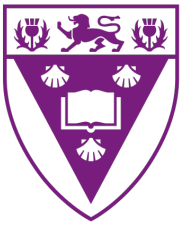


RHODES UNIVERSITY

Where leaders learn

RHODES UNIVERSITY

MASTERS THESIS

TiRiFiG, A Graphical 3D Kinematic Modelling Tool

Author:
Samuel Nyarko TWUM

Supervisors:
Dr. Gyula I. G. JÓZSA
Dr. Peter KAMPHUIS
Prof. Oleg SMIRNOV

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Centre for Radio Astronomy Techniques and Technologies
Department of Physics and Electronics

January 21, 2019

Declaration of Authorship

I, Samuel Nyarko Twum, declare that this thesis titled, "TiRiFiG, A Graphical 3D Kinematic Modelling Tool" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

Galaxy kinematics is of crucial importance to understanding the structure, formation and evolution of galaxies. The studies of mass distributions giving rise to the missing mass problem, first raised by Zwicky (1933), give us an insight into dark matter distributions which are tightly linked to cosmology. Neutral hydrogen (H I) has been widely used as a tracer in the kinematic studies of galaxies. The Square Kilometre Array (SKA) and its precursors will produce large H I datasets which will require kinematic modelling tools to extract kinematic parameters such as rotation curves. TiRiFiC (Józsa et al., 2007) is an example of such a tool for 3D kinematic modelling of resolved spectroscopic observations of rotating disks in terms of the tilted-ring model with varying complexities. TiRiFiC can be used to model a large number (20+) of parameters which are set in a configuration file (`.def`) for its execution. However, manually editing these parameters in a text editor is uncomfortable. In this work, we present TiRiFiG, **Ti**lted **Ri**ng **Fi**tting **G**UI, which is the graphical user interface that provides an easy way for parameter inputs to be modified in an interactive manner.

Acknowledgements

First, I thank Prof. Oleg Smirnov and the SKA for granting me this bursary to study for this program; and Theophilus Narh for informing me about this bursary opportunity. Special thanks to my supervisors, Dr. Gyula Jozsa and Dr. Peter Kamphuis, who supported and mentored me to complete my research and write my thesis; and to Eric Kamau and Francois Joubert for their input in the development of TiRiFiG.

For a memorable, productive and enjoyable stay at Rhodes, thanks to: Lerato Sebokolodi, Diana Yeboah, James Chege, Kwazi Nhlakanipho, Gift Sichone, Siyanda Matika and all postgraduate students at the Department of Physics and Electronics; and to Prof. Chitambo, Ronel and the entire staff for their work, which has provided us with the needed support and the right conditions for productive academic work. I specially thank Cyndie Russeeawon and Ulrich Mbou Sob for proofreading my work and sending useful corrections and comments. I also thank Mr. and Mrs. Akoto-Danso, the Allen Flat household and the Rock Family Church for their prayers, love and care.

I thank my family, mentors and friends back home, for their continuous support and encouragement throughout my period of study. Final thanks goes to Almighty God for His grace and providence during the course of my study. This achievement would not have been possible without Him.

Contents

List of Figures

List of Tables

Dedicated to my parents

Chapter 1

Preview to Theory and Thesis Objective

"*The dinosaurs became extinct because they didn't have a space program. And if we become extinct because we don't have a space program, it'll serve us right!*" ~L. Niven

