, `paste()` and `copy()` methods which are implemented via buttons on the toolbar and items in the 'Edit' menu. If the user selects the 'Save' button or the 'Save Results' menu item then the `SaveResults()` method is called. This makes use of the `JFileChooser` class to save the current contents of `textPanel` to a text file. The contents of `textPanel` are written to a String using the `getText()` method and written to the selected output file using the `FileWriter` class. All exceptions generated by this process are caught and the user is informed via a `JOptionPane`.

5.12 Concluding remarks

The most important aspects of the implementation of the GammaFinder application have been described in this chapter. Appendix A presents a full listing of the code.

6. Results and Evaluation

6.1 Introduction

In this chapter the results are presented of an evaluation of GammaFinder. The developmental and testing process is reviewed in Section 6.2. Section 6.3 contains a discussion of the time and space complexity of the application. In Section 6.4, GammaFinder is evaluated in terms of its ability to detect real sources in HIJASS data.

6.2 The development and testing process

This section contains an overview of the development and testing process.

Firstly, a prototype application, GF.java, was created. This could read in a single spectrum (from a text file) and perform a moving window Gamma Test on it. It could also create Hanning smoothed spectra and run a moving window Gamma Test on these. Any of the resulting spectra could be examined on a `JFrame`. This prototype was used to test the algorithms later included in the `setGVals()`, `setH2H4H8Vals()` and `setG2G4G8Vals()` methods of the `HiCube` class. The algorithms were tested by comparing the returned results with those expected for the test input spectrum (calculated using *winGamma* and the equation for Hanning smoothing). When conducting a moving window Gamma Test with GF.java it is possible to set the maximum number of nearest neighbours, p, to be larger than the window width, something not possible with *winGamma*. Experiment showed this to be useful in providing an optimum peak in a gamma spectrum at the position of a discontinuity in the raw spectrum (see Section 3.4). GF.java was used to perform much of the analysis presented in Chapter 3 and to produce Figures 3.4 to 3.8.

The development process for GammaFinder itself essentially consisted of creating and testing Java code to implement the classes and methods described in Chapter 4.

The `Voxel` class was created first. This was tested by the use of a `VoxelTest` class. This creates a `Voxel` object and then sets the values of the other instance variables (i.e. testing the `setH2Val()` etc. methods). The test class also tests the 'get' methods (i.e. `getGVal` etc.).

The HiCube class was then created. This was tested with a HiCubeTest class. This creates a HiCube object, passing a 3-dimensional float array to the HiCube constructor, and then calling each of the 'set' methods of the class (i.e. setGVals, setH2H4H8Vals etc.) to set the instance variables of each Voxel object in the voxArray instance variable as well as the noise and zChanQual instance variables. Each of the 'get' methods (i.e. getVoxArray() etc.) is also tested.

Next, the GuiFrame class was created. The basic GUI was created first with each of the options in the ActionPerformed inner class (i.e. relating to a menu item or tool bar item) initially being set to "//do nothing". These options were then filled in and tested one by one.

The response to the 'Open File' menu item was the first to be coded. This involved coding the readFile() method. The techniques of handing and reporting exceptions relating to problems with opening and reading the FITS file (Section 5.3) were tested using a combination of valid and invalid input files.

Next, the response to the 'Display Spectrum' button was coded. This included creating the DisplayPanel class and the setGamThres() and updateDispPars() methods. The whole process was tested using real HIJASS datacubes. The methods of handling exceptions caused by the user supplying ineligible xcoord, ycoord or gamThres values (Section 5.10) were tested by deliberately choosing incorrect values.

Next, a JTextArea object was added to the JTabbedPane on the GUI and the cut(), paste() etc. methods added to the relevant sections of the ActionPerformed inner class. The saveResults() method was then coded. These were all tested by writing, editing, and annotating text onto the JTextArea object and saving that text to a file.

Finally, the findSources() method was coded. The results of running this on a real datacube are described in Section 6.4.

6.3 Performance measurement and optimisation

Following the coding and testing described in Section 6.2, measurements were made of the time complexity and space complexity of the application. This analysis was undertaken on a PC with a 512 Mbytes RAM running Red Hat Linux.

The time complexity of the application was investigated by measuring the difference between the start time and finish time of the main methods, utilising the `System.currentTimeMillis()` method. Table 6.1 shows the time in milliseconds taken for the main methods to execute for a variety of different datacube sizes.

	Cube Size		
	10x10x1024	20x20x1024	40x40x1024
`readFile()`	72	255	788
`setH2H4H8Vals()`	24	38	105
`setGVals()`	5413	19234	79479
`setNoise()`	54	119	409
`setZChanQual()`	49	139	640
`setG2G4G8Vals()`	3745	14826	60789
`findGals()`	75	356	845

Table 6.1. The run time (in milliseconds) for the main methods of GammaFinder for datacubes of various sizes.

Clearly `setGVals()` and `setG2G4G8Vals()` account for the vast majority of the run time of the application. These both run in O(n) time (where n is the total number of voxels in the datacube). This is expected since they both involve a triple nested loop running through every voxel in the datacube (see Sections 5.4 and 5.7). It was, therefore, decided to optimise these two methods in a bid to reduce run time.

The optimisation analysis identified several possible ways of improving the efficiency of the code, including the use of the autoincrement operator and the elimination of common subexpressions. The static method `Math.pow()` was being called twice

within the innermost loop in both methods. A fragment of the original code of setGVals() follows:

```
deltaSum += ( (float) Math.pow( ( ( q + 1 ) / 2 ), 2) );
```

This meant that, for example, for a 40x40x1024 datacube, the `Math.pow()` method was being called more than 3 million times. It was decided to code this without calling the static method, e.g. as:

```
deltaSum += ( (float) ( ( q + 1 ) * ( q + 1 ) / 4 );
```

This had the effect of reducing the run time of the two methods by more than a factor of 4. Table 6.2 shows the time for the main methods of the application to execute in a 30x30x1024 datacube before and after the code optimisation.

	before optimisation	after optimisation
`setGVals()`	45115	10545
`setG2G4G8Vals()`	34389	8122

Table 6.2. The run time (in milliseconds) for `setGVals()` and `setG2G4G8Vals()` on a 30x30x1024 datacube before and after code optimisation.

Following code optimisation, measurements were made of the amount of memory GammaFinder needs to run to completion. Java's garbage collection system allocates a fixed amount of memory when the JVM is started. This can be set of startup, e.g. the following command will set the maximum amount of memory to 200 megabytes: `java –Xmx200m GammaFinder`

When this amount of memory is used up, an `'Out Of Memory'` error occurs.

The memory requirements of GammaFinder as a function of cube size are presented in Table 6.3. Column 2 contains an estimation of the memory required to store the `voxArray` instance variable for the datacube sizes of Column 1 (this is the major data storage requirement). Column 3 lists the minimum amount of memory required

to run a datacube through GammaFinder without an 'Out Of Memory' error occurring.

Cube Size	voxArray Size (Mbytes)	Minimum Memory (Mbytes)
10x10x1024	3	7
20x20x1024	13	21
30x30x1024	28	47
40x40x1024	50	83
50x50x1024	78	129
90x90x1024	253	413

Table 6.3. The memory requirements of GammaFinder for different datacube sizes.

Table 6.3 implies that the minimum memory requirement for a datacube can be estimated by multiplying the voxArray size by a factor of 1.64. This implies that the largest cube size which could be run on a PC with 512 Mbytes RAM would be around 100x100x1024. Larger cubes would have to make extensive use of virtual memory (swapping). A typical full HIJASS cube has dimensions 150x150x1024. Such a cube would have to be split into 2 or 3 cubes to be analysed by GammaFinder on a PC with 512 Mbytes RAM. PCs are now becoming available with up to 1.2 Gbytes of memory. It seems likely that, in the near future, machines will be commonly available with sufficient memory to process whole HIPASS / HIJASS cubes in one go.

The total run time of GammaFinder as a function of datacube size was estimated by using the unix 'time' command. The results are presented in Table 6.4. Column 1 lists various datacube dimensions. Column 2 lists the fraction of a whole datacube (considered to be one with dimensions 150x150x1024) each of these represents. Column 3 lists the estimated memory requirements (see above). Column 4 lists the measured run time.

Cube Size	% of whole cube	Memory Requirements	run time (sec)
47x47x1024	10%	144Mbytes	78sec
75x75x1024	20%	190Mbytes	160sec
94x94x1024	40%	456Mbytes	336sec

Table 6.4. The total run time of GammaFinder as a function of cube size.

The total run time scales as O(n) for cube sizes up to 94x94x1024. This is expected since the slowest methods scale as O(n). The need to use swap space resulted in such a

huge degradation in run time that it did not prove feasible to run GammaFinder on datacubes much larger than 94x94x1024 on the test PC. It is far more efficient to split larger cubes into several smaller cubes and process each separately.

The run times from Table 6.4 are acceptable for processing HIJASS / HIPASS datacubes. If split into several smaller cubes, a whole datacube cube could be processed in around 15 minutes. Compared to the time taken to observe HIJASS / HIPASS data and to reduce the data into datacube form (Barnes et al. 2001), 15 minutes per datacube represents a very small extra amount of data processing time.

6.4 Evaluating GammaFinder's performance on real data

This section contains an evaluation of GammaFinder in terms of its ability to detect real sources in HIJASS / HIPASS data. This is done by comparing the results of running GammaFinder on a HIJASS datacube with the contents of the published HIJASS catalogue (Lang et al. 2003) from the same datacube. The HIJASS cube centred at Right Ascension=$04^h\ 32^m$ and Declination=74^o was chosen for the evaluation. Within HIJASS naming nomenclature, this cube is denoted 'p74cube4'.

A brief review of the selection techniques used to compile the HIJASS catalogue and a discussion of the contents of the catalogue from p74cube4 is given in Section 6.4.1.

A key question to be asked is what fraction of real sources identified by visual inspection of a datacube or by the use of an existing automatic finder are also selected by GammaFinder? This is considered in Section 6.4.2.

Another important question concerns how the number of real sources compared to spurious sources changes with Gamma S/N. This is considered in Section 6.4.3.

The other major question is whether GammaFinder can identify real sources not identified by previously used methods. This is considered in Section 6.4.4.

In Section 6.4.5, the detection limits of GammaFinder are explored.

6.4.1 The HIJASS catalogue

The published HIJASS catalogue (Lang et al. 2003) was created by a combination of visual inspection of the datacubes and using the PolyFind automatic finding algorithm (see Section 2.7). Any 'possible' sources were subjected to 'narrow-band follow-up'. Only those confirmed were included in the catalogue. Hence, this catalogue forms an ideal basis against which to test the effectiveness of GammaFinder.

The 19 galaxies found in cube p74cube4 by Lang et al. and included in the HIJASS catalogue are listed in Table 6.5. The values for S_{int}, S_{pk}, V_o and W20 were derived using tasks from the MIRIAD software package (Sault et al. 1995).

HIJASS NAME	R.A.	DEC.	S_{int} Jy kms^{-1}	S_{pk} Jy	V_o kms^{-1}	W20 kms^{-1}	CLASS	COUNTERPART
J0545+73	05:45:19.9	73:36:41	4.7±0.5	68	1053	127	ID	KUG 0539+735
J0545+72	05:45:26.1	72:22:07	10.2±0.5	78	1080	199	ID	UGC 03343
J0556+75	05:56:28.7	75:18:40	23.7±0.9	228	811	146	ID	UGC 03371
J0602+73	06:02:25.1	73:04:46	18.3±0.6	224	1095	119	ID	UGC 03384
J0610+71	06:10:23.0	71:23:17	16.0±0.7	87	1270	268	ID	UGC 03403
J0615+71	06:15:24.5	71:06:29	8.9±0.3	56	3981	325	ID	UGC 03422
J0619+75	06:19:15.8	75:01:14	1.7±0.4	41	1324	85	ASS	J06201+7458
J0619+78	06:19:20.7	78:14:56	38.3±1.4	368	838	209	ID	NGC 2146
J0711+71	07:11:32.4	71:49:40	21.5±0.8	99	3144	321	ID	UGC 03697
J0713+73	07:13:28.8	73:29:30	10.5±0.6	105	2702	183	ID	UGC 03730
J0716+75	07:16:10.9	75:43:42	9.7±0.5	138	1118	96	ID	UGC 03739
J0721+74	07:21:20.9	74:18:45	1.5±0.5	61	966	77	PUG	
J0722+77	07:22:12.5	77:49:01	7.1±0.6	67	2643	172	ID	UGC 03794
J0730+72	07:30:35.1	72:29:29	12.3±0.9	108	2620	223	ID	UGC 03864
J0736+74	07:36:12.5	74:25:10	6.6±0.6	70	3776	221	ID	UGC 03906
J0736+73	07:36:39.4	73:41:59	13.5±0.6	111	943	188	ID	UGC 03909
J0744+72	07:44:58.4	72:47:54	10.1±0.6	201	2476	83	ID	UGC 03975
J0750+74	07:50:14.4	74:19:57	14.6±0.9	84	3943	298	ID	UGC 04028
J0751+72	07:51:46.1	72:59:37	10.6±1.1	119	3470	156	ID	NGC2441

Table 6.5. The parameters of sources within the HIJASS catalogue from p74cube4.

The 'CLASS' column in Table 6.5 denotes whether the HI emission line is associated with a previously catalogued galaxy or not. This was determined by reference to the NASA Extragalatic Database (NED – http://nedwww.ipac.caltech.edu). NED is the standard master list of identified extragalactic objects, including accurate positions and redshifts of all published sources. If there is an object within NED which matches the HIJASS source in both position (defined as being within 6 arcmin, i.e. 2.5**?** of the positional accuracy of HIJASS) and cz (defined as being within 100 km s^{-1}) then this is listed as 'ID' (i.e. Identification). We may be relatively certain that these HIJASS sources are HI detections of the galaxy included in NED. Although these galaxies have been detected and catalogued before, the HIJASS data represents the first time many of these sources have been detected in HI.

If there is an object within NED which is spatially coincident with the HIJASS source (i.e.within 6 arcmin) but for which no redshift (and, hence, no cz) is listed in NED, then this is listed as 'ASS' (i.e. Association). Such a HIJASS source may be

associated with the catalogued galaxy. However, since there is no redshift for the catalogued galaxy we cannot be certain that the HIJASS source is an HI detection of the catalogued galaxy. A program of obtaining optical redshifts for these 'ASS' galaxies is being undertaken in order to determine whether each is the optical counterpart of the HIJASS source.

Finally, if there is no object within NED which could be spatially coincident with the HIJASS source then the 'CLASS' column contains the classification 'PUG' (i.e. Previously Uncatalogued Galaxy). These HIJASS sources are definitely not associated with a previously catalogued galaxy. Often a study of the Digital Sky Survey shows a likely optical counterpart to the HIJASS source (within the positional uncertainty of the HIJASS detection). These are generally faint or low surface brightness galaxies and have, thus, escaped being catalogued before (i.e. exactly the sort of galaxies HIPASS and HIJASS were conceived to discover – see Section 2.2). However, several instances are known where no galaxy can be seen on the Digital Sky Survey, despite the HI detection being unambiguous. These 'invisible' galaxies are arguably the most fascinating discoveries of the HIPASS and HIJASS surveys. Details of three of these have been published (Kilborn et al. 2000, Ryder et al. 2001, Boyce et al. 2001).

6.4.2 Can GammaFinder detect those galaxies in the HIJASS catalogue ?

GammaFinder was run on p74cube4 with `gamThres`=2.0. Table 6.6 shows the results from GammaFinder for these 19 objects. GammaFinder selected all but one of the 19 sources with a Gamma S/N >6.0. One source, J0619+75, was detected with a significantly lower Gamma S/N than any other galaxy (2.46). This source actually has the lowest peak flux (41 mJy) and second to lowest integrated flux (1.75 Jy km s^{-1}) of any source in the whole HIJASS catalogue. This source was selected from a visual inspection and confirmed by narrow-band follow-up. It was not selected by the PolyFind automatic finding algorithm.

HIJASS NAME	Hann FWHM	Gamma S/N
J0545+73	H2	6.57
J0545+72	H4	13.95
J0556+75	H2	82.17
J0602+73	H2	97.94
J0610+71	H4	12.53
J0615+71	H8	14.36
J0619+75	H2	2.46
J0619+78	H2	166.87
J0711+71	H4	19.59
J0713+73	H2	12.72
J0716+75	H2	31.47
J0721+74	H8	8.08
J0722+77	H4	11.46
J0730+72	H8	18.89
J0736+74	H4	7.98
J0736+73	H8	29.49
J0744+72	H2	33.57
J0750+74	H4	9.57
J0751+72	H2	6.16

Table 6.6. GammaFinder results for the 19 galaxies within p74cube4 included in Lang et al.'s HIJASS catalogue.

Table 6.6 suggests that GammaFinder will be able to find almost all sources currently included in the HIJASS catalogue provided a `gamThres` value of around 6.0 is used. Nonetheless, the human eye may still be able to find the odd source missed by GammaFinder and other finding algorithms.

6.4.3 The number of possible sources found by GammaFinder as a function of Gamma S/N

Table 6.7 lists the number of possible sources selected by GammaFinder from p74cube4, as a function of Gamma S/N.

The number of possible sources starts to increase significantly at Gamma S/N < 8.0 and particularly sharply at a Gamma S/N < 6.0. Ideally, we want to know what fraction of sources at each Gamma S/N are real and what fraction are spurious. This can only be accurately determined by undertaking narrow-band follow-up observations of all possible sources returned by GammaFinder in a representative set of data. If we could subject all 180 sources found with Gamma S/N > 5.0 to narrow-band follow-up observations, then we could determine what fraction are real and what fraction are spurious down to Gamma S/N = 5.0. Such an investigation would require about 5 days of telescope time.

Gamma S/N	No. of possible sources
4.0 → 5.0	369
5.0 → 6.0	93
6.0 → 7.0	36
7.0 → 8.0	23
8.0 → 9.0	6
9.0 → 10.0	3
10.0 → 11.0	0
11.0 → 12.0	4
12.0 → 13.0	3
> 13.0	12

Table 6.7. The number of possible sources returned by GammaFinder (from p74cube4) as a function of Gamma S/N.

6.4.4 Can GammaFinder detect galaxies not included in the HIJASS catalogue ?

Table 6.6 suggests that most possible sources returned by GammaFinder with GammaS/N < 6.0 will be spurious. Therefore, for this evaluation, GammaFinder was run over p74cube4 with `gamThres` = 6.0.