1.1 Astronomy

Throughout history, mankind has been fascinated by the night sky. Anyone who looks up in the night recognises (apart from the moon) the stars, the planets, later on their moons and other bodies in the solar system and the patterns they are forming in the sky. These patterns, also known as stellar constellations, were interpreted as manifestations of the existence of higher entities and enriched mythology, but they also served for practical purposes in the art of navigation. Later on, it became apparent that we can learn a lot about physics and the universe we live in by observing the night sky over the course of the year. One of the most dramatic discoveries was the replacement of the Earth as the centre of the solar system (geocentric model) by a picture in which the sun is at the centre of the solar system (heliocentric model). Some of the well known scientists whose work contributed significantly to our understanding of the universe include Galileo Galilei, Nicolaus Copernicus, Johannes Kepler and Albert Einstein. Although there is multiwavelength radiation emission coming from space, these contributions were made during a time when the only means we peered into the universe was through optical telescopes. Currently, we have multiwavelength astronomy which observes the sky in radio, infrared, optical, x-ray and gamma-ray. However, this thesis discusses only radio astronomy in Sect. 1.2 below. Our desire to fully understand the universe and our place in the universe by answering questions about the age of the universe, the history of the universe, the existence of (intelligent) life elsewhere, chemical elements and their properties, just to mention but a few, is intrinsic to human existence.

1.2 Radio Astronomy

Prior to and in the early 20th century, observations of the universe were conducted in the visible regime of the electromagnetic spectrum and around the 1920s, the majority of scientists believed that our own Milky Way galaxy was the entire universe. Deeper surveys, however, supplied new knowledge and deeper insights into our solar system and the Milky Way galaxy. This revealed the existence of other nearby galaxies (Hubble, 1929). Although we gained new knowledge, we were still limited in our understanding of the universe as we were missing the information buried in the other regions of the electromagnetic spectrum.

Then, in 1930, Karl Jansky, working on investigating the sources of radio transmission interference at Bell Labs, made a serendipitous discovery of radio signals coming from a source in the sky (Jansky, 1933). He first thought of the sun as the likely source. However, further investigation by Jansky revealed that this signal was actually coming from a radio source other than the sun. This discovery opened up the possibility to explore the night sky and beyond in the radio spectrum. The discovery of Jansky won the interest of another radio engineer called Grote Reber who wanted to probe further into the radio waves coming from the cosmos. Reber privately built a parabolic dish of about 9 m in diameter to study the radio waves and confirmed Jansky's discovery. After three successive upgrades of his dish's receiver, Reber detected a radio signal from the Milky Way galaxy in 1938 to confirm Jansky's discovery. Reber did additional surveys which were the first to detect the well-known radio galaxy Cygnus A. Results from the surveys were later published in engineering and astronomy journals[1].

[1] https://www.nrao.edu/archives/Reber/reber$_{p}$*ublist.shtml*

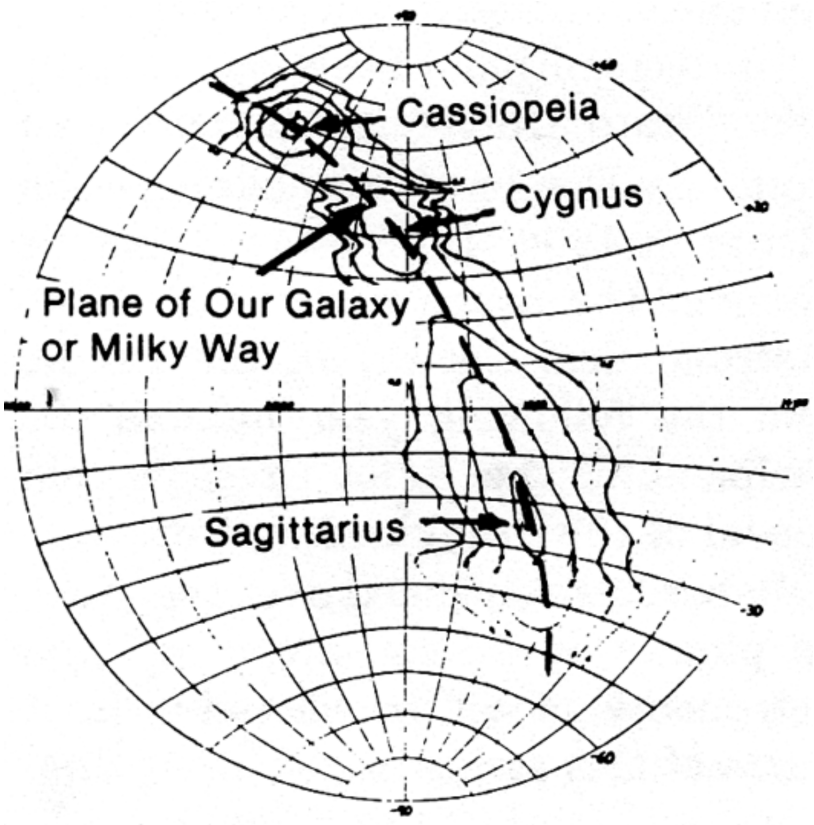


FIGURE 1.1: Historical image confirming Jansky's detection published in Reber (1944).
Image Credit: Cosmic Search Issue 13 Page 14

Figure 1.1 is a map of the radio sky showing radiation concentrated along the plane of the Milky Way. To date, radio telescopes have been built and stationed at various locations for observations of the sky to provide astronomers with much more detailed information to understand our universe.

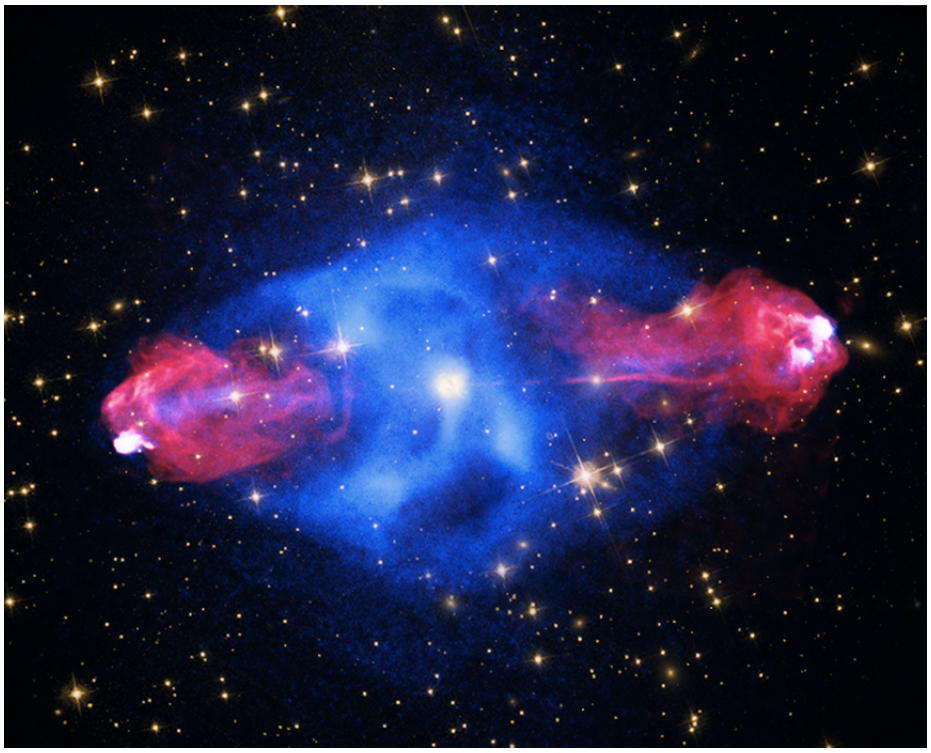


FIGURE 1.2: Light from Cygnus A.
Image Credit: X-ray: NASA/CXC/SAO; Optical: NASA/STScI; Radio: NSF/NRAO/AUI/VLA.

The detailed image in Fig. 1.2 reveals the active galaxy, Cygnus A, in light across the electromagnetic spectrum with x-ray data in blue, radio emission in red and confined to the yellow hues, optical wavelength data.