Initially, besides the 18 sources with Gamma S/N > 6.0 already included in the HIJASS catalogue, GammaFinder returned a further 69 possible sources. A visual inspection of the spectra relating to these on the "Display Panel" of GammaFinder enabled 38 to be rejected as obviously false, primarily being due to negative RFI 'spikes' in channels adjacent to the selected 'source' or places where double-peaked

sources were being returned twice. Hence, an additional 31 possible sources, which could not be obviously rejected as false, were selected by GammaFinder. Table 6.8 lists the output results from GammaFinder for these 31 possible sources (labelled GF1 to GF31).

Table 6.9 presents derived parameters for the 31 possible sources. These were obtained in the same way as for those galaxies currently included in the HIJASS Catalogue. As noted above, the only way to ascertain with certainty whether each of these possible sources is real is to undertake a narrow-band follow-up observation. Nonetheless, in lieu of this, it is still possible to estimate the likelihood of the possible sources being real by comparing their coordinates to the NASA Extragalactic Database (NED) and the Digital Sky Survey (DSS).

SOURCE	XPIX	YPIX	ZPIX	HANN TYPE	GAMMA S/N
GF1	12	16	455	H8	6.36
GF2	26	32	567	H8	8.07
GF3	34	95	455	H8	6.51
GF4	45	21	559	H8	8.42
GF5	45	34	543	H8	7.74
GF6	47	9	551	H8	9.81
GF7	48	124	455	H8	7.01
GF8	49	21	559	H8	6.23
GF9	53	11	695	H8	7.11
GF10	54	9	703	H8	11.07
GF11	54	15	559	H8	6.94
GF12	59	41	539	H4	7.03
GF13	65	23	519	H8	24.92
GF14	65	35	527	H8	7.31
GF15	69	110	471	H8	6.76
GF16	78	17	471	H8	9.50
GF17	81	11	511	H8	7.25
GF18	93	18	663	H8	6.40
GF19	96	34	495	H8	6.17
GF20	104	31	471	H8	26.33
GF21	111	55	679	H8	8.70
GF22	113	64	511	H8	6.10
GF23	123	20	559	H8	7.00
GF24	124	19	567	H8	7.08
GF25	131	13	471	H8	7.20
GF26	133	112	423	H4	6.06
GF27	136	14	551	H8	7.18
GF28	138	19	711	H8	6.95
GF29	139	13	543	H8	7.98
GF30	139	44	575	H8	6.48
GF31	142	13	543	H8	7.83

Table 6.8. The results of running GammaFinder over p74cube4 with `gamThres` = 6.0.

The 'COUNTERPART' column of Table 6.9 shows the result of comparing each of the 31 sources with NED. 10 of the 31 possible sources are coincident with a previously catalogued galaxies in both position and velocity. These are listed as 'ID' in Table 6.9. These are undoubtedly real. Figure 6.1 shows DSS images of these 10 galaxies, along with their HI spectra. Hence, even without further follow-up work, we can be certain that GammaFinder has added at least another 10 galaxies to the 19 in p74cube4 already in the HIJASS catalogue, i.e. at least another 50%. This in itself is a remarkable result.

Of the other 21 possible sources in Table 6.9, 14 have a galaxy spatially coincident (within the positional uncertainty of HIJASS) within NED but this galaxy currently has no measured redshift. Hence, these are classified as 'ASS'. However, a study of the DSS suggests that, in most cases, the catalogued galaxy appears to be too distant to be the optical counterpart to the HIJASS source. Nonetheless, in 3 cases (GF2, GF3 and GF14) the catalogued galaxy does look (in terms of morphology and distance) like a likely optical counterpart to the HIJASS source. Figure 6.2 shows the DSS images of these three galaxies, along with the HI JASS spectra of GF2, GF3 and GF14.

SOURCE	R.A.	DEC.	S_{int} Jy kms^{-1}	S_{pk} mJy	V_o kms^{-1}	W20 kms^{-1}	COUNTERPART	
GF1	07:46:31.0	69:59:27	1.7±0.9	54	4101	112	ID	UGC03984
GF2	07:37:44.4	71:16:38	4.7±0.7	46	2532	249	ASS	
GF3	07:44:01.9	75:29:13	8.9±0.9	77	4023	333	ASS	
GF4	07:21:09.4	70:39:00	7.6±1.2	61	2541	394	ASS	
GF5	07:22:30.7	71:34:50	13.0±0.9	72	2861	311	ID	UGC 03804
GF6	07:17:54.0	69:58:22	11.0±1.2	74	2691	309	NOF	
GF7	07:35:34.1	77:33:02	4.9±0.9	56	4016	345	NOF	
GF8	07:18:12.8	70:45:23	7.0±1.0	71	2700	312	ASS	
GF9	07:13:23.5	70:06:09	6.6±1.1	59	782	345	ASS	
GF10	07:12:34.0	70:00:55	6.7±1.3	69	722	358	ASS	
GF11	07:13:44.4	70:25:02	5.8±1.0	76	2597	429	ASS	
GF12	07:11:47.6	72:10:41	5.7±0.7	64	2884	149	ID	UGC 03701
GF13	07:04:50.9	71:01:49	10.8±0.8	86	3226	238	ID	UGC 03644
GF14	07:07:28.8	71:54:09	5.0±0.9	77	2980	238	ASS	
GF15	07:08:08.2	76:49:16	6.6±0.9	41	3876	310	ASS	
GF16	06:55:19.7	70:43:23	7.3±1.0	61	3858	351	ID	UGC03575
GF17	06:52:11.3	70:15:56	4.3±0.7	59	3301	177	NOF	
GF18	06:42:34.3	70:43:36	3.3±0.6	44	1267	151	ASS	
GF19	06:39:41.6	71:47:17	2.3±0.6	73	3750	135	NOF	
GF20	06:32:48.5	71:31:44	16.3±0.6	85	3584	135	ID	UGC 03474
GF21	06:24:52.2	73:09:00	8.3±0.5	87	1046	176	ID	UGC 03453
GF22	06:22:29.9	73:44:14	4.4±0.6	49	3329	177	ID	UGC 03444
GF23	06:18:16.5	70:42:37	3.1±0.7	45	2583	162	NOF	
GF24	06:17:34.5	70:38:08	3.6±0.9	55	2574	172	NOF	
GF25	06:12:32.4	70:11:48	5.0±0.7	83	3827	147	ID	UGC03415
GF26	05:53:28.0	76:38:24	4.7±0.5	49	4390	205	ID	UGC03364
GF27	06:08:41.6	70:11:15	6.6±0.8	52	2699	355	ASS	
GF28	06:06:08.2	70:27:33	4.2±0.9	87	581	198	ASS	
GF29	06:07:19.9	70:07:48	8.3±1.1	62	2701	419	ASS	
GF30	06:01:27.0	72:08:55	6.4±0.9	49	2542	348	ASS	
GF31	06:04:20.9	70:02:31	6.3±1.1	60	2624	363	NOF	

Table 6.9. Derived parameters for the 31 new possible sources found by GammaFinder in p74cube4.

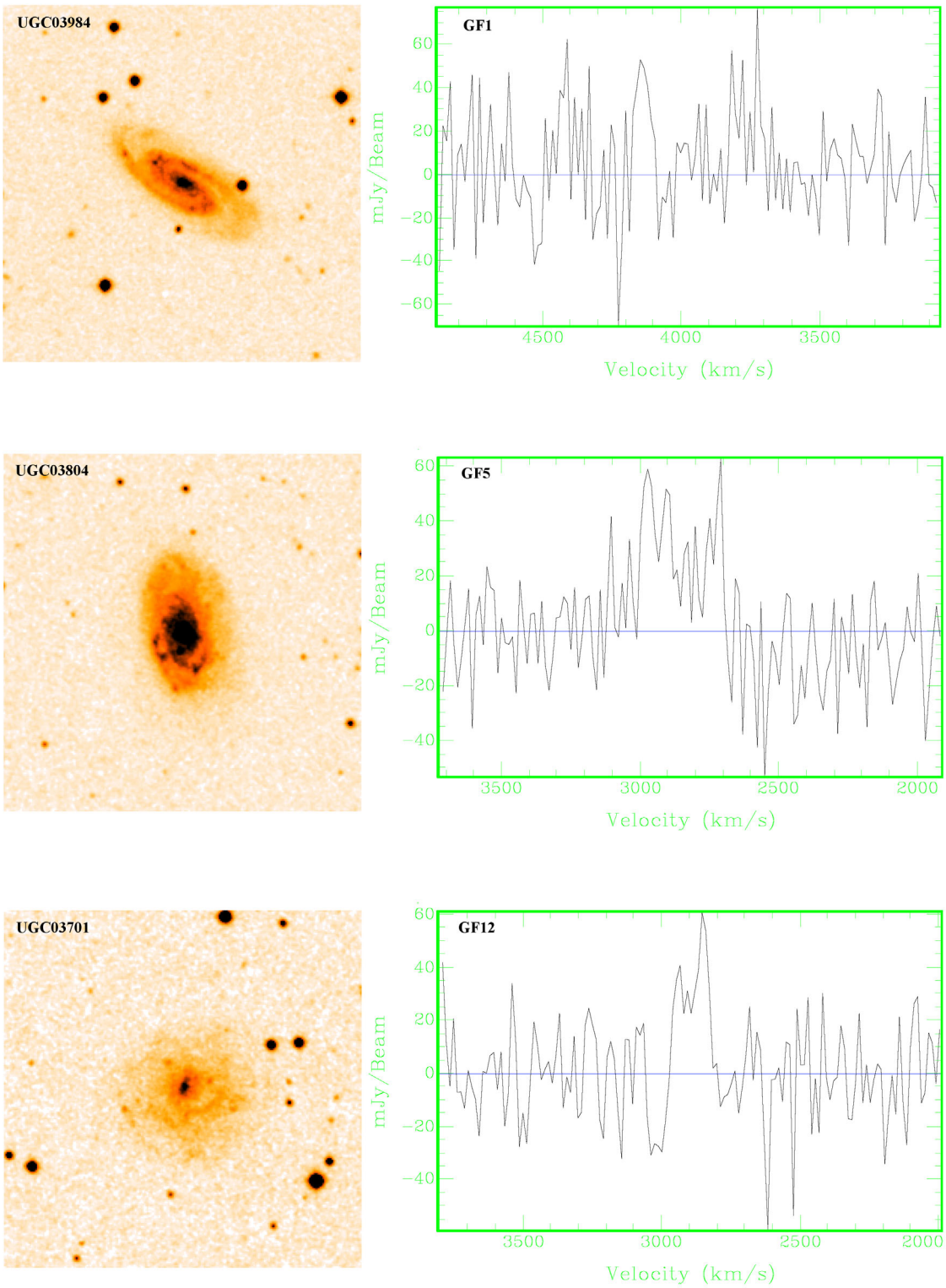


Figure 6.1. Digital Sky Survey images and HIJASS spectra for the 10 new 'ID' sources found by GammaFinder in p74cube4. The DSS images are 4arcmin x 4arcmin.

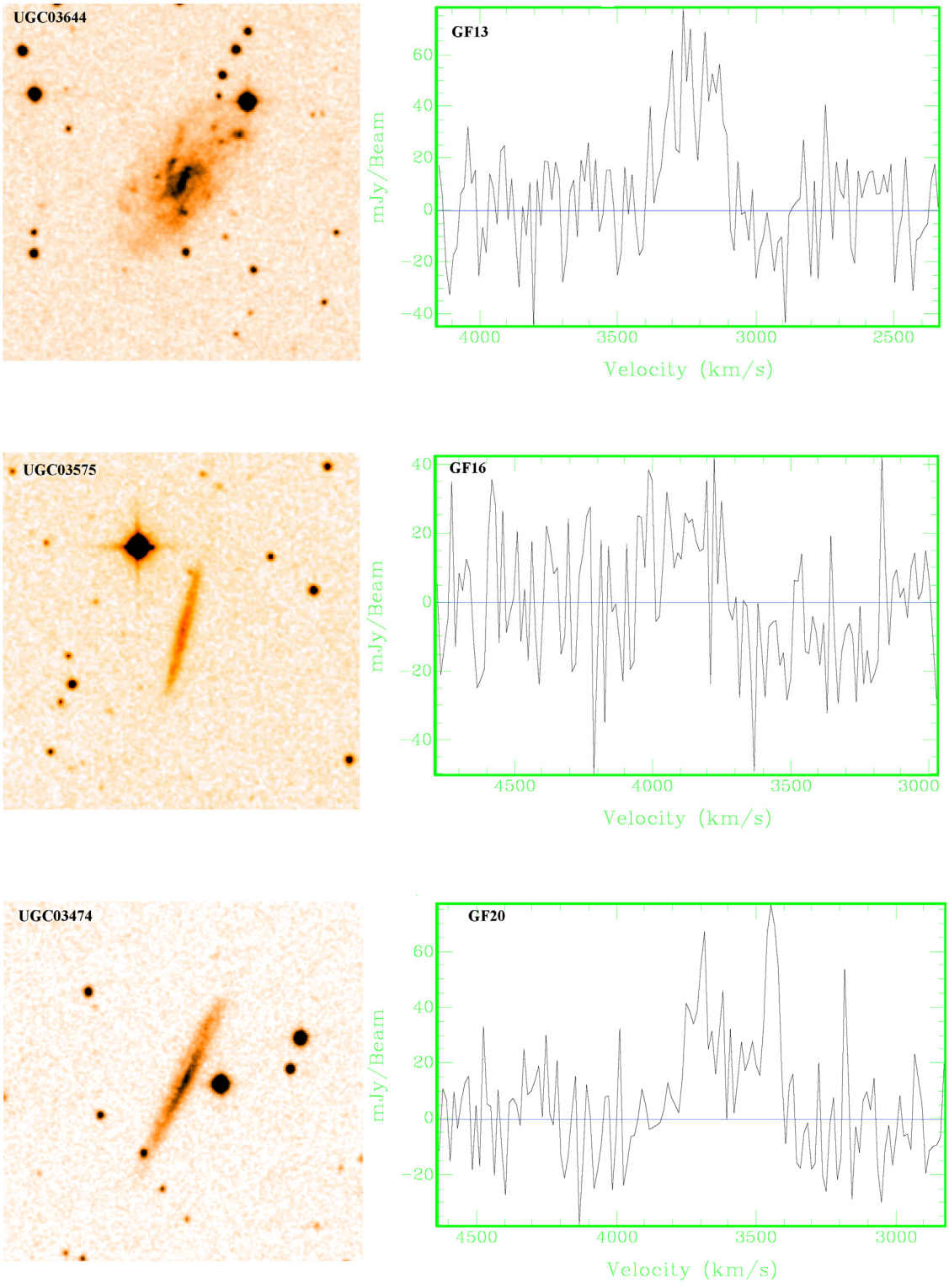
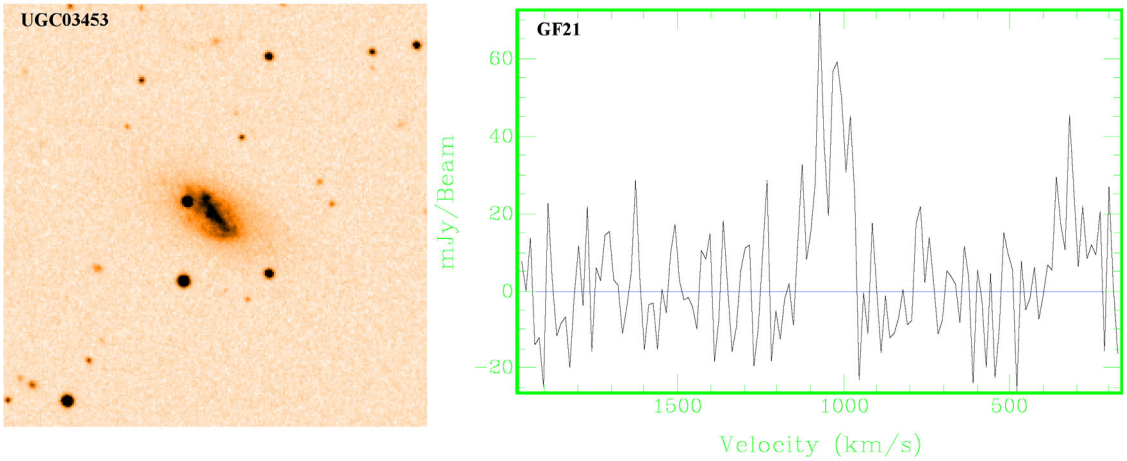
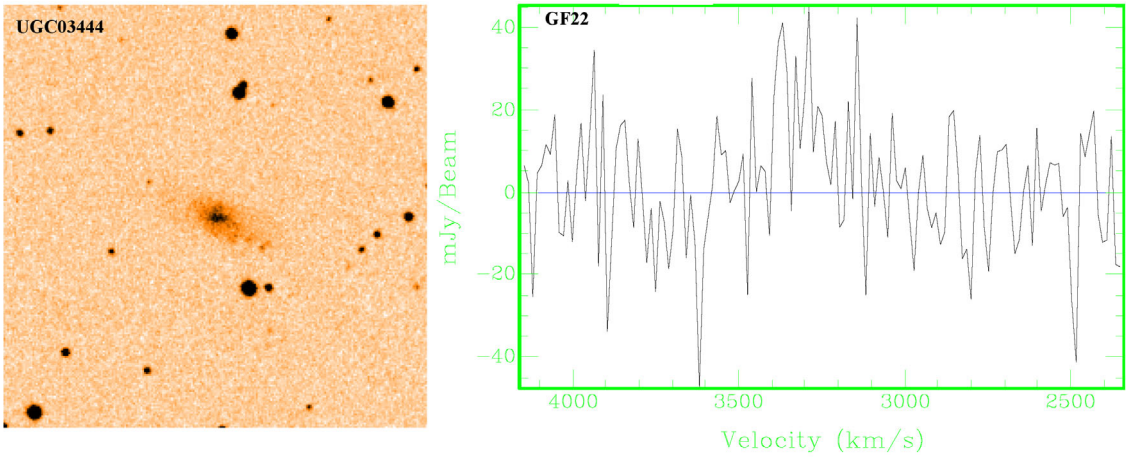


Figure 6.1. Continued.





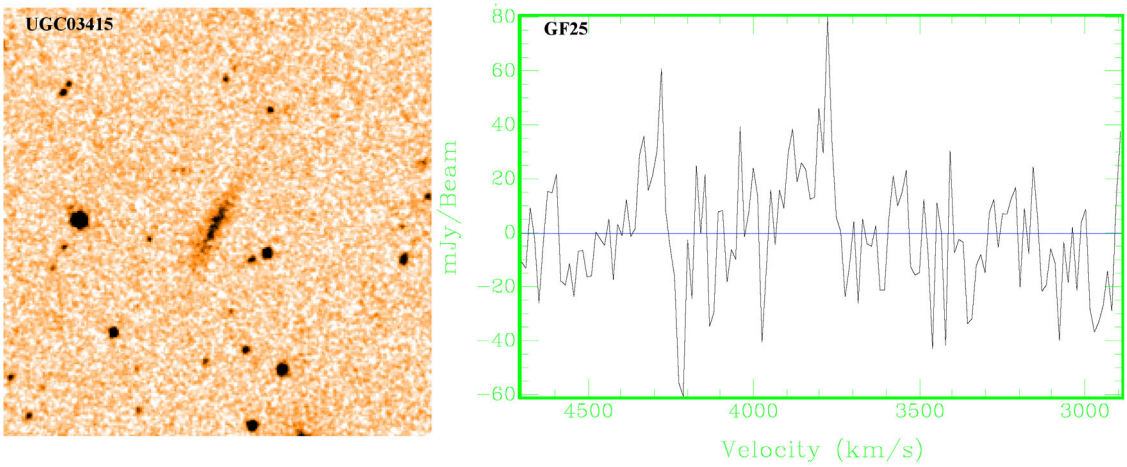


Figure 6.1. Continued.

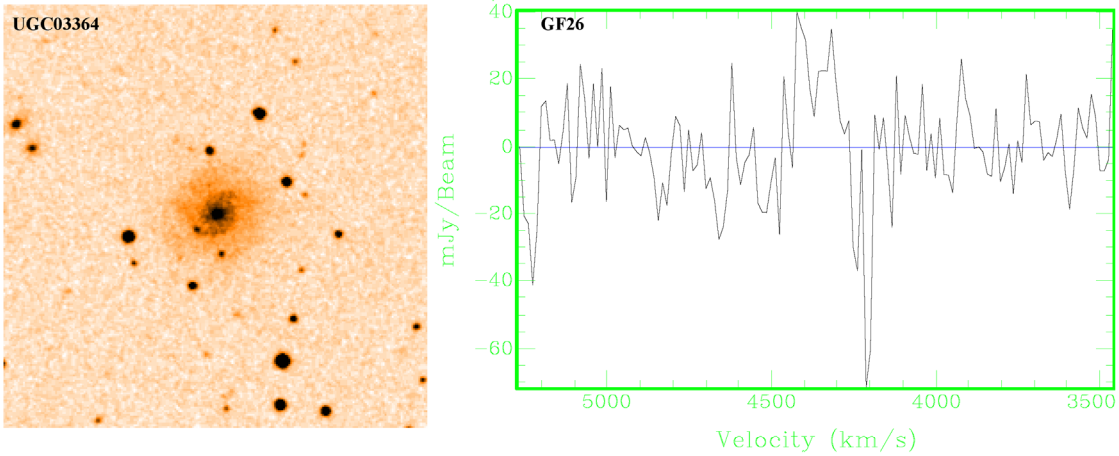


Figure 6.1. Concluded.

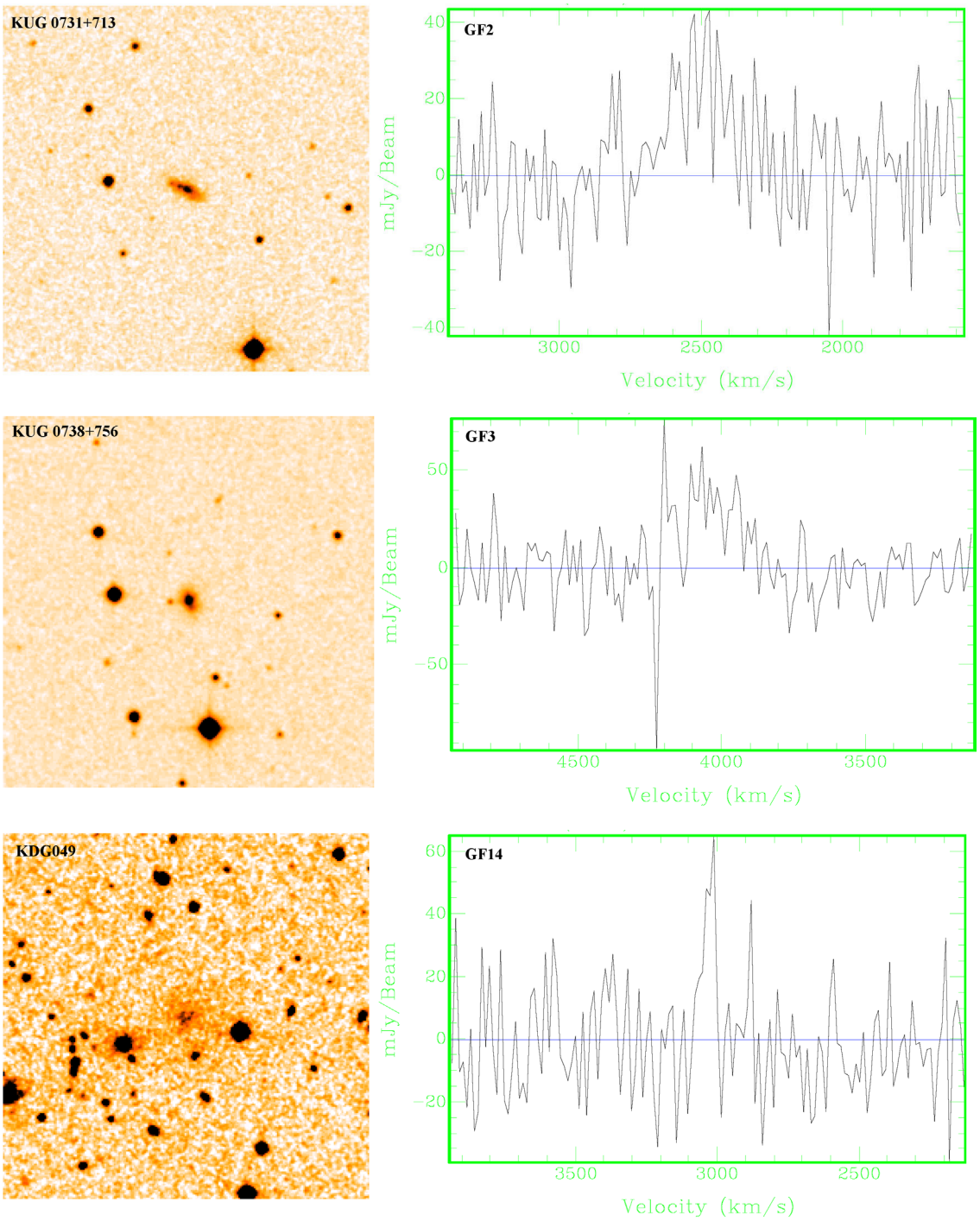


Figure 6.2. DSS images of the three 'ASS' galaxies which appear likely to be the optical counterparts of a GammaFinder possible source, along with the HIJASS spectra of the GammaFinder possible source.

There are 7 possible sources in Table 6.9 which have no galaxy spatially coincident within NED. They are classed here as NOF ('No Object Found') since, without narrow-band follow-up observations, it would be wrong to imply that they are definitely galaxies. A study of the DSS at the positions of the NOFs does, however, show a galaxy spatially coincident with one of these, GF31. This galaxy (not previously catalogued) looks like a very plausible optical counterpart to the HIJASS possible source. Figure 6.3 shows the DSS image of this galaxy, along with the HIJASS spectrum of GF31.

There are 17 possible sources in Table 6.9 for which a plausible counterpart cannot be found in NED or seen on the DSS. It is likely that most of these possible sources are not real. As noted above, there are several known examples of HIJASS/HIPASS sources which have no obvious optical counterpart. There may be further examples of these types of object within these 17 possible sources. They should certainly be subjected to narrow-band follow-up to ascertain if they are real or not. The HIJASS spectra of these 17 possible sources are presented in Figure 6.4.

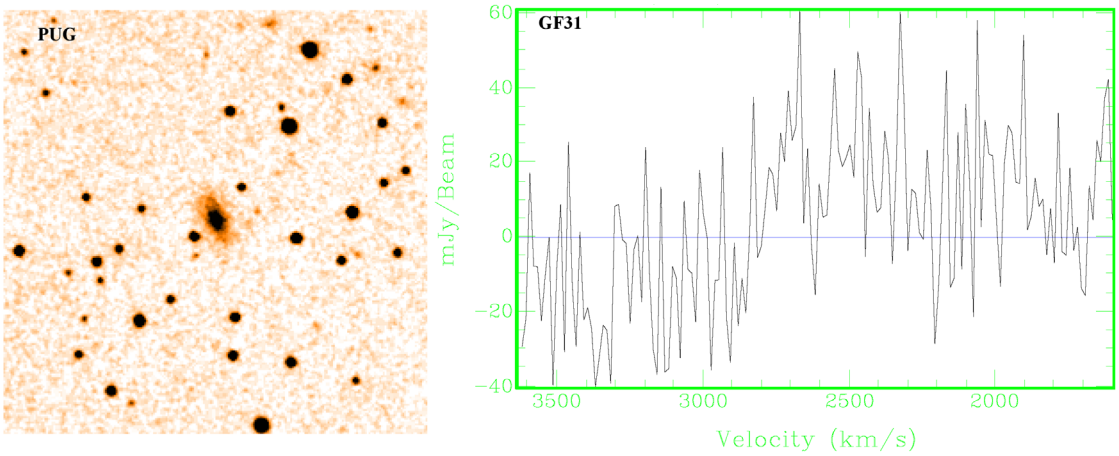


Figure 6.3. DSS image of the (previously uncatalogued) probable optical counterpart to GF31, along with the HIJASS spectrum of the GF31.

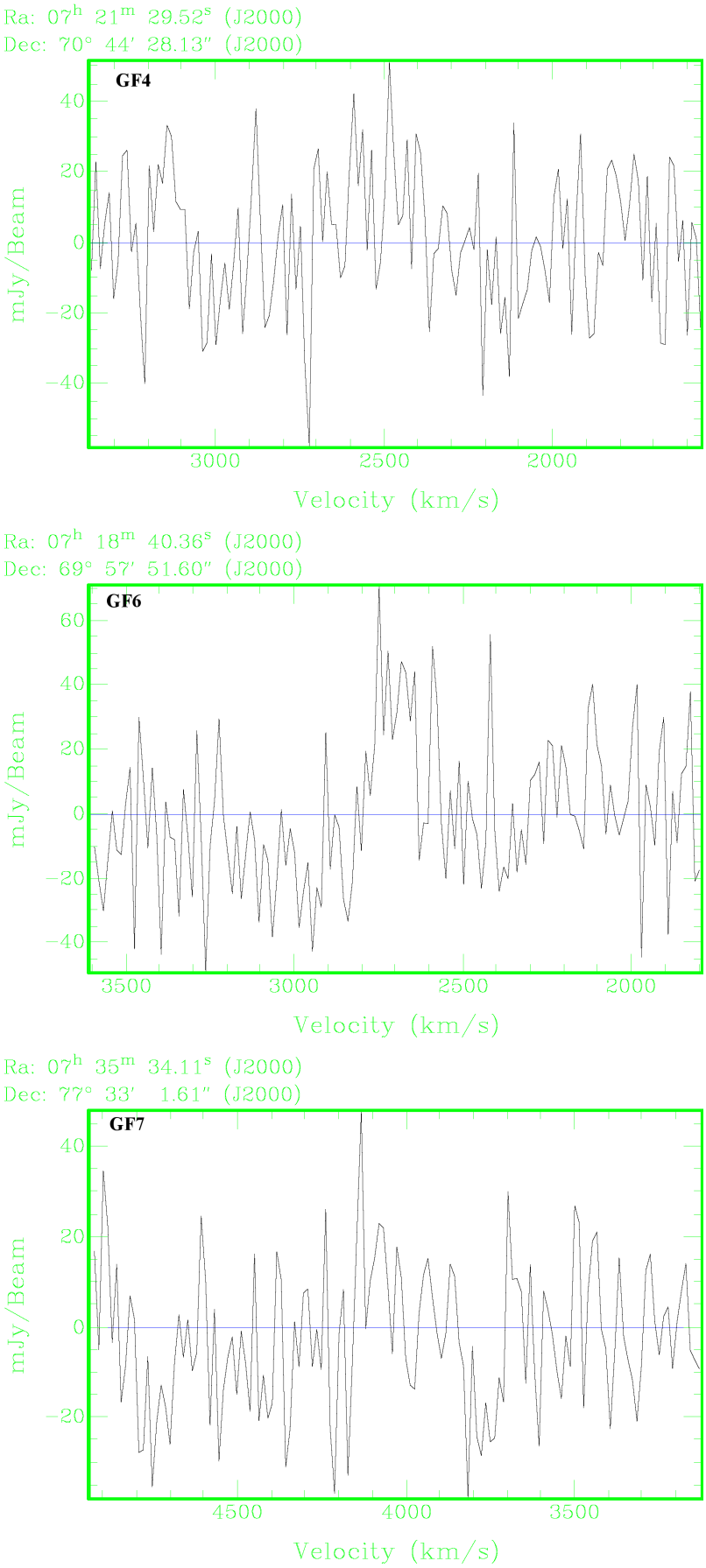
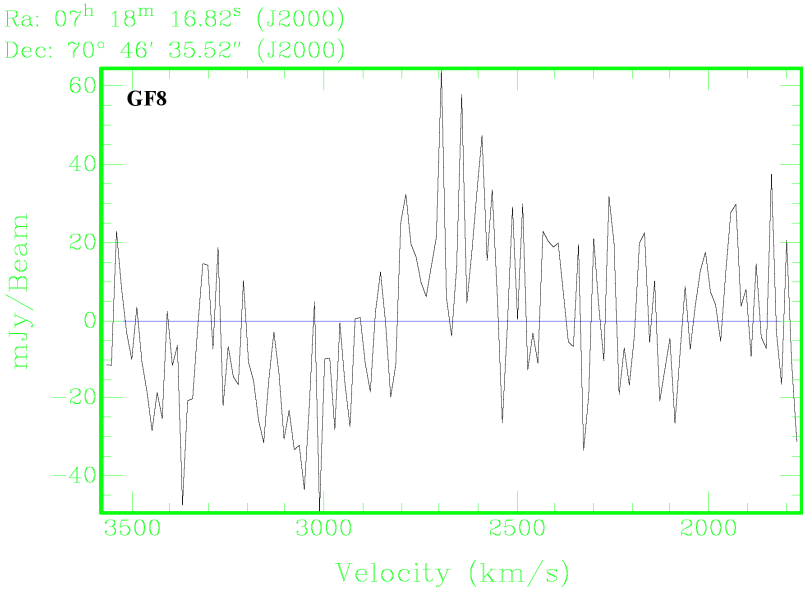
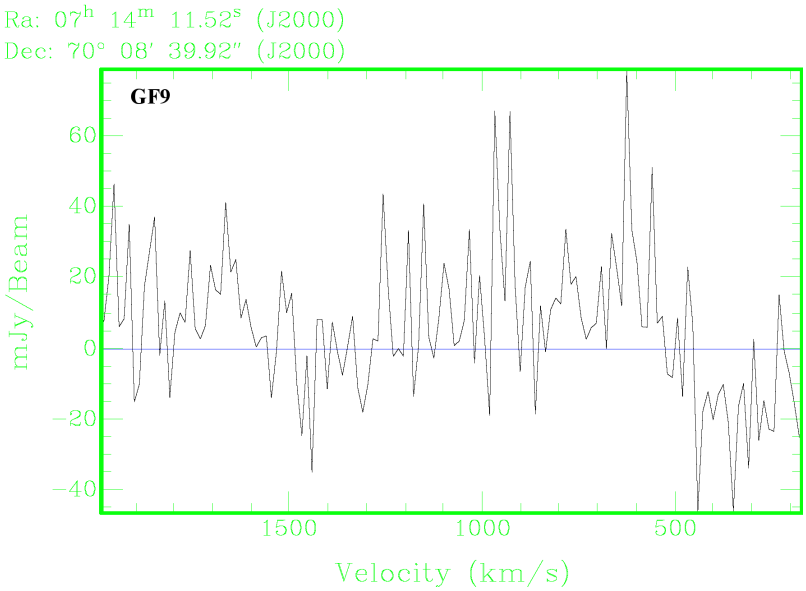


Figure 6.4. HIJASS spectra of those 17 possible sources with no optical counterpart.





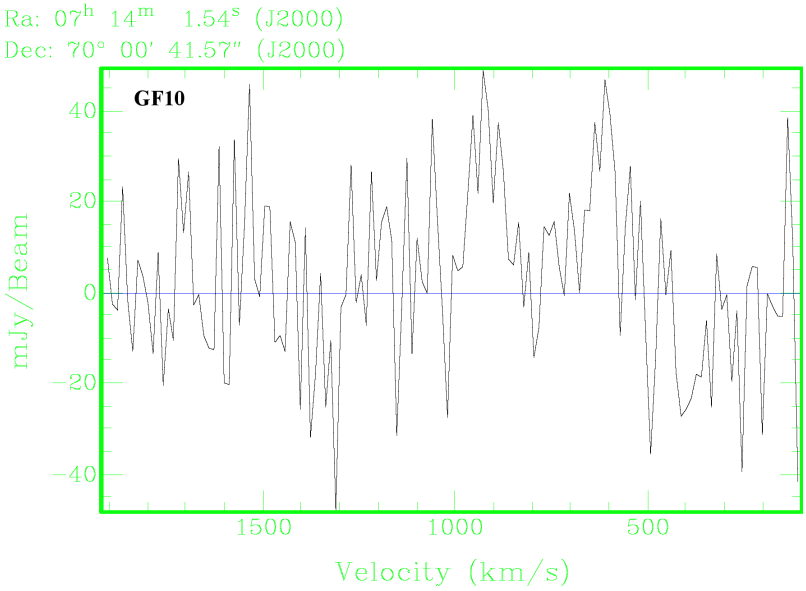


Figure 6.4. Continued.