1.2.1 Radio Interferometry

Despite the fact that observations with radio telescopes provide an additional window to observe astrophysical phenomena unseen in the optical, single dish observations are hampered by the fact that they do not provide an angular resolution comparable to the optical. This is mainly because the resolving power of a telescope scales proportionally to the wavelength, making it much harder to resolve fine details at the much longer radio wavelengths. Expressed mathematically, we have

$$\theta_R \sim \frac{\lambda}{D} \tag{1.1}$$

where θ_R, λ and D represent the angular resolution of the telescope, wavelength of the observed radiation and the diameter of the telescope respectively. The equivalent of a 1 m optical telescope observing at a violet wavelength (400 nm) would have an angular resolution of $0.1''$. On the other hand, a 65 m radio telescope observing 5 cm radio wavelength would have an angular resolution of $190''$. Single-dish radio telescopes are hence not suitable for several astronomical purposes that require higher angular resolution, for example in extragalactic studies, where it is required to spatially resolve

the observed objects such as distant galaxies.

In order to achieve better angular resolution in the radio spectrum, we would need to build kilometre sized dishes. However, that option would come with mechanical and financial challenges. Instead, by building a group of individual telescope dishes, which get electronically coupled, one can point the dishes to the same patch of the sky and overcome the problem of poor resolution. The arrays are equipped with electronics that average and correlate the signals and store the correlated signals as visibilities. These can be used to reconstruct the sky brightness at a much higher resolution than that of an individual antenna in the array. It can be calculated as in Eq. 1.1, by replacing the antenna diameter with the largest distance between any of the several antennas (Thompson et al., 2017). Fig. 1.3 is a sketch of a two element interferometer.

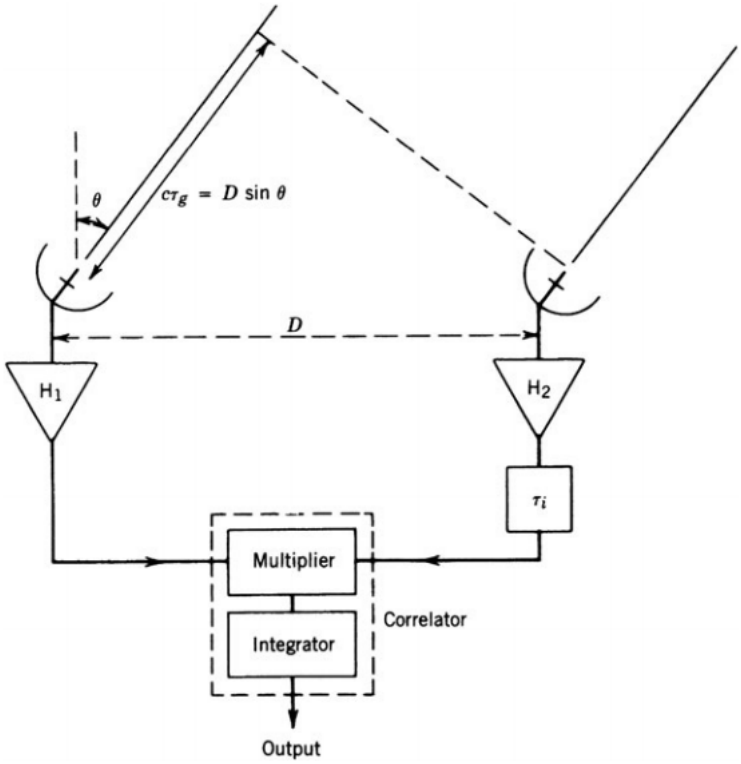


FIGURE 1.3: A two element interferometer.
Image Credit: Thompson et al. (2017).

Radio interferometry hence simulates telescopes with diameters equal to the longest baseline (i.e. distance between antenna pair). Examples of radio interferometers include the Jansky Very Large Array (JVLA) and the Atacama Large Millimeter Array (ALMA). The JVLA has a maximum baseline of 35 km and observes electromagnetic radiation at wavelengths ranging from 7 mm to 4 m, corresponding to a resolution up to 1.4" when observing the 21 cm line. ALMA with a maximum baseline of 15 km and observing wavelength between 0.3 mm and 9 mm obtains up to 5 mas resolution which is better than the Hubble Space Telescope's resolution (0.05") while observing at 0.03 mm. The Very Long Baseline Array (VLBA) is an even more fascinating

telescope with a maximum baseline size of 8000 km. It represents a system of ten radio-telescope antennas each with a dish 25 m in diameter from Mauna Kea on the big island of Hawaii to St. Croix in the U.S. Virgin Islands shown in Fig. 1.4. With such a long baseline, we can obtain a resolution of 0.2 mas at 1 cm wavelength.