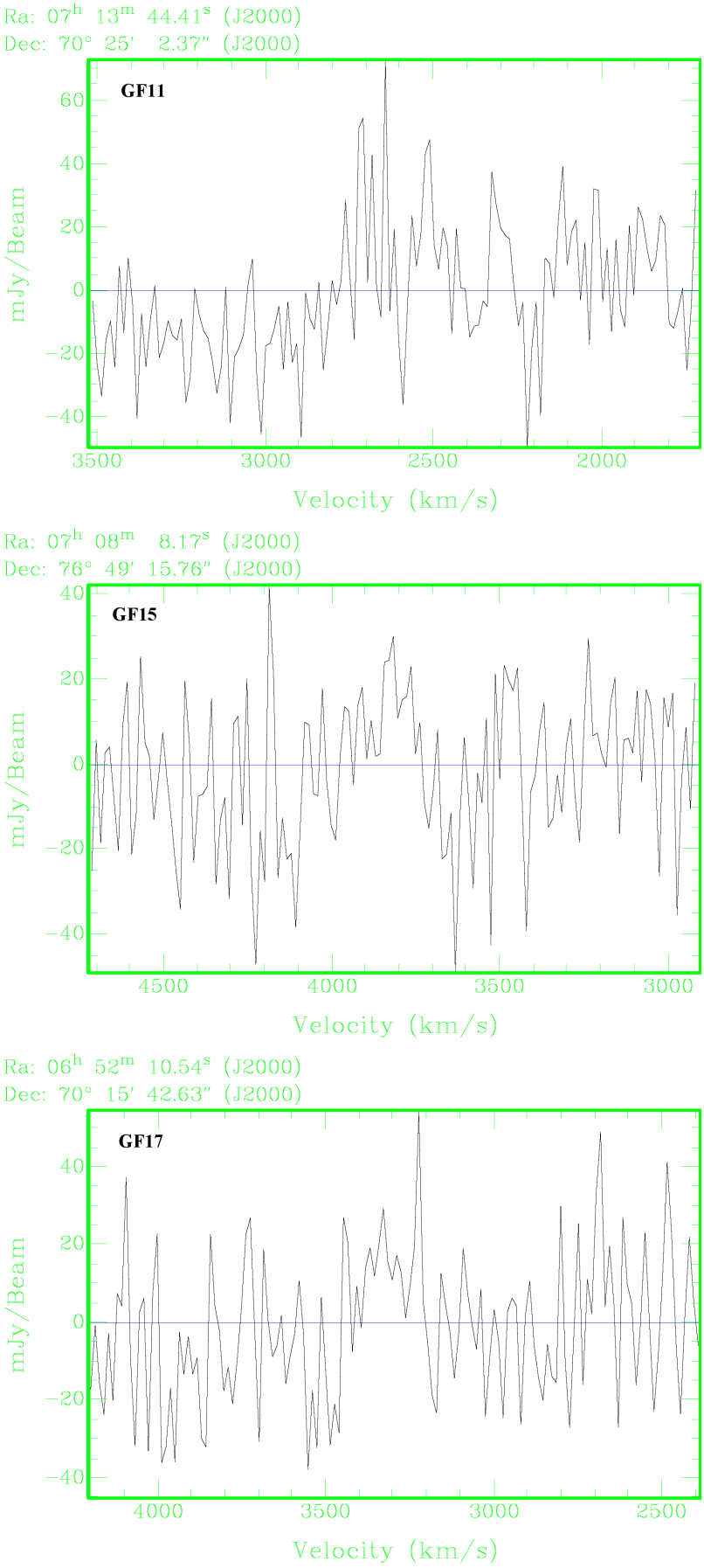


Figure 6.4. Continued.

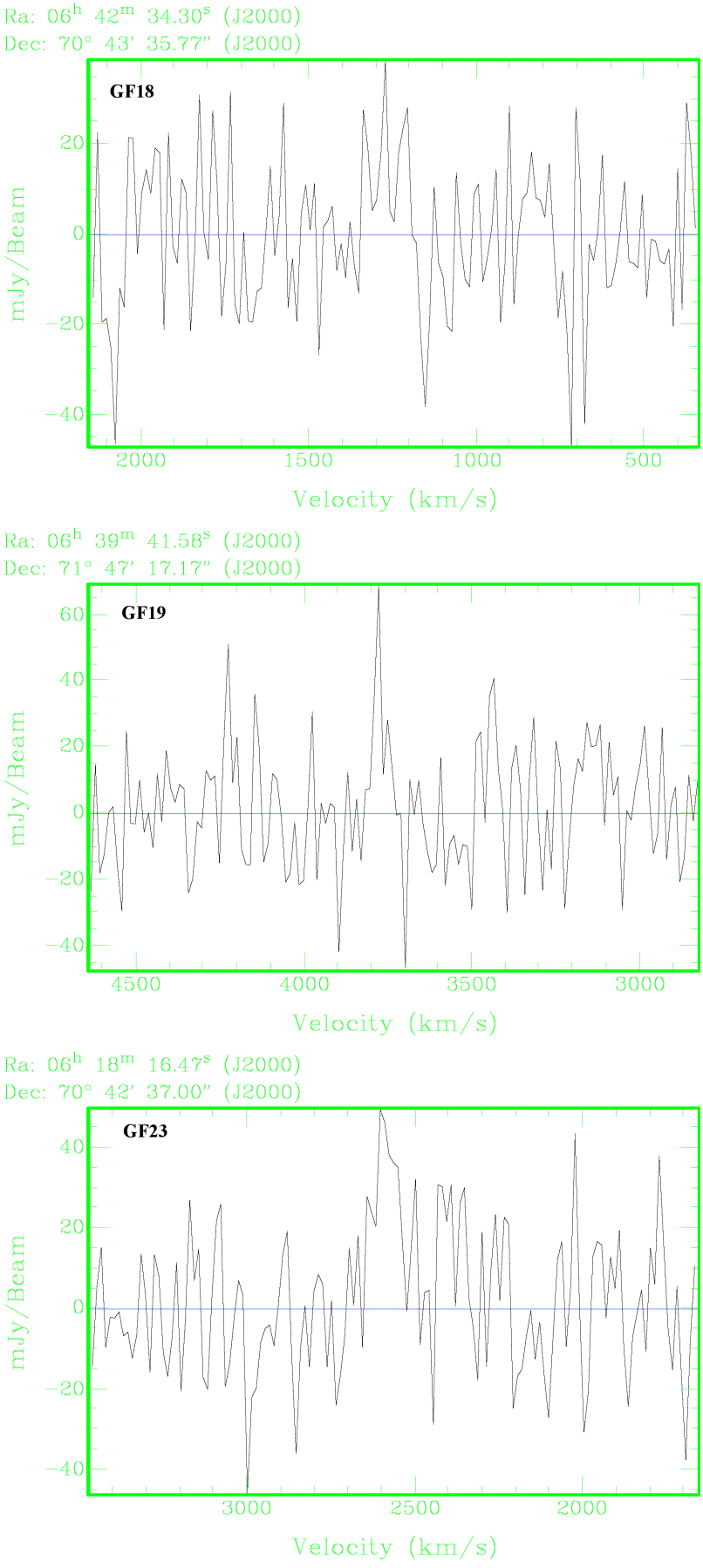


Figure 6.4. Continued.

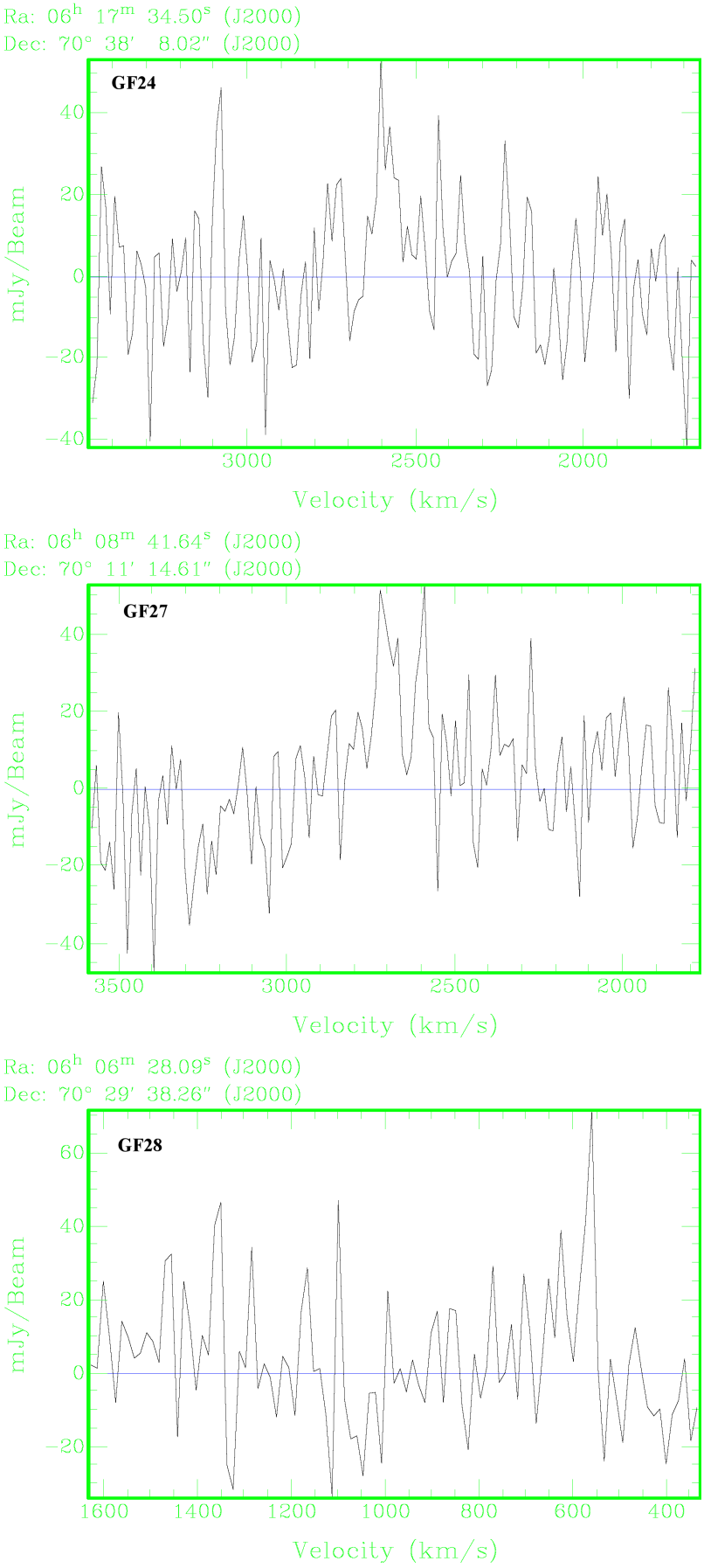


Figure 6.4. Continued.

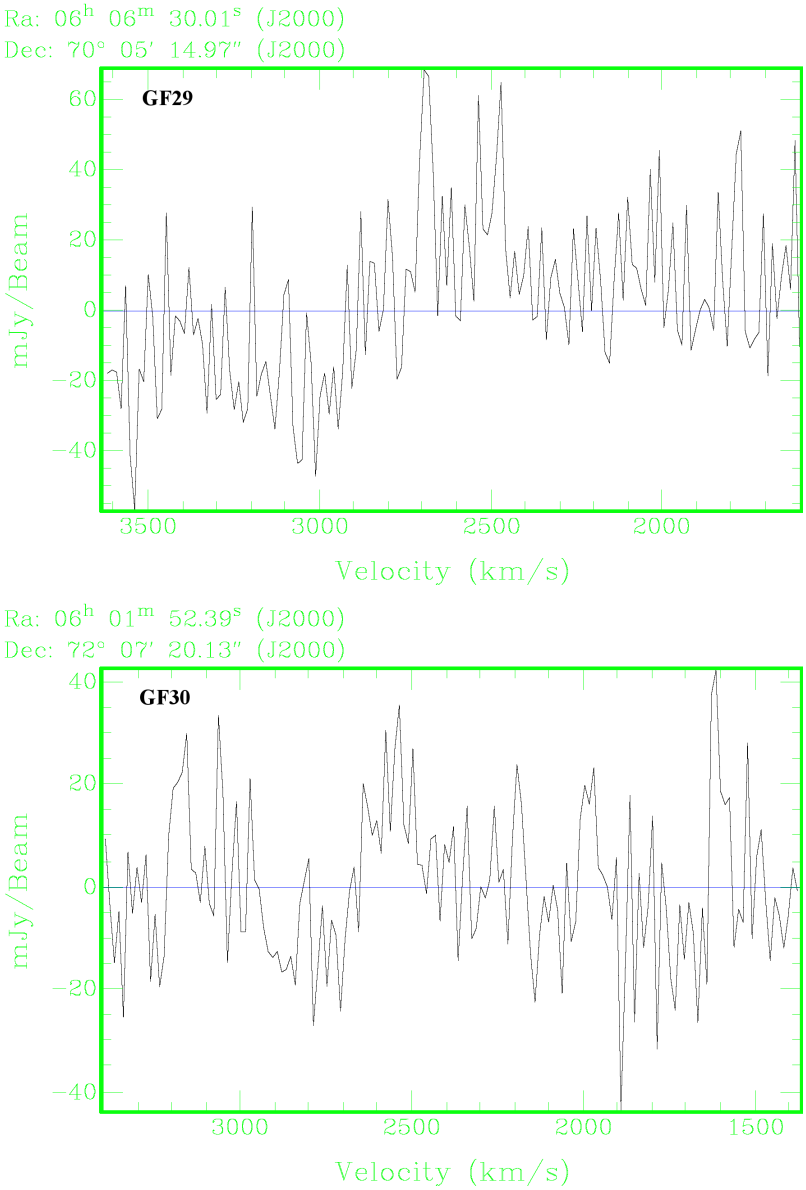


Figure 6.4. Concluded.

6.4.5 Detection limits of GammaFinder

Since GammaFinder is finding significant numbers of sources not found by visual inspection or by existing automatic finders, it must be selecting galaxies with lower peak fluxes and/or lower integrated fluxes. This issue cannot be fully studied without a detailed narrow-band follow-up program to determine the reality of GammaFinder sources to some limiting Gamma S/N. A preliminary analysis is presented in this section.

Figure 6.5 is a plot of log (Sint) against W20, showing the galaxies from the Lang et al. HIJASS catalogue which lie within cube p74cube (open triangles) and also

showing the additional 31 possible sources found by GammaFinder within p74cube4. Those 14 of the new sources which are probably real (see Section 6.4.4) are plotted as filled squares. The other 17 possible sources are plotted as fixed hexagons.

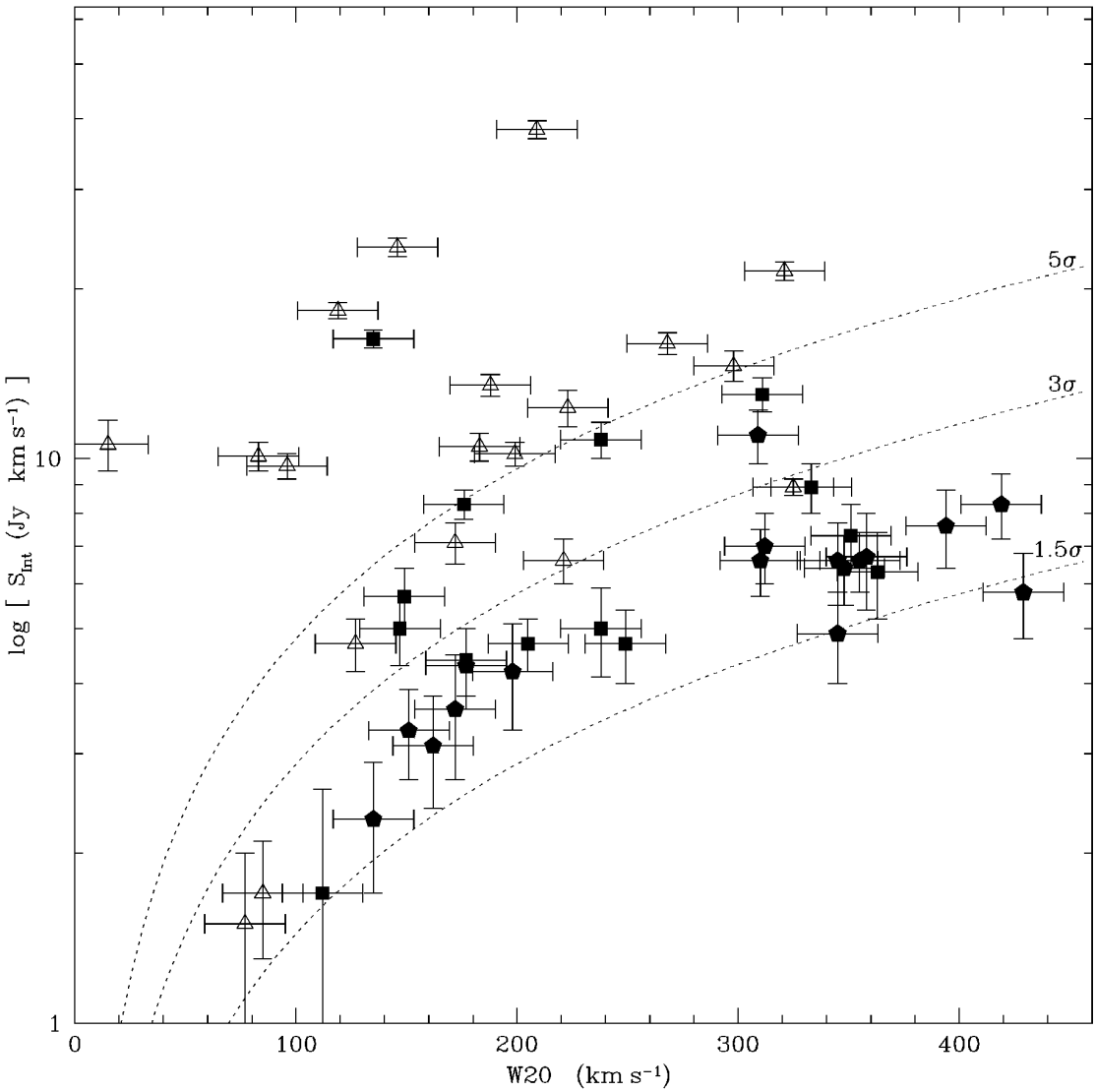


Figure 6.5. Plot of log(Sint) against W20 for galaxies from p74cube4. The open triangles represent galaxies from the Lang et al. HIJASS catalogue. The filled squares represents those 14 of the possible sources found by GammaFinder which are either IDs (10) or otherwise have plausible optical counterparts. The fixed hexagons represent the other possible sources.

As discussed in Section 2.7, the HIJASS catalogue is essentially peak flux limited with a completeness limit at about $5⑨_{noise}$ and a detection limit at about $3⑨_{noise.}$ Lang et al. showed that for galaxies in the HIJASS catalogue

$S_{int} \propto k\ W20\ S_{pk}$ (6.1)

where k is a constant which depends on the emission line's profile shape. For galaxies in the HIJASS catalogue k = 0.6!0.1. Hence, a peak flux limit translates to an integrated flux limit which is a function of W20. Plotted on Figure 6.5 are the loci of this relationship for the S_{pk} values corresponding to the completeness and detections limits of the HIJASS survey (i.e $5⑨_{noise}$ and $3⑨_{noise}$ respectively). The lower of these can be seen to constrain the galaxies in Lang et al's catalogue well, with only 2 galaxies lying below this limit. These have relatively narrow velocity-widths: the assumed value of k=0.6 is probably not strictly applicable to these.

There are several important points to note from Figure 6.5.

1. GammaFinder is finding a significant number of extra sources in the area where the completeness of the HIJASS catalogue falls below 100%. GammaFinder finds 6 possible sources between the HIJASS catalogue's detection limit of $3⑨_{noise}$ and 'completeness limit' of $5⑨_{noise}$. At least 5 of these are real (since they are IDs). This compares to the 6 sources below the completeness limit in the HIJASS catalogue.
2. GammaFinder finds 24 possible sources below the HIJASS catalogue's 'detection limit' of $3⑨_{noise}$. At least 5 of these are real (since they are IDs) and another 4 are likely to be real (the ASSs and PUG of Figures 6.2 and 6.3).
3. GammaFinder seems to be selecting galaxies on the basis of their peak flux but to a lower peak flux limit than that of the HIJASS catalogue. GammaFinder does not appear to be selecting galaxies on the basis of their integrated flux. The limiting S_{int} it can detect depends on W20 with a similar functional form to that seen for the galaxies in the HIJASS catalogue. The locus of a peak flux limit of $1.5⑨_{noise}$ is also plotted on Figure 6.5. This seems to be a good fit to a 'detection limit' for GammaFinder on this datacube.

The limited evaluation presented here suggests that GammaFinder essentially operates as a peak flux limited finder but that it can select galaxies to much fainter limits than existing finders. If the fall in the detection limit from around $3⑨_{noise}$ to $1.5⑨_{noise}$ is

confirmed by a more detailed evaluation then GammaFinder will be able to add very substantial numbers of new galaxies to those found in HIJASS and HIPASS data by existing methods.

6.5 Concluding remarks

The performance of GammaFinder both in terms of its ability to detect astronomical sources and its computational efficiency has been explored in this chapter. In Chapter 7 some conclusions are drawn from the project and some suggestions made for future work.

7. Conclusions and Future Work

7.1 Introduction

Section 7.1 of this chapter contains some ideas for future work. In Section 7.2 some conclusions from the project are drawn.

7.2 Future Work

Three areas in which future work might be undertaken are considered in this section.

7.2.1 Full evaluation of GammaFinder

The evaluation presented in Section 6.4 was, of necessity, rather limited in scope. A more detailed evaluation needs to be undertaken with the aim of investigating:

1. How the ratio of real sources to spurious sources returned by GammaFinder varies with Gamma S/N. From this can be gained a clearer idea of the best limiting Gamma S/N (`gamThres`) value to use.
2. How sensitive GammaFinder is to detecting galaxies as a function of peak and/or integrated flux. In Section 6.4.5 it was suggested that GammaFinder is operating as a peak flux limited finder, though to much better sensitivity than existing methods. Does this conclusion hold to fainter limiting Gamma S/N or with larger samples of sources?

The steps in a fuller investigation should be:

1. Run GammaFinder on several HIJASS / HIPASS datacubes.
2. Create a list of all those returned possible sources (down to some faint Gamma S/N limit) which cannot be easily dismissed as unreal.
3. Check this list against currently catalogued HIJASS and HIPASS sources.
4. Cross-check the list with NED to see if any match with catalogued galaxies in both space and velocity (i.e. are 'ID's).
5. All sources which are not already in the HIJASS / HIPASS catalogues or are not IDs need to be re-observed to test if they are real.

Once these steps have been completed, the number of real sources compared to spurious sources as a function of Gamma S/N can be easily studied. Also S_{pk}, S_{int} and W20 measurements can be made of all real sources down to faint Gamma S/N. Hence,

the sensitivity of GammaFinder as a function of S_{pk}, S_{int} and W20 and Gamma S/N can be investigated. The results of this analysis will enable the most efficient and effective use to be made of GammaFinder in applying it to existing databases of HIPASS and HIJASS data.

7.2.2 Running GammaFinder on HIPASS / HIJASS data

If the significantly improved sensitivity of GammaFinder is confirmed by the fuller evaluation proposed in Section 7.2.1, than an obvious use of GammaFinder would be to re-analyse the large dataset of completed HIPASS and HIJASS observations. This could result in a huge increase in the number of galaxies found within the survey data. For a peak flux limited survey of a homogenous distribution of galaxies we expect $N_{obj} \} S_{pk}^{-5/2}$. If the completeness limit of GammaFinder is $1.5⑨_{noise}$ below that of the existing catalogues (as the detection limit appears to be), then GammaFinder could increase the number of sources above the completeness limit by more than a factor of 2.

7.2.3 Further development of GammaFinder

The evaluation of Section 6.4.5 suggests that the core functionality of GammaFinder is highly effective. No changes should be made to this at least until after a full evaluation has been performed. Further improvements could be made to the functionality of the GUI but care needs to be taken not to obscure the main functionality of the application. Possible improvements might include the ability to re-scale the spectra and/or to zoom in on particular channel ranges. It might also be useful to be able to display 2 spectra at the same time (for example, to compare the raw spectrum to the gamma spectrum).

Following a full evaluation and prior to being released to the community, GammaFinder needs to have a detailed on-line User Guide written for it. A menu item to call such a guide has been coded into the application. A user guide written in HTML could be implemented via a `JEditorPane`.

A possible way to further develop GammaFinder would be to make it part of a more general HIPASS/HIJASS image analysis package bringing together functions which

at present require the use of several different software packages. Such a package could include the ability to display 2-dimensional 'slices' through the datacube (as can currently be undertaken by KVIEW – Gooch 1995) and to undertake basic object parameter measurements (as can currently be undertaken using the tasks of MIRIAD). Having these functions and those of GammaFinder in a single application would make the investigation of HIPASS/HIJASS data a much more integrated process.

7.3 Conclusions

GammaFinder is an effective new application for finding sources in HIPASS or HIJASS datacubes. A preliminary evaluation suggests that it may be able to lower the detection limit on such data to only $1.5⑨_{noise}$, compared to a detection limit of around $3⑨_{noise}$ for current detection methods. This could lead to a large increase in the number of sources which can be found in HIPASS and HIJASS data and significantly boost the scientific exploitation of these two major surveys.

Appendix

Source code for Voxel.java

```
/*
   Class to define a Voxel object
   Part of GammaFinder appliciation.
   @author Peter J. Boyce
   @version 1.0 2003/09/15
*/
public class Voxel
{
   //constructor
   public Voxel( float voxelValue )
   {
      voxVal = voxelValue;
   }

   //method to set the values of the instance varibles
   public void setVoxVal( float a ) { voxVal = a; }
   public void setH2Val( float b ) { h2Val = b; }
   public void setH4Val( float c ) { h4Val = c; }
   public void setH8Val( float d ) { h8Val = d; }
   public void setGVal( float e ) { gVal = e; }
   public void setG2Val( float f ) { g2Val = f; }
   public void setG4Val( float g ) { g4Val = g; }
   public void setG8Val( float h ) { g8Val = h; }

   //methods to return the values of the instance variables
   public float getVoxVal() { return voxVal; }
   public float getH2Val() { return h2Val; }
   public float getH4Val() { return h4Val; }
   public float getH8Val() { return h8Val; }
   public float getGVal() { return gVal; }
   public float getG2Val() { return g2Val; }
   public float getG4Val() { return g4Val; }
   public float getG8Val() { return g8Val; }

   //instance variables
   private float voxVal, h2Val, h4Val, h8Val, gVal, g2Val, g4Val, g8Val;
}
```

Source code for HiCube.java

```
/*
   Class to represent a 3-dimensional
   HIJASS or HIPASS datacube.
   Part of the GammaFinder application.
   @author Peter J. Boyce
   @version 1.0 2003/09/15
*/
import org.eso.fits.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;

public class HiCube
{
   //constructor
   public HiCube( float[][][] inDataArray )
   {
      xpix = inDataArray.length;
      ypix = inDataArray[0].length;
      zpix = inDataArray[0][0].length;
      voxArray = new Voxel[xpix][ypix][zpix];
      zChanQual = new boolean[zpix];

      //create a Voxel object at each (i,j,k) in VoxArray
      for( int i = 0; i < xpix; i++ )
         for( int j = 0; j < ypix; j++ )
            for( int k = 0; k < zpix; k++)
               voxArray[i][j][k] = new Voxel( inDataArray[i][j][k] );

   }


   /*
   Method used set the h2Val, h4Val, h8Val instance variables
   in every Voxel in voxArray.
   */
   public void setH2H4H8Vals()
   {
      double temp;

      for( int i = 0; i < xpix; i++)
      {
         for( int j = 0; j < ypix; j++ )
         {
            //create Hanning smoothed spectrum FWHM = 2 channels
            for(int k = 1; k < (zpix - 1) ; k = k + 2 )
            {
               temp = (0.5 * voxArray[i][j][k].getVoxVal() ) +
                       ( 0.25 * ( voxArray[i][j][k-1].getVoxVal() +
voxArray[i][j][k+1].getVoxVal() )  );
               voxArray[i][j][k].setH2Val( (float) temp );
            }

            //create Hanning smoothed spectra FWHM = 4 channels
            for( int k = 3; k < ( zpix - 3 ) ; k = k + 4)
            {
               temp = ( voxArray[i][j][k].getVoxVal() + 0.854 * (voxArray[i][j][k-
1].getVoxVal() + voxArray[i][j][k+1].getVoxVal())
                          + 0.5 * ( voxArray[i][j][k-2].getVoxVal() +
voxArray[i][j][k+2].getVoxVal() )
                         + 0.146 * ( voxArray[i][j][k-3].getVoxVal() +
voxArray[i][j][k+3].getVoxVal() )) / 4;
               voxArray[i][j][k].setH4Val( (float) temp );
            }

            //creat Hanning smoothed spectrum, FWHM = 8 channels
            for( int k = 7; k < ( zpix - 7 ) ; k = k + 8)
            {
               temp = ( voxArray[i][j][k].getVoxVal()
```

```
                        + 0.962 * (voxArray[i][j][k-1].getVoxVal() +
voxArray[i][j][k+1].getVoxVal())
                        + 0.854 * ( voxArray[i][j][k-2].getVoxVal() +
voxArray[i][j][k+2].getVoxVal() )
                        + 0.691 * ( voxArray[i][j][k-3].getVoxVal() +
voxArray[i][j][k+3].getVoxVal() )
                        + 0.500 * ( voxArray[i][j][k-3].getVoxVal() +
voxArray[i][j][k+3].getVoxVal() )
                        + 0.309 * ( voxArray[i][j][k-4].getVoxVal() +
voxArray[i][j][k+4].getVoxVal() )
                        + 0.146 * ( voxArray[i][j][k-5].getVoxVal() +
voxArray[i][j][k+5].getVoxVal() )
                        + 0.038 * ( voxArray[i][j][k-6].getVoxVal() +
voxArray[i][j][k+6].getVoxVal() )) / 8;
               voxArray[i][j][k].setH8Val( (float) temp );
            }
         }
      }
   }


   /*
   Method used set the gVal instance variables
   in every Voxel in voxArray.
   */
   public void setGVals()
   {

      float[] deltaK = new float[16];
      float[] gammaK = new float[16];
      int posq, posl;
      float deltaSum, gammaSum, sumX, sumX2, sumXY, sumY, intercept, gradient;

      for( int i = 0; i < xpix; i++)
      {
         for( int j =0; j < ypix; j++ )
         {
            for( int k = 0; k < zpix; k++)
            {

               //calculate gammaK[q] and deltaK[q] for q = 1 to 15 at this (i,j,k).
               for( int q = 1; q <= 15; q++ )
               {

                  deltaSum = 0;
                  gammaSum = 0;

                  //window is 11 channels wide
                  for( int l = k - 5; l <= k + 5; l++)
                  {
                    if( q % 2 == 1 )
                    {
                        deltaSum +=  (float)  (     ( ( q + 1 ) / 2 ) * ( ( q + 1 ) /
2 )              );
                        posq = l + ( q + 1 )/2;
                        posl = l;
                     }
                     else
                    {
                       deltaSum += (q * q / 4);
                        posq = l + q / 2;
                        posl = l;
                     }

                     if( posq < 0 )
                       posq += zpix;
                     if( posq >= zpix )
                       posq -= zpix;
                     if( posl < 0 )
                       posl += zpix ;
                    if( posl >= zpix )
                       posl -=  zpix;

                     gammaSum += (float) ( ( voxArray[i][j][posq].getVoxVal() -
voxArray[i][j][posl].getVoxVal() ) *
                                          ( voxArray[i][j][posq].getVoxVal() -
voxArray[i][j][posl].getVoxVal() ) );
```

```
                }
                deltaK[q] = deltaSum / 11;
                gammaK[q] = gammaSum / 22;
             }

             //undertake linear regression to find gamma statistic at this (i,j,z)
             sumX = 0;
             sumX2 = 0;
             sumY = 0;
             sumXY = 0;

             for ( int q = 1; q <= 15; q++)
             {
                sumX += deltaK[q];
                sumX2 += ( deltaK[q] * deltaK[q] );
                sumY += gammaK[q];
                sumXY += ( deltaK[q] * gammaK[q] );
             }

             gradient = (sumXY - (sumX * sumY / 15) ) / ( (sumX2) - ( sumX * sumX /
15 )   );
             intercept = ( sumY - ( gradient * sumX ) ) / 15;

             //copy value of gamma statistic to g2Val of relevant Voxel object
             voxArray[i][j][k].setGVal( intercept );


          }
       }
    }
 }



 /*
 Method used to set the noise instance variable
 */
 public void setNoise()
 {
    int npix = xpix * ypix * zpix;
    float[] noiseArray = new float[ npix ];
    int l = 0;
    int count = 0;

    for( int i = 0; i < xpix; i++ )
       for( int j = 0; j < ypix; j++ )
          for( int k = 0; k < zpix; k++ )
          {
             noiseArray[l] = voxArray[i][j][k].getVoxVal();
             l++;
             if( voxArray[i][j][k].getVoxVal() == 0 )
                count++;
          }


    Arrays.sort( noiseArray );

    noise = (float) Math.sqrt( noiseArray[ ( ( l - count ) / 2 ) + count ]  );

 }

 /*
 Method to set the values in the zChanQual array.
 */
 public void setZChanQual()
 {

    float medallZ;
    float[] IFPixMed = new float[zpix];
    float[] IFPixMedRT = new float[(xpix * ypix)];
    int l;

    //find median gVal in each z-channel
    for( int k = 0; k < zpix; k++ )
    {
       l = 0;
```

```
      for( int i = 0; i < xpix; i++ )
      {
         for ( int j = 0; j < ypix; j++ )
         {
            IFPixMedRT[l] = voxArray[i][j][k].getGVal();
            l++;
         }
      }
      Arrays.sort( IFPixMedRT );

      IFPixMed[k] = IFPixMedRT[l/2];
   }

   //find median gVal from IFPixMed
   float[] shadow = new float[zpix];
   for( int k = 0; k < zpix; k++)
      shadow[k] = IFPixMed[k];
   Arrays.sort( shadow );
   medallZ = shadow[ (zpix-1) / 2 ];

   //set zChanQual to true if median gVal in channel < 1.5medallZ
   for(int k = 0; k < zpix; k++ )
   {
      if( IFPixMed[k] <= (medallZ * 1.5) )
      {
          zChanQual[k] = true;
      }
   }
}

/*
Method used set the g2Val, g4Val, g8Val instance variables
in every Voxel in voxArray.
*/
public void setG2G4G8Vals()
{

   float gammaSum, deltaSum, ncount, gradient, intercept, sumX, sumX2, sumY, sumXY;
   int posq, posl;
   float[] deltah2K = new float[16];
   float[] gammah2K = new float[16];
   float[] deltah4K = new float[16];
   float[] gammah4K = new float[16];
   float[] deltah8K = new float[16];
   float[] gammah8K = new float[16];

   //perform moving window gamma test on h2Val, h4Val and h8val  spectra
   // voxels with 'bad' quality flags are exlcuded from the process.
   for( int i = 0; i < xpix; i++)
   {
      for( int j =0; j < ypix; j++ )
      {

         //conduct moving window gamma test on h2Val spectrum at (i,j)
         for( int k = 1; k < ( zpix -1 ); k = k + 2)
         {
            for( int q = 1; q <= 15; q++ )
            {
               deltaSum = 0;
               gammaSum = 0;
               ncount = 0;

               for( int l = k - 10; l <= k + 10; l = l + 2)
               {
                  posl = l;

                  if( posl < 0 )
                     posl +=  ( zpix -1 ) ;
                  if( posl >= zpix )
                     posl -= ( zpix - 1 );

                  if( zChanQual[posl] == true )
                  {
                      ncount++;
                     if( q % 2 == 1 )
                     {
                        posq = l - ( q + 1 );
```

```
                            if( posq < 0 )
                              posq +=  ( zpix - 1 );
                            if( posq >= zpix )
                              posq -= ( zpix - 1);

                           if( zChanQual[posq] == true )
                           {
                               deltaSum +=  (float) ( ( ( q + 1 ) / 2 ) * ( ( q + 1 ) /
2 ) );



                               gammaSum += (float)( ( voxArray[i][j][ posq ].getH2Val()
- voxArray[i][j][ posl ].getH2Val() ) *
                                                    ( voxArray[i][j][ posq ].getH2Val()
- voxArray[i][j][ posl ].getH2Val() ) );
                             }
                             else
                            {
                               //do nothing
                             }

                          }
                          else
                          {
                            posq = l + q;

                             if( posq < 0 )
                              posq += ( zpix - 1 );
                             if( posq >= zpix )
                              posq -= ( zpix - 1 );

                             if( zChanQual[ posq ] == true )
                            {
                               deltaSum += (q * q / 4);
                                gammaSum += (float) ( ( voxArray[i][j][posq].getH2Val()
- voxArray[i][j][posl].getH2Val() ) *
                                                     ( voxArray[i][j][posq].getH2Val()
- voxArray[i][j][posl].getH2Val() ) );
                             }
                             else
                            {
                               //do nothing
                             }
                          }
                        }
                        else
                       {
                          //do nothing
                        }
                    }
                    deltah2K[q] = deltaSum / ncount;
                    gammah2K[q] = gammaSum / ( 2 * ncount );
                }

                sumX = 0;
                sumX2 = 0;
                sumY = 0;
                sumXY = 0;

                for ( int q = 1; q <= 15; q++)
                {
                   sumX += deltah2K[q];
                   sumX2 += ( deltah2K[q] * deltah2K[q] );
                   sumY += gammah2K[q];
                   sumXY += ( deltah2K[q] * gammah2K[q] );
                }

                gradient = (sumXY - (sumX * sumY / 15) ) / ( (sumX2) - ( sumX * sumX /
15 )   );
                intercept = ( sumY - ( gradient * sumX ) ) / 15;
                voxArray[i][j][k].setG2Val( intercept );
            }


            //conduct moving window gamma test on h4Val spectrum at (i,j)
```

```
            for( int k = 3; k < ( zpix - 3 ); k = k + 4)
            {
               for( int q = 1; q <= 15; q++ )
               {
                  deltaSum = 0;
                  gammaSum = 0;
                  ncount = 0;

                  for( int l = k - 20; l <= k + 20; l = l + 4)
                  {
                     posl = l;

                     if( posl < 0 )
                        posl +=  ( zpix - 3 ) ;
                     if( posl >= zpix )
                        posl -= ( zpix - 3 );

                     if( zChanQual[posl] == true )
                     {
                         ncount++;
                         if( q % 2 == 1 )
                        {
                           posq = l - ( ( 2 * q ) + 2 );

                            if( posq < 0 )
                              posq += ( zpix - 3 );
                            if( posq >= zpix )
                              posq -= ( zpix - 3 );

                           if( zChanQual[posq] == true )
                           {
                               deltaSum += (float) ( ( q + 1 ) * ( q + 1 ) / 4 );


                               gammaSum += (float) ( ( voxArray[i][j][posq].getH4Val()
- voxArray[i][j][posl].getH4Val() ) *
                                                     ( voxArray[i][j][posq].getH4Val()
- voxArray[i][j][posl].getH4Val() ) );
                            }
                            else
                           {
                              //do nothing
                            }
                         }
                         else
                        {
                          posq = l + ( 2 * q );

                            if( posq < 0 )
                              posq += ( zpix - 3 );
                            if( posq >= zpix )
                              posq -= ( zpix - 3 );

                            if( zChanQual[ posq ] == true )
                           {
                               deltaSum += (float) ( q * q / 4 );
                                gammaSum += (float) ( ( voxArray[i][j][posq].getH4Val()
- voxArray[i][j][posl].getH4Val() ) *
                                                     ( voxArray[i][j][posq].getH4Val()
- voxArray[i][j][posl].getH4Val() ) );
                            }
                            else
                           {
                              //do nothing
                            }
                         }
                      }
                      else
                     {
                        //do nothing
                      }
                  }
                  deltah4K[q] = deltaSum / ncount;
                  gammah4K[q] = gammaSum / ( 2 * ncount );
               }

               sumX = 0;
```

```
                sumX2 = 0;
                sumY = 0;
                sumXY = 0;

                for ( int q = 1; q <= 15; q++)
                {
                   sumX += deltah4K[q];
                   sumX2 += ( deltah4K[q] * deltah4K[q] );
                   sumY += gammah4K[q];
                   sumXY += ( deltah4K[q] * gammah4K[q] );
                }

                gradient = (sumXY - (sumX * sumY / 15) ) / ( (sumX2) - ( sumX * sumX /
15 )   );
                intercept = ( sumY - ( gradient * sumX ) ) / 15;
                voxArray[i][j][k].setG4Val( intercept );
             }


             //conduct moving window gamma test on h2Val spectrum at (i,j)
             for( int k = 7; k < ( zpix - 7 ); k = k + 8)
             {

                for( int q = 1; q <= 15; q++ )
                {
                   deltaSum = 0;
                   gammaSum = 0;
                   ncount = 0;

                   for( int l = k - 40; l <= k + 40; l = l + 8)
                   {
                     posl = l;

                      if( posl < 0 )
                        posl += ( zpix - 7 ) ;
                     if( posl >= zpix )
                       posl -= ( zpix - 7 );

                     if( zChanQual[posl] == true )
                     {
                         ncount++;
                         if( q % 2 == 1 )
                        {
                           posq = l - ( ( 4 * q ) + 4 );

                            if( posq < 0 )
                              posq += ( zpix - 7 );
                            if( posq >= zpix )
                              posq -= ( zpix - 7 );


                           if( zChanQual[posq] == true )
                           {
                               deltaSum +=  (float) (  ( q + 1 ) * ( q + 1 ) / 4  );


                                gammaSum += (float) ( ( voxArray[i][j][posq].getH8Val()
- voxArray[i][j][posl].getH8Val() ) *
                                                      ( voxArray[i][j][posq].getH8Val()
- voxArray[i][j][posl].getH8Val() ) );
                            }
                           else
                           {
                             //do nothing
                            }
                         }
                         else
                         {
                           posq = l + ( 4 * q );

                            if( posq < 0 )
                              posq += ( zpix - 7 );
                            if( posq >= zpix )
                              posq -= ( zpix - 7 );

                            if( zChanQual[ posq ] == true )
                           {
```

```
                              deltaSum += ( q * q / 4 );
                               gammaSum += (float) ( ( voxArray[i][j][ posq
].getH8Val()  - voxArray[i][j][ posl ].getH8Val() ) *
                                                    ( voxArray[i][j][ posq
].getH8Val()  - voxArray[i][j][ posl ].getH8Val() ) );
                            }
                            else
                           {
                              //do nothing
                            }
                         }
                      }
                      else
                      {
                         //do nothing
                      }
                   }
                   deltah8K[q] = deltaSum / ncount;
                   gammah8K[q] = gammaSum / ( 2 * ncount );
                }

                sumX = 0;
                sumX2 = 0;
                sumY = 0;
                sumXY = 0;

                for ( int q = 1; q <= 15; q++)
                {
                   sumX += deltah8K[q];
                   sumX2 += ( deltah8K[q] * deltah8K[q] );
                   sumY += gammah8K[q];
                   sumXY += ( deltah8K[q] * gammah8K[q] );
                }

                gradient = (sumXY - (sumX * sumY / 15) ) / ( (sumX2) - ( sumX * sumX /
15 )   );
                intercept = ( sumY - ( gradient * sumX ) ) / 15;
                voxArray[i][j][k].setG8Val( intercept );
             }
          }
       }
    }


    //method to return voxArray
    public Voxel[][][] getVoxArray()
    {
       return voxArray;
    }


    //method to return a particular voxel
    public Voxel getVoxel(int i, int j, int k)
    {
       return voxArray[i][j][k];
    }

    //method to return the noise instance variable
    public float getNoise()
    {
       return noise;
    }

    //method to return the zChanQual instance variable
    public boolean[] getZChanQual()
    {
       return zChanQual;
    }

    //instance variables
    private int xpix, ypix, zpix;
    private Voxel[][][] voxArray;
    private boolean[] zChanQual;
    private float noise;

}
```