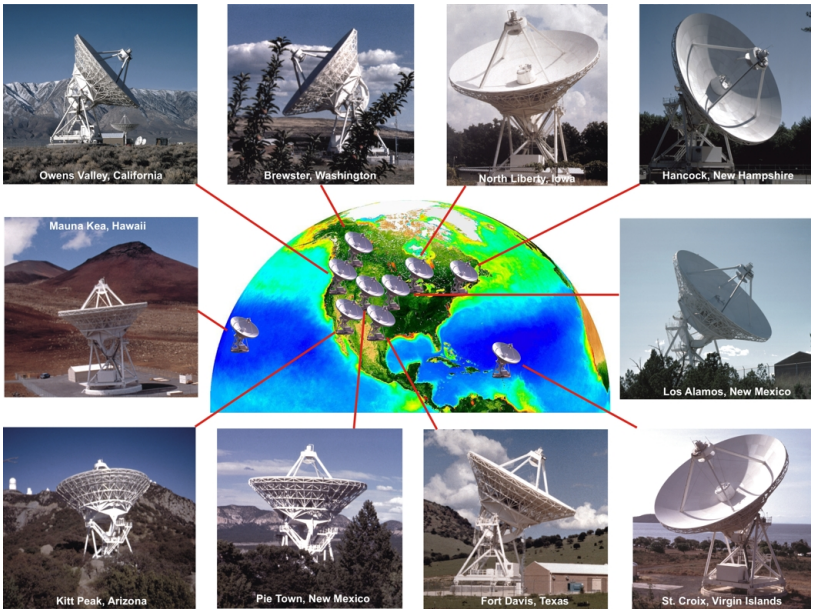


FIGURE 1.4: VLBA Montage.
Image courtesy of NRAO/AUI and Earth image courtesy of the SeaWiFS Project NASA/GSFC and ORBIMAGE.

Different arrays have different sizes, baselines and configurations but they all do the same job of measuring visibilities from the sky. What we measure with the interferometer is the so-called "coherence" or "visibility". The measured visibilities are related to the sky brightness by the Van Cittert-Zernike theorem (Thompson et al., 2017)

$$V(u,v,w) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(l,m)I(l,m)e^{-2\pi i[ul+vm+w(\sqrt{1-l^2-m^2}-1)]}\frac{dldm}{\sqrt{1-l^2-m^2}}, \tag{1.2}$$

where $V(u,v,w)$ represents the measured visibilities at the baseline coordinates (u,v,w), $I(l,m)$ represents the sky brightness distribution and $A(l,m)$ represents the antenna response with respect to the effective collecting area and direction of the incoming photons.

In an interferometric experiment, the coherence is sampled at enough (u,v) points to synthesise the resolution of a large aperture with the maximum u and v that can be measured, most frequently making use of the Earth rotation to fill the "UV-plane". And this is the basis for aperture synthesis, a technique for which M. Ryle was awarded the Nobel Prize in 1974. Since the Earth rotates, the (u,v) points constantly change. With this knowledge, we can fill the (u,v) plane using the Earth's rotation. The coarseness of sampling of the visibility plane determines how well the image can be reconstructed. An ideal interferometer would have infinite sampling so that we could

have our image perfectly reconstructed. However, due to engineering and physical limitations, we do not fully or infinitely sample the sky. Instead, we are limited to a discrete sampling function. The Fourier transform of the sampling function (a sum of δ-functions, each of which is multiplied with a weight) is the point spread function (PSF). Since there is only limited sampling of the visibility plane, the image produced is not exactly the sky brightness $I(l, m)$ but some modified version of the sky brightness $I^d(l, m)$. The true sky brightness convolved with the PSF returns the Fourier transform of the sampled visibilities.