Source code for GammaFinder.java

```
/*
   Class to create an application which finds possible sources
   in HIJASS and DIPASS data.
   @author Peter J. Boyce
   @version 1.0 2003/09/15
*/
import org.eso.fits.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;

public class GammaFinder
{
   public static void main( String[] args )
   {
      GuiFrame guiFrame = new GuiFrame();
      guiFrame.setTitle( "GammaFinder" );
      guiFrame.pack();
      guiFrame.show();
   }
}

class GuiFrame extends JFrame implements ActionListener
{
   //GuiFrame constructor
   public GuiFrame()
   {

      //set cross platform 'look and feel'
      try
      {
         UIManager.setLookAndFeel( UIManager.getCrossPlatformLookAndFeelClassName() );
      }
      catch (Exception e )
      {
         JOptionPane dialog = new JOptionPane();
         dialog.showMessageDialog( this, "There was a problem setting Look and Feel.
Default will be used",
                                   "Information", JOptionPane.PLAIN_MESSAGE);
      }

      setDefaultCloseOperation( EXIT_ON_CLOSE );

      Container contentPane = getContentPane();
      contentPane.setLayout( new BorderLayout( 1, 5 ) );

      //create a menu
      mbar = new JMenuBar();


      //create a top level menu item
      m1 = new JMenu( "File" );
      m1.setMnemonic( KeyEvent.VK_F );

      //create a list of menu items and add to m1
      j1 = new JMenuItem( "Open File" );
      j1.addActionListener( this );
      j1.setMnemonic( KeyEvent.VK_O );
      j2 = new JMenuItem( "Save Results" );
      j2.addActionListener( this );
      j2.setMnemonic( KeyEvent.VK_S );
      j3 = new JMenuItem( "Run Finder" );
      j3.addActionListener( this );
      j3.setMnemonic( KeyEvent.VK_I );
      j4 = new JMenuItem( "Exit" );
      j4.addActionListener( this );
      j4.setMnemonic( KeyEvent.VK_X );
```

```
m1.add(j1);
m1.add(j2);
m1.addSeparator();
m1.add(j3);
m1.addSeparator();
m1.add(j4);

//create a top level menu item
m2 = new JMenu( "Edit" );
m2.setMnemonic( KeyEvent.VK_E );

//create a list of menu items and add to m2
j5 = new JMenuItem( "Cut" );
j5.addActionListener( this );
j5.setMnemonic( KeyEvent.VK_C );
j6 = new JMenuItem( "Copy" );
j6.addActionListener( this );
j6.setMnemonic( KeyEvent.VK_Y );
j7 = new JMenuItem( "Paste" );
j7.addActionListener( this );
j7.setMnemonic( KeyEvent.VK_P );
m2.add( j5 );
m2.add( j6 );
m2.add( j7 );


//create a top level menu item
m3 = new JMenu( "Help" );
m3.setMnemonic( KeyEvent.VK_H );

//create a list of menu items and add to m3
j8 = new JMenuItem( "Help Topics" );
j8.addActionListener( this );
j8.setMnemonic( KeyEvent.VK_T );
j9 = new JMenuItem( "About GalFinder" );
j9.addActionListener( this );
j9.setMnemonic( KeyEvent.VK_A );
m3.add( j8 );
m3.add( j9 );

//add top level menu items to menu bar
mbar.add( m1 );
mbar.add( m2 );
mbar.add( m3 );

//show menu bar
setJMenuBar( mbar );

//create ToolBar
panel1 = new JPanel();
panel1.setLayout( new FlowLayout( FlowLayout.LEFT, 2, 3 ) );
panel1.setBorder( new LineBorder (Color.black)  );

//create buttons for ToolBar
image1 = new ImageIcon( "Open24.gif" );
image2 = new ImageIcon( "Save24.gif" );
image3 = new ImageIcon( "Find24.gif" );
image4 = new ImageIcon( "Cut24.gif" );
image5 = new ImageIcon( "Copy24.gif" );
image6 = new ImageIcon( "Paste24.gif" );

button1 = new JButton( "  Open  ", image1 );
button1.setVerticalTextPosition( AbstractButton.BOTTOM );
button1.setHorizontalTextPosition( AbstractButton.CENTER );
button1.setMnemonic( KeyEvent.VK_O );
button1.addActionListener( this );
button1.setBorder( BorderFactory.createRaisedBevelBorder() );
button1.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
button1.setToolTipText("Open New File");

button2 = new JButton( "  Save  ", image2 );
button2.setVerticalTextPosition( AbstractButton.BOTTOM );
button2.setHorizontalTextPosition( AbstractButton.CENTER );
button2.setMnemonic( KeyEvent.VK_S );
button2.addActionListener( this );
button2.setBorder( BorderFactory.createRaisedBevelBorder() );
button2.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
```

```
button2.setToolTipText("Save Results to a Text File");

button3 = new JButton( "  Find  ", image3 );
button3.setVerticalTextPosition( AbstractButton.BOTTOM );
button3.setHorizontalTextPosition( AbstractButton.CENTER );
button3.setMnemonic( KeyEvent.VK_I );
button3.addActionListener( this );
button3.setBorder( BorderFactory.createRaisedBevelBorder() );
button3.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
button3.setToolTipText("Search for galaxies in the cube");

button4 = new JButton( "  Cut   ", image4 );
button4.setVerticalTextPosition( AbstractButton.BOTTOM );
button4.setHorizontalTextPosition( AbstractButton.CENTER );
button4.setMnemonic( KeyEvent.VK_C );
button4.addActionListener( this );
button4.setBorder( BorderFactory.createRaisedBevelBorder() );
button4.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
button4.setToolTipText("Cut selected text from Results Panel to clipboard");

button5 = new JButton( "  Copy  ", image5 );
button5.setVerticalTextPosition( AbstractButton.BOTTOM );
button5.setHorizontalTextPosition( AbstractButton.CENTER );
button5.setMnemonic( KeyEvent.VK_Y );
button5.addActionListener( this );
button5.setBorder( BorderFactory.createRaisedBevelBorder() );
button5.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
button5.setToolTipText("Copy selected text from Results Panel to clipboard");

button6 = new JButton( " Paste  ", image6 );
button6.setVerticalTextPosition( AbstractButton.BOTTOM );
button6.setHorizontalTextPosition( AbstractButton.CENTER );
button6.setMnemonic( KeyEvent.VK_P );
button6.addActionListener( this );
button6.setBorder( BorderFactory.createRaisedBevelBorder() );
button6.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
button6.setToolTipText("Paste text into Results Panel from clipboard");


jspace = new JLabel( "           " );

gLab = new JLabel( " Gamma S/N Threshold " );
gLab.setHorizontalAlignment(JLabel.CENTER);
gLab.setForeground( Color.black );
gLab.setFont( new Font( "Courier", Font.BOLD, 12 ) );
gLab.setLabelFor( gammaThres );

//create JCombo box for selecting Gamma S/N threshold
gammaThres = new JComboBox();
gammaThres.addItem("2.0");
gammaThres.addItem("2.5");
gammaThres.addItem("3.0");
gammaThres.addItem("3.5");
gammaThres.addItem("4.0");
gammaThres.addItem("4.5");
gammaThres.addItem("5.0");
gammaThres.addItem("5.5");
gammaThres.addItem("6.0");
gammaThres.addItem("6.5");
gammaThres.addItem("7.0");
gammaThres.addItem("7.5");
gammaThres.addItem("8.0");
gammaThres.addItem("8.5");
gammaThres.addItem("9.0");
gammaThres.addItem("9.5");
gammaThres.addItem("10.0");
gammaThres.setSelectedItem("5.0");
gammaThres.setEditable( true );
gammaThres.addActionListener( this );

//add components to ToolBar
panel1.add( button1 );
panel1.add( button2 );
panel1.add( button3 );
panel1.add( button4 );
panel1.add( button5 );
panel1.add( button6 );
```

```
        panel1.add( jspace );
        panel1.add( gLab );
        panel1.add( gammaThres );

        //add ToolBar to contentPane
        contentPane.add( "North", panel1 );


        //create a JTabbedPanel and add to contentPane
        tabbedPanel = new JTabbedPane();
        contentPane.add( "Center", tabbedPanel );


        //create panel with a text area and scroll bar, add to page 1 of tabbedPanel
        panel2 = new JPanel();
        textPanel = new JTextArea( 24, 100 );
        textPanel.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
        textPanel.setLineWrap( true );
        textPanel.setWrapStyleWord( true );
        scroll = new JScrollPane( textPanel,
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER );
        panel2.add( scroll );
        tabbedPanel.addTab( "Results Panel", panel2 );

        //create panel for displaying spectra
        panel3 = new JPanel();
        panel3.setLayout( new BorderLayout() );

        //create panel for selecting spectrum to display
        panel3n = new JPanel() ;
        panel3n.setLayout( new FlowLayout( FlowLayout.LEFT, 2, 4 ) );

        selSpec = new JLabel( "  Select Type of Spectrum " );
        selSpec.setHorizontalAlignment(JLabel.CENTER);
        selSpec.setForeground( Color.black );
        selSpec.setFont( new Font( "Courier", Font.BOLD, 12 ) );
        selSpec.setLabelFor( typeOfSpec1 );
        panel3n.add( selSpec );

        //create JCombo box for selecting type of spectrum to display
        typeOfSpec1 = new JComboBox();
        typeOfSpec1.addItem("Raw Spectrum");
        typeOfSpec1.addItem("Gamma Spectrum");
        typeOfSpec1.addItem("Hann2");
        typeOfSpec1.addItem("Hann4");
        typeOfSpec1.addItem("Hann8");
        typeOfSpec1.addItem("Gam2");
        typeOfSpec1.addItem("Gam4");
        typeOfSpec1.addItem("Gam8");
        typeOfSpec1.setSelectedItem("Raw Spectrum");
        typeOfSpec1.addActionListener( this );
        panel3n.add( typeOfSpec1 );

        space1 = new JLabel( "       " );
        panel3n.add( space1 );

        x1 = new JLabel( "X Coord" );
        x1.setHorizontalAlignment(JLabel.CENTER);
        x1.setForeground( Color.black );
        x1.setFont( new Font( "Courier", Font.BOLD, 12 ) );
        x1.setLabelFor( x1Text );
        panel3n.add( x1 );

        //add JTextfields to read coords of spectrum to be displayed
        x1Text = new JTextField( 4 );
        x1Text.setText("1");
        panel3n.add( x1Text );

        jspace2 = new JLabel( "    " );
        panel3n.add( jspace2 );

        y1 = new JLabel( "Y Coord" );
        y1.setHorizontalAlignment(JLabel.CENTER);
        y1.setForeground( Color.black );
        y1.setFont( new Font( "Courier", Font.BOLD, 12 ) );
        y1.setLabelFor( y1Text );
```

```
      panel3n.add( y1 );

      y1Text = new JTextField( 4 );
      y1Text.setText("1");
      panel3n.add( y1Text );

      space3 = new JLabel( "        " );
      panel3n.add( space3 );

      //ad button to display requested spectrum
      button7 = new JButton( "  Display Spectrum  " );
      button7.setMnemonic( KeyEvent.VK_D );
      button7.addActionListener( this );
      button7.setBorder( BorderFactory.createRaisedBevelBorder() );
      button7.setFont( new Font( "Courier", Font.PLAIN, 12 ) );
      button7.setToolTipText("Displays the selected spectrum");
      panel3n.add( button7);

      panel3.add( "North", panel3n );

      //create dispPanel for displaying spectra on, add to JTabbedPane
      dispPanel = new DisplayPanel( );
      panel3.add( "Center", dispPanel );
      tabbedPanel.addTab( "Display Panel", panel3 );


      //set the default Gamma S/N threshold
      gamThres = (float) 5.0;


   }

    //inner class to speicfy action to be perfomed when interactive element
    //of the GUI is selected
   public void actionPerformed( ActionEvent e )
   {

      //action if 'Open' or 'Open File' is selected
      if( e.getSource() == j1 || e.getSource() == button1 )
      {

         //open and read FITS datacube, create HICube object, hiCube
        fileFlag = true;

        readFile();

         if( fileFlag == true )
         {
            JOptionPane dialog = new JOptionPane();
            dialog.showMessageDialog( this, "File read in successfully, will now
create a HiCube", "Information", JOptionPane.PLAIN_MESSAGE );
         }
         else
             return;

         //set the h2Val, h4Val and h8Val instance variables of the Voxels in hiCube
        hiCube.setH2H4H8Vals();

         //set the gVal instance variables of the Voxels in hiCube
         hiCube.setGVals();

         //set the noise instance variable of hiCube
         hiCube.setNoise();

         //set the ZChanQual instance variable of hiCube
         hiCube.setZChanQual();

         //set the g2Val, g4Val and g8Val instance variables of the Voxels in hiCube
         hiCube.setG2G4G8Vals();


         //inform user that hiCube has been successfully created
         JOptionPane dialog1 = new JOptionPane();
         dialog1.showMessageDialog( this, "HiCube succesffully created. Image Stats
are on Results Panel",
                                    "Information", JOptionPane.PLAIN_MESSAGE );
```

```
         //display of textPanel image dimension, noise and bad z-channels
         textPanel.append( "The image has been succesffully read in and a HiCube
created");
         textPanel.append( "\n" );
         textPanel.append( "Image Size = ( " + xpix + ", " + ypix + ", " + zpix + ")
\n"  );
         textPanel.append( "Average noise in image is " + hiCube.getNoise() + "Jy \n"
);
         textPanel.append( "The following z channels were found to suffer from
significant RFI and were cleaned \n");

         boolean[] qualArray = hiCube.getZChanQual();

         for( int j = 0; j < zpix; j++)
           if( qualArray[j] == false )
               textPanel.append( j + "\n" );

      }

      //action if 'Save' or 'Save Results' is selected
      else if( e.getSource() == j2 || e.getSource() == button2 )
      {
          //save results on textPanel to a text file
         saveResults();
      }

      //action if 'Find' of 'Run Finder' is selected
      else if( e.getSource() == j3 || e.getSource() == button3 )
      {
          //find possible galaxies in the datacube
         findSources();
      }

      //action if 'Exit' is selected
      else if ( e.getSource() == j4 )
      {
          System.exit( 0 );
      }

      //action if 'Cut' is selected
      else if ( e.getSource() == j5 || e.getSource() == button4)
      {
          textPanel.cut();
      }

      //action if 'copy' is selected
      else if ( e.getSource() == j6 || e.getSource() == button5 )
      {
          textPanel.copy();
      }

      //action if 'Paste' is selected
      else if ( e.getSource() == j7 || e.getSource() == button6 )
      {
          textPanel.paste();
      }

      //action if 'Help' is selected
      else if ( e.getSource() == j8 )
      {
          //call up Help dialog
      }

      //action if 'About' is selected
      else if ( e.getSource() == j9 )
      {
          JOptionPane dialog = new JOptionPane();
          dialog.showMessageDialog( this, "GalFinder was developed by Peter J. Boyce",
"Information", JOptionPane.PLAIN_MESSAGE );
      }

      //action if value of gamThres is altered
      else if ( e.getSource() == gammaThres )
      {
         gamThres = setGamThres();
```

```
      }

      //action if "Display Spectrum" is selected
      else if ( e.getSource() == button7 )
      {
         displayFlag = true;
         updateDispPars();
         if( displayFlag == true )
         {
            dispPanel.updateSPanel( hiCube.getVoxArray(), xcoord, ycoord, specType);
            dispPanel.setVisible( true );
         }
      }

   }


   /*
   Method used to open a FITS datacube, read the data and create a
   new Hicube object, hiCube
   */
   private void readFile()
   {


      //enable user to choose input file with JFileChooser
      float[][][] inFileData = null;
      JFileChooser fc = null;

      if( fc == null );
      {
         fc = new JFileChooser();
         fc.addChoosableFileFilter(new FitsFilter());
         fc.setAcceptAllFileFilterUsed(false);
      }

      int returnVal = fc.showOpenDialog( GuiFrame.this );

      if( returnVal == JFileChooser.APPROVE_OPTION)
      {
         FitsFile infile;

         try
         {
            infile = new FitsFile( fc.getSelectedFile() );
         }
         catch (FitsException ee)
         {
            JOptionPane dialog = new JOptionPane();
            dialog.showMessageDialog( this, "Error opening fits file", "Error",
JOptionPane.ERROR_MESSAGE );
            fileFlag = false;
           return;
         }
         catch (IOException ioe)
         {
            JOptionPane dialog = new JOptionPane();
            dialog.showMessageDialog( this, "Error opening fits file", "Error",
JOptionPane.ERROR_MESSAGE );
            fileFlag = false;
           return;
         }


         //read data from file selected
         int noHDU = infile.getNoHDUnits();

         for (int i=0; i<noHDU; i++)
         {
            FitsHDUnit hdu = infile.getHDUnit(i);
            FitsHeader hdr = hdu.getHeader();
            int noKw = hdr.getNoKeywords();
            int type = hdr.getType();
            int size = (int) hdr.getDataSize();

           if (type == Fits.IMAGE)
            {
```

```
                FitsMatrix dm = (FitsMatrix) hdu.getData();
                int naxis[] = dm.getNaxis();
                 int nv, off, npix;
                 int nval = dm.getNoValues();

                if ( nval > 0 )
                 {
                   int ncol = naxis[0];
                    xpix = naxis[0];
                    ypix = naxis[1];
                    zpix = naxis[2];
                    inFileData = new float[xpix][ypix][zpix];
                    int nrow = nval/ncol;
                    float data[] = new float[ncol];

                   float val;

                    int ic = 0;
                    int jc = 0;
                    int kc = 0;
                    off = nv = 0 ;
                   for (int nr=0; nr<nrow; nr++)
                    {
                      try
                       {
                         dm.getFloatValues(off, ncol, data);
                         for (int n = 0; n<ncol; n++)
                          {
                            val = data[n];
                            inFileData[ic][jc][kc] = val ;

                             if( ic == ( xpix - 1 )  && jc == ( ypix - 1 ) )
                            {
                               ic = 0;
                               jc = 0;
                                 kc++;
                             }
                             else if ( ic == ( xpix - 1 )  && jc != ( ypix - 1 ) )
                            {
                               ic = 0;
                                jc++;
                             }
                             else
                               ic++;

                          }
                       }
                       catch (FitsException ee)
                       {
                          JOptionPane dialog = new JOptionPane();
                          dialog.showMessageDialog( this, "Error reading fits file",
"Error", JOptionPane.ERROR_MESSAGE );
                          return;
                       }

                      off += ncol;
                   }

                }
              }
               else if (type==Fits.BTABLE || type==Fits.ATABLE)
              {
                  JOptionPane dialog = new JOptionPane();
                  dialog.showMessageDialog( this, "Data is not in fits format", "Error",
JOptionPane.ERROR_MESSAGE );
                  return;
               }
           }
         }
         else
         {
            fileFlag = false;
            return;
         }

         //create a new HiCube object with datra read from file
         hiCube = new HiCube( inFileData );
```

```
   }



   /*
   Method used to copy cointents of textPanel to file selected by user
   */
   private void saveResults()
   {

      //enable user to choose output file with JFileChooser
      JFileChooser fc = null;

      if( fc == null );
      {
         fc = new JFileChooser();

         //add custom file filter and disable the deafult file filter
         fc.addChoosableFileFilter(new TxtFilter());
         fc.setAcceptAllFileFilterUsed(false);
      }

      int returnVal = fc.showSaveDialog( GuiFrame.this );


      //copy contents of textPanel to selected text file
      if( returnVal == JFileChooser.APPROVE_OPTION)
      {

         String fileText = textPanel.getText();
         int fTLen = fileText.length();

         FileWriter outputFile;

         try
         {
            outputFile = new FileWriter( fc.getSelectedFile()  );
            outputFile.write( fileText, 0, (fTLen -1) );
            outputFile.close();
         }
         catch( IOException ioe)
         {
            JOptionPane dialog = new JOptionPane();
            dialog.showMessageDialog( this, "Error writing File", "Error",
JOptionPane.ERROR_MESSAGE );
            return;
         }
      }
      else
         return;
   }


   /*
   Method to read user supplied value of gamThres
   @param return  returns value of gamThres
   */
   private float setGamThres()
   {
      float retVal;
      String selected = (String) gammaThres.getSelectedItem();
      try
      {
         retVal = Float.parseFloat( selected );
      }
      catch( NumberFormatException nfe )
      {
         JOptionPane dialog = new JOptionPane();
         dialog.showMessageDialog( this, "Input value has to be in float format",
"Error", JOptionPane.ERROR_MESSAGE );
         gammaThres.setSelectedItem( Float.toString( gamThres ) );
         return gamThres;
      }
      if( retVal < 0 )
      {
         JOptionPane dialog = new JOptionPane();
```

```
         dialog.showMessageDialog( this, "Input value has to be > 0", "Error",
JOptionPane.ERROR_MESSAGE );
         gammaThres.setSelectedItem( Float.toString( gamThres ) );
         return gamThres;
      }
      return retVal;
   }


   /*
   Method to read user supplied values for xcoord, ycoord snd specType from GUI
   */
   private void updateDispPars()
   {
      try
      {
         xcoord = Integer.parseInt( x1Text.getText() );
         ycoord = Integer.parseInt( y1Text.getText() );
      }
      catch( NumberFormatException nfe )
      {
         JOptionPane dialog = new JOptionPane();
         dialog.showMessageDialog( this, "Coord data have to be in integer format",
"Error", JOptionPane.ERROR_MESSAGE );
         displayFlag = false;
         return;
      }

      if( xcoord >= xpix || xcoord < 0 || ycoord >= ypix || ycoord < 0 )
      {
         JOptionPane dialog = new JOptionPane();
         String message = "Pix coords have to be between ( 0, 0 ) and ( " + xpix + ",
" + ypix + " )";
         dialog.showMessageDialog( this, message, "Error", JOptionPane.ERROR_MESSAGE
);
         displayFlag = false;
         return;
      }

      specType = (String) typeOfSpec1.getSelectedItem();

   }


   /*
   Method to seach hiCube for possible galaxies and return result to textPanel
   */
   public void findSources()
   {

      boolean[][][] found = new boolean[xpix][ypix][zpix];
      float medianG2, medianG4, medianG8, modmeanG2, modmeanG4, modmeanG8, variance;
      float[] kvalues = new float[zpix];
      float[] valuesG2 = new float[ (zpix - 1) / 2 ];
      float[] valuesG4 = new float[ (zpix - 3) / 4 ];
      float[] valuesG8 = new float[ (zpix - 7) / 8 ];
      int medposG2 = ( zpix - 1 ) / 4;
      int medposG4 = ( zpix - 3 ) / 8;
      int medposG8 = ( zpix - 7 ) / 16;

      //calculate noise variance  value above which spectra will be excluded from
search
      float worstvar = 10 * hiCube.getNoise() * hiCube.getNoise();

      textPanel.append( "\n" );
      textPanel.append( "RESULTS OF RUNNING FINDER WIth GAMMA S/N = " + gamThres +
"\n" );
      textPanel.append( "\n");
      textPanel.append( "    xpix    ypix    zpix    Hann   Gamma S/N     \n"  );
      textPanel.append( "                              Type                \n"  );


      //search gamma spectra at each (x,y) position in hiCube
      for( int i = 1; i < xpix - 1; i++ )
      {
         for( int j = 1; j < ypix - 1; j++ )
         {
```

```
            //calculate average variance in spectrum
            for(int k=0; k < zpix; k++)
               kvalues[k] = hiCube.getVoxel( i, j, k ).getGVal();
             Arrays.sort( kvalues );
             variance = kvalues[(zpix-1)/2];

             //proceed if spectrum is not too noisy
             if( variance > 0  &&  variance < worstvar )
             {

                //calculate median gamma statistic in g2Val gamma Spectrum
                 for ( int l4 = 0; l4 < (zpix - 1)/2; l4++)
                    valuesG2[l4] = hiCube.getVoxel( i, j, ((2*l4) + 1) ).getG2Val();
                  Arrays.sort( valuesG2 );
                medianG2 = valuesG2[ medposG2 ];


                //calculate median gamma statistic in g4Val gamma Spectrum
                 for ( int l5 = 0; l5 < (zpix - 3)/4; l5++)
                    valuesG4[l5] = hiCube.getVoxel( i, j, ((4*l5) + 3) ).getG4Val();
                 Arrays.sort( valuesG4 );
                 medianG4 = valuesG4[ medposG4 ];


                //calculate median gamma statistic in g8Val gamma Spectrum
                for ( int l6 = 0; l6 < (zpix - 7)/8; l6++)
                   valuesG8[l6] = hiCube.getVoxel( i, j, ((8*l6) + 7) ).getG8Val();
                 Arrays.sort( valuesG8 );
                 medianG8 = valuesG8[ medposG8 ];
                 modmeanG8 = Stats.ModMean( valuesG8 );


                 //search for significant peaks  in g2Val gamma spectra
                 for( int k = 3; k < (zpix - 3) ; k = k + 2 )
                {


                    if( ( hiCube.getVoxel( i, j, k ).getG2Val()  > (gamThres * medianG2)
)  &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i, j,
k+2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i, j,
k-2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i+1,
j, k ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i-1,
j, k ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i,
j+1, k ).getG2Val() ) &&
                       ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i, j-
1, k ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i+1,
j, k+2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i-1,
j, k+2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i,
j+1, k+2 ).getG2Val() ) &&
                       ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i, j-
1, k+2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i+1,
j, k-2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i-1,
j, k-2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i,
j+1, k-2 ).getG2Val() ) &&
                       ( hiCube.getVoxel( i, j, k ).getG2Val() > hiCube.getVoxel( i, j-
1, k-2 ).getG2Val() ) &&
                        ( hiCube.getVoxel( i, j, k ).getH2Val() > 0  ) )
                   {
                      textPanel.append( FormatToString.displayInt( i, 8)  +
                                        FormatToString.displayInt( j, 8 ) +
                                        FormatToString.displayInt( k, 8 ) +
                                        FormatToString.displayString( "H2", 8) +
                                        FormatToString.displayFloat( ( hiCube.getVoxel(
i, j, k ).getG2Val()/medianG2), 12) + "\n");
                       found[i][j][k] = true;
```

```
                }
            }

            //search for significant peaks in g4val gamma spectra
            for( int k = 7; k < (zpix - 7) ; k = k + 4 )
            {
               if( (  hiCube.getVoxel( i , j, k ).getG4Val() > (gamThres *
medianG4) ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i, j,
k+4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i, j,
k-4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i+1,
j, k ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i-1,
j, k ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i,
j+1, k ).getG4Val() ) &&
                    ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i, j-
1, k ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i+1,
j, k+4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i-1,
j, k+4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i,
j+1, k+4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i, j-
1, k+4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i+1,
j, k-4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i-1,
j, k-4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i,
j+1, k-4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getG4Val() > hiCube.getVoxel( i, j-
1, k-4 ).getG4Val() ) &&
                   ( hiCube.getVoxel( i, j, k ).getH4Val() > 0 )  &&
                    ( found[i][j][k] == false ) && ( found[i][j][k+2] == false ) &&
                   ( found[i][j][k-2] ==false  ) && ( found[i][j][k+4] == false ) &&
                   ( found[i][j][k-4] ==false  ) && ( found[i+1][j][k] == false ) &&
                   ( found[i-1][j][k] ==false  ) && ( found[i][j+1][k] == false ) &&
                   ( found[i][j-1][k] ==false  ) && ( found[i+1][j+1][k] == false )
&&
                   ( found[i-1][j+1][k] ==false  ) && ( found[i+1][j-1][k] == false
) &&
                   ( found[i-1][j-1][k] ==false  ) && ( found[i+1][j+1][k+2] ==
false ) &&
                   ( found[i-1][j+1][k+2] ==false  ) && ( found[i+1][j-1][k+2] ==
false ) &&
                   ( found[i-1][j-1][k+2] ==false  ) && ( found[i+1][j+1][k-2] ==
false ) &&
                   ( found[i-1][j+1][k-2] ==false  ) && ( found[i+1][j-1][k-2] ==
false ) &&
                   ( found[i-1][j-1][k-2] ==false  ) && ( found[i+1][j][k+2] ==
false ) &&
                   ( found[i-1][j][k+2] ==false  ) && ( found[i][j+1][k+2] == false
) &&
                   ( found[i][j-1][k+2] ==false  )  && ( found[i+1][j][k-2] == false
) &&
                   ( found[i-1][j][k-2] ==false  ) && ( found[i][j+1][k-2] == false
) &&
                   ( found[i][j-1][k-2] ==false  )  )

                {
                   textPanel.append( FormatToString.displayInt( i, 8)  +
                                     FormatToString.displayInt( j, 8 ) +
                                     FormatToString.displayInt( k, 8 ) +
                                     FormatToString.displayString( "H4", 8) +
                                     FormatToString.displayFloat( ( hiCube.getVoxel(
i, j, k ).getG4Val()/medianG4), 12) + "\n");

                   found[i][j][k] = true;
                }
            }

            //search for significant peaks in g8Val gamma spectra
            for( int k = 15; k < (zpix - 15) ; k = k + 8 )
```

```
              {
                if(   ( ( hiCube.getVoxel( i, j, k ).getG8Val() > (gamThres *
medianG8)  ) ||
                     ( hiCube.getVoxel( i, j, k ).getG8Val() > (gamThres * modmeanG8)
) ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i, j,
k+8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i, j,
k-8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i+1,
j, k ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i-1,
j, k ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i,
j+1, k ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i, j-
1, k ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i+1,
j, k+8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i-1,
j, k+8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i,
j+1, k+8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i, j-
1, k+8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i+1,
j, k-8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i-1,
j, k-8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i,
j+1, k-8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getG8Val() > hiCube.getVoxel( i, j-
1, k-8 ).getG8Val() ) &&
                      ( hiCube.getVoxel( i, j, k ).getH8Val() > 0 ) &&
                      ( found[i][j][k] == false ) && ( found[i][j][k+2] == false ) &&
                      ( found[i][j][k-2] == false ) && ( found[i][j][k+4] == false )
&&
                      ( found[i][j][k-4] == false ) && ( found[i][j][k+6] == false )
&&
                      ( found[i][j][k-6] == false ) && ( found[i][j][k+8] == false )
&&
                      ( found[i][j][k-8] == false ) && ( found[i+1][j][k] == false )
&&
                     ( found[i-1][j][k] ==false  ) && ( found[i][j+1][k] == false ) &&
                     ( found[i][j-1][k] == false ) && ( found[i+1][j][k+2] == false )
&&
                     ( found[i-1][j][k+2] ==false  ) && ( found[i][j+1][k+2] == false
) &&
                     ( found[i][j-1][k+2] == false ) && ( found[i+1][j+1][k+2] ==
false ) &&
                     ( found[i-1][j+1][k+2] ==false  ) && ( found[i+1][j-1][k+2] ==
false ) &&
                     ( found[i-1][j-1][k+2] == false ) && ( found[i+1][j+1][k-2] ==
false ) &&
                     ( found[i-1][j+1][k-2] ==false  ) && ( found[i+1][j-1][k-2] ==
false ) &&
                     ( found[i-1][j-1][k-2] == false ) && ( found[i+1][j+1][k+4] ==
false ) &&
                     ( found[i-1][j+1][k+4] ==false  ) && ( found[i+1][j-1][k+4] ==
false ) &&
                     ( found[i-1][j-1][k+4] == false ) && ( found[i+1][j+1][k-4] ==
false ) &&
                     ( found[i-1][j+1][k-4] ==false  ) && ( found[i+1][j-1][k-4] ==
false ) &&
                     ( found[i-1][j-1][k-4] == false ) && ( found[i+1][j][k-2] ==
false ) &&
                     ( found[i-1][j][k-2] ==false  ) && ( found[i][j+1][k-2] == false
) &&
                     ( found[i][j-1][k-2] == false ) && ( found[i+1][j][k+4] == false
) &&
                     ( found[i-1][j][k+4] ==false  ) && ( found[i][j+1][k+4] == false
) &&
                     ( found[i][j-1][k+4] == false ) && ( found[i+1][j][k-4] == false
) &&
                     ( found[i-1][j][k-4] ==false  ) && ( found[i][j+1][k-4] == false
) &&
```

```
                    ( found[i][j-1][k-4] == false ) && ( found[i+1][j][k+8] == false
) &&
                    ( found[i-1][j][k+8] ==false  ) && ( found[i][j+1][k+8] == false
) &&
                    ( found[i][j-1][k+8] == false ) && ( found[i+1][j][k-8] == false
) &&
                    ( found[i-1][j][k-8] ==false  ) && ( found[i][j+1][k-8] == false
) &&
                    ( found[i][j-1][k-8] == false )   )


                {


                     if( modmeanG8 < medianG8 )
                        textPanel.append( FormatToString.displayInt( i, 8 )  +
                                          FormatToString.displayInt( j, 8 ) +
                                          FormatToString.displayInt( k, 8 ) +
                                          FormatToString.displayString( "H8", 8) +
                                          FormatToString.displayFloat( (hiCube.getVoxel( i,
j, k ).getG8Val()/modmeanG8), 12 ) + "\n" );
                     else
                        textPanel.append( FormatToString.displayInt( i, 8 )  +
                                          FormatToString.displayInt( j, 8 ) +
                                          FormatToString.displayInt( k, 8 ) +
                                          FormatToString.displayString( "H8", 8) +
                                          FormatToString.displayFloat( (hiCube.getVoxel( i,
j, k ).getG8Val()/medianG8), 12 ) +   "\n" );

                    found[i][j][k] = true;
                 }
              }
           }
        }
     }

     JOptionPane dialog = new JOptionPane();
     dialog.showMessageDialog( this, "Finding completed. Results are on Display
Panel.",
                                  "Information", JOptionPane.PLAIN_MESSAGE );
  }


  //GUI related instance variables
  private JScrollPane scroll;
  private JMenuItem j1, j2, j3, j4, j5, j6, j7, j8, j9;
  private JMenu m1, m2, m3;
  private JMenuBar mbar;
  private JButton button1, button2, button3, button4, button5, button6, button7, x1B,
x1F;
  private ImageIcon image1, image2, image3, image4, image5, image6;
  private JLabel x1, y1, jspace, jspace1, jspace2;
  private JTextField x1Text, y1Text;
  private JPanel panel1, panel2, panel3, panel3n;
  private JComboBox typeOfSpec1, gammaThres;
  private JLabel label1, selSpec, space1, space2, space3, gLab;
  private JTabbedPane tabbedPanel;
  private boolean fileFlag, displayFlag;

  //non-GUI instance variables
  private HiCube hiCube;
  private String specType;
  private JTextArea textPanel;
  private DisplayPanel dispPanel;
  private int xpix, ypix, zpix, xcoord, ycoord;
  private float gamThres;

}
```

Source code for DisplayPanel.java

```
/*
   Class to define a DisplayPanel object
   Part of GammaFinder appliciation.
   @author Peter J. Boyce
   @version 1.0 2003/09/15
*/
import java.awt.event.*;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.border.*;
import java.lang.Math;


public class DisplayPanel extends JPanel
{
   //instance variables
   private Voxel[][][] voxVals;
   private int xc, yc, nzpix, sf, specType;

   //constructor
   public DisplayPanel()
   {
       super();
       setOpaque(false);
       setPreferredSize( new Dimension( 760, 320 ) );
   }


   //method to update spectrum displayed
   public void updateSPanel(Voxel[][][] voxelValues, int x, int y, String inSpecType )
   {
      voxVals = voxelValues;
      xc = x;
      yc = y;
      nzpix = voxVals[0][0].length;
      String spectrumType = inSpecType;

      if( spectrumType.compareTo( "Raw Spectrum" ) == 0 )
      {
         sf = -1000;
         specType = 1;
      }
      else if ( spectrumType.compareTo( "Hann2" ) == 0 )
      {
         sf = -1000;
         specType = 2;
      }
      else if ( spectrumType.compareTo( "Hann4" ) == 0 )
      {
          sf = -1000;
         specType = 3;
      }
      else if ( spectrumType.compareTo( "Hann8" ) == 0 )
      {
         sf = -1000;
         specType = 4;
      }
      else if ( spectrumType.compareTo( "Gam2" ) == 0 )
      {
         sf = -200000;
         specType = 5;
      }
      else if ( spectrumType.compareTo( "Gam4" ) == 0 )
      {
        sf = -600000;
         specType = 6;
      }
      else if ( spectrumType.compareTo( "Gam8" ) == 0 )
      {
        sf = -1000000;
         specType = 7;
      }
      else if ( spectrumType.compareTo( "Gamma Spectrum" ) == 0 )
```

```
      {
         sf = -100000;
          specType = 8;
      }

      repaint();
   }


   public void paintComponent(Graphics graf)
   {
      Graphics2D g2 = (Graphics2D) graf;
      g2.draw( new Line2D.Double(5, 200, 750, 200 ));
      g2.setBackground( Color.white );

      GeneralPath polyline = new GeneralPath(GeneralPath.WIND_EVEN_ODD,1024);

      if( specType == 1 )
      {
         polyline.moveTo( 5*2, 200 + (int) (voxVals[xc][yc][0].getVoxVal() * sf) );

         for ( int j = 1; j < nzpix; j++)
         {
            polyline.lineTo( (j+5)*700/nzpix, 200 + (int) (
voxVals[xc][yc][j].getVoxVal() * sf ) );
         }
      }
      else if ( specType == 2 )
      {
         polyline.moveTo( 6*2, 200 + (int) (voxVals[xc][yc][1].getH2Val() * sf) );

        for(int j = 3; j < (nzpix - 1) ; j = j + 2 )
         {
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getH2Val() * sf ) );
         }
      }
      else if ( specType == 3 )
      {
         polyline.moveTo( 8*2, 200 + (int) (voxVals[xc][yc][3].getH4Val() * sf) );

         for( int j = 7; j < ( nzpix - 3 ) ; j = j + 4)
         {
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getH4Val() * sf ) );
         }
      }
      else if ( specType == 4 )
      {
         polyline.moveTo( 12*2, 200 + (int) (voxVals[xc][yc][7].getH8Val() * sf) );

         for( int j = 15; j < ( nzpix - 7 ) ; j = j + 8)
         {
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getH8Val() * sf ) );
         }
      }
      else if ( specType == 5 )
      {
         polyline.moveTo( 6*2, 200 + (int) (voxVals[xc][yc][1].getG2Val() * sf) );


        for(int j = 3; j < (nzpix - 1) ; j = j + 2 )
         {
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getG2Val() * sf ) );
         }
      }

      else if ( specType == 6 )
      {
         polyline.moveTo( 8*2, 200 + (int) (voxVals[xc][yc][3].getG4Val() * sf) );


        for(int j = 7; j < (nzpix - 3) ; j = j + 4 )
         {
```

```
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getG4Val() * sf ) );
         }
      }
       else if ( specType == 7 )
      {
         polyline.moveTo( 12*2, 200 + (int) (voxVals[xc][yc][7].getG8Val() * sf) );


        for(int j = 15; j < (nzpix - 7) ; j = j + 8 )
         {
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getG8Val() * sf ) );
         }
      }

      else if ( specType == 8 )
      {
         polyline.moveTo( 5*2, 200 + (int) (voxVals[xc][yc][0].getGVal() * sf) );


        for(int j = 1; j < nzpix; j++ )
         {
            polyline.lineTo( (j+5)*700/nzpix , 200 + (int) (
voxVals[xc][yc][j].getGVal() * sf ) );
         }
      }

      g2.draw( polyline );

   }

}
```

Source code for FormatDisplay.java

```
/*
   Class containg static method toformat displayed output
   Part of the GammaFinder application.
   @author Peter J. Boyce
   @version 1.0 2003/09/15
*/
import java.lang.Integer;
import java.lang.Double;
public class FormatDisplay
{

   /*
   method to convert  integer to string of desired length
   @param number  integer to be converted to string
   @param width   desired length of string
   @param return  returns String of desired length
   */
   public static String displayInt(int number, int width)
   {
      String outputString;
      int itemLength, count;
      outputString = Integer.toString( number );
      itemLength = outputString.length();
      if ( itemLength < width )
      for ( count = itemLength + 1; count <= width; count = count + 1 )
         outputString = " " + outputString;
      return outputString;
   }


   /*
   method to convert float to string of desired length
   @param number  float to be converted to string
   @param width   desired length of string
   @param return  returns String of desired length
   */   public static String displayFloat( float number, int width)
   {
      String outputString;
      int itemLength, count;
      outputString = Float.toString( number );
      itemLength = outputString.length();
      if ( itemLength < width )
      for ( count = itemLength + 1; count <= width; count = count + 1 )
         outputString = " " + outputString;
      return outputString;
   }

   /*
   method to convert String to another String of desired length
   @param inputLine  String to be converted
   @param width      desired length of string
   @param return     returns String of desired length
   */
   public static String displayString( String inputLine, int width)
   {
      String outputString = inputLine;
      int itemLength = outputString.length();
      if ( itemLength < width )
      {
      for ( int count = itemLength + 1; count <= width; count = count + 1 )
         outputString = " " + outputString;
      }
      else
        outputString = outputString.substring(0, width);


      return outputString;
   }

}
```

Source code for Stats.java

```
/*
   Class containg static method to find modal value of a float array
   Part of the GammaFinder application.
   @author Peter J. Boyce
   @version 1.0 2003/09/15
*/
import java.util.*;
public class Stats
{

   /*
   Method to find the modal value of a float array
   @param inArray  the float array within which the odal value is to be found
   @param return   returns the modal value
   */
   public static float ModMean( float[] inArray)
   {
      float medmod;
      int length = inArray.length;
      int[] count = new int[20];

      for( int i = 0; i < length; i++)
      {
         for( int j = 0; j < 20; j++ )
         {
            if( ( inArray[i] >= (j * 0.00001) + 0.00001 ) &&
                ( inArray[i] < ( ( j + 1 ) * 0.00001) + 0.00001 ) )
               count[j]++;
         }
      }


      int temp = count[0];
      int jmode = 0;
      for ( int j = 1; j < 20; j++ )
      {
         if ( count[j] > temp )
         {
            temp = count[j];
            jmode = j;
         }
      }


      int nlength = count[jmode];
      float jlower = (float) ( ( jmode * 0.00001 ) + 0.00001 );
      float jupper = (float) ( ( ( jmode + 1 ) * 0.00001 ) + 0.00001 );


      float[] newArray = new float[nlength];


      int z = 0;
      for( int i = 0; i < length; i++ )
      {
         if( inArray[i] >= jlower && inArray[i] < jupper )
         {
             newArray[z] = inArray[i];
                z++;
         }
      }


      if( nlength == 0 )
          medmod = 20;
      else if( nlength % 2 == 0 )
         medmod = newArray[ ( nlength / 2 ) - 1];
      else
          medmod = newArray[ ( nlength - 1 ) / 2 ];

      return medmod;
   }
}
```

References

Banks, G.D. et al. 1999. New galaxies discovered in the first blind HI survey of the Centaurus A group. *Astrophysical Journal*, **524**, 612.

Barnes, D.G. et al. 2001. The HI Parkes All Sky Survey: southern observations, calibration and robust imaging. *Monthly Notices of the Royal Astronomical Society*, **322**, 486.

Blackman, R.B. and Tukey, J.W. 1959. Particular Pairs of windows. In: *The measurement of power spectra, from the point of view of communications engineering*. New York: Dover. p.98.

Boyce, P.J. et al. 2001. A blind HI survey of the M81 group. *Astrophysical Journal*, **560**, L127.

Branch, D. 1998. Type IA supernovae and the Hubble constant. *Annual Review of Astronomy and Astrophysics*, **36**, 17

Burke, B.F and Graham-Smith, F. 2002. *An introduction to radio astronomy*. Cambridge: Cambridge University Press.

Chuzhanova, N.A., Jones, A.J. and Margetts, S. 1998. Feature selection for genetic sequence classification. *Bioinformatics*, **14**(2), 139.

Duffet-Smith, P., 1988. *Practical astronomy with your calculator*. Cambridge: Cambridge University Press.

Durrant, P.J. 2001. *winGamma*™: A non-linear data analysis and modelling tool with applications to flood prediction. *PhD Thesis*, Cardiff University.

Evans, D. 2002. Data derived estimates of noise for unknown smooth models using near neighbour asymptotics, *PhD Thesis*, Cardiff University.

Evans, D. and Jones, A.J. 2002. A proof of the Gamma test. *Proceedings of the Royal Society Series A*, **458**, 2759.

Freidman, J.H., Bentley, J.L., Finkel, R.A. 1979. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, **3**(3), 209.

Gooch, R. 1995. In: Shaw, R.A., Payne, H.E. & Haynes, J.E. eds. *ASP Conf Series 77, Astronomical Data Analysis Software and Systems IV*. San Francisco: ASP. p.144.

Jones, A.J., Evans, D., Margetts, S. and Durrant, P.J., 2002a. The Gamma test. In: Sarker, R et al. eds. *Heuristic and Optimization for Knowledge Discovery*. Hershey: Idea Group Publishing.

Jones, A.J., Tsui, A.P.M. and de Oliveira, A.G. 2002b. Neural models of arbitrary chaotic systems: construction and the role of time delayed feedback in control and synchronization. *Complexity International*, **9**, paperID: tsui01.

Kilborn, V.A. et al. 2000. An extragalactic HI cloud with no optical counterpart. *Astronomical Journal*, **120**, 1342.

Kilborn, V.A. 2001, A catalogue of HI-selected galaxies from the south celestial cap region of sky. *PhD Thesis*, University of Melbourne.

Koncar, N. 1997. Optimisation methodologies for direct inverse neurocontrol. *PhD Thesis*, Imperial College, London.

Lang, R.H. et al. 2003. First results from the HI Jodrell All Sky Survey: inclination-dependent selection effects in a 21-cm blind survey. *Monthly Notices of the Royal Astronomical Society,* **342**, 738.

Meyer, M.J. et al. 2003. The HIPASS Catalogue: I – Data Presentation. *Monthly Notices of the Royal Astronomical Society*, submitted.

Minchin, R.F. 2001. Properties of galaxies found in a deep blind neutral hydrogen survey. *PhD Thesis*, University of Wales, Cardiff.

de Oliveira, A.G 1999. Synchronization of chaos and applications to secure communications, *PhD Thesis,* Imperial College, London.

Rohlfs, K. 1996. *Tools of Radio Astronomy*. New York: Springer-Verlag.

Ryder, S.D. et al. 2001. HIPASS detection of an intergalactic gas cloud in the NGC 2442 group. *Astrophysical Journal*, **555**, 232.

Sault, R.J., Teuben, P.J. and Wright, M.C.H., 1995, A retrospective view of MIRIAD. In: Shaw, R.A., Payne, H.E. & Haynes, J.E. eds. *ASP Conf Series 77, Astronomical Data Analysis Software and Systems IV*, ASP, San Francisco, p.433.

Stefansson, A., Koncar, N. and Jones, A.J. 1997. A note on the Gamma test. *Neural Computing and Applications*, **5**, 131.

Tsui, A.P.M., Jones, A.J. and de Oliveira, A.G. 2002. The construction of smooth models using irregular embeddings determined by a Gamma test analysis. *Neural Computing and Applications*, **10**(4), 18.

Zwaan, M.A. et al. 2003a. The 1000 brightest HIPASS galaxies: the HI mass function and $\oslash_{HI}$. *Astronomical Journal*, **125**, 2842.

Zwaan, M.A. et al. 2003b. The HIPASS catalogue: II – completeness, reliability and parameter accuracy. *Monthly Notices of the Royal Astronomical Society*, submitted.

Bibliography

Sahni, S. 2000. *Data structures, algorithms, and applications in Java*. Singapore: McGraw-Hill.

Gutz, S. 2000. *Up to speed with swing*. Greenwich: Manning.

Goodrich, M.T. and Tamassia, R. 2000. *Data structures and algorithms in Java*. New York: JohnWiley.

Davies, J.I., Impey, C., Phillipps, S. (eds)., 1999. *The low surface brightness Universe*. San Francisco: ASP.

Stevens, P. and Pooley, R. 200. *Using UML, software engineering with objects and components.* New York: Addison-Wesley.