In order to restore the original image that has been corrupted by the convolution kernel (PSF), we perform an operation known as deconvolution to undo the effects of the convolution to enable astronomers to have quality images to do the science. In the deconvolution process, the sampled visibilities are interpolated onto the unsampled regions of the (u, v) plane to obtain a model of the true sky $I(l, m)$ compatible with the data. This requires assumptions about the true sky to fill the unmeasured parts of the Fourier plane.

Visualising Datasets from Interferometers

Since radio telescopes also serve as spectrometers, the visibilities are stored as datasets that have several channels, i.e., the observing band is cut in many pieces with a small frequency width. When the original sky images are restored by integrating over all channels, typical for radio continuum observations, a sky image for each channel can be produced, thus creating a data cube.

In such a cube, each channel can be interpreted as the line-of-sight velocity of the medium (see Fig. 1.5) in the case of a line observation. A voxel represents a unit from the three dimensional data cube with two axes representing spatial dimensions (right ascension and declination) and the third axis representing the spectral or velocity dimension.

Most available software packages for astronomers come with the capability to do both 2D and 3D visualisation of data cubes. Ekers and Allen (1975) were amongst those who did pioneering work in data visualisation in radio astronomy. They investigated techniques of displaying single datasets and the possibility of extending to multiple datasets. Currently, KARMA (Gooch, 1996) (see Fig. 1.6) is a visualisation tool widely being used among radio astronomers. Other available visualisation software used by astronomers includes DS9 (Smithsonian Astrophysical Observatory, 2000), TIGGER[2] and SLICERASTRO (Punzo et al., 2017). Besides these standalone visualisation tools, other commonly used astronomical packages also have data cube viewers bundled in as well. CASA (McMullin et al., 2007) and AIPS[3] are classic examples. Aside from these, there are libraries (PYFITS, etc) available in the PYTHON programming language that can be used to visualise data cubes in both 2D and 3D. Perhaps, the challenge faced when using these tools, especially in the

[2]https://github.com/ska-sa/meqtrees/wiki/Tigger
[3]http://www.aips.nrao.edu/index.shtml

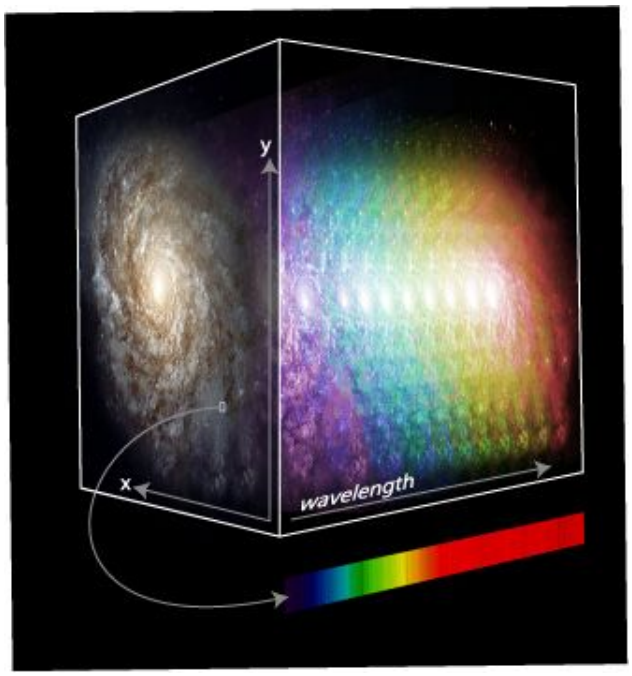


FIGURE 1.5: Example Data cube.
Image Credit: ScienceWise Magazine.

case of rendering, has to do with the limited computing resources, specifically RAM and CPU, being overly exploited. With the influx of even more data from observations from new instruments and the desire for better visualisation tools, many more tools will emerge with state of the art options to allow astronomers to have a better perspective of the data they are working with.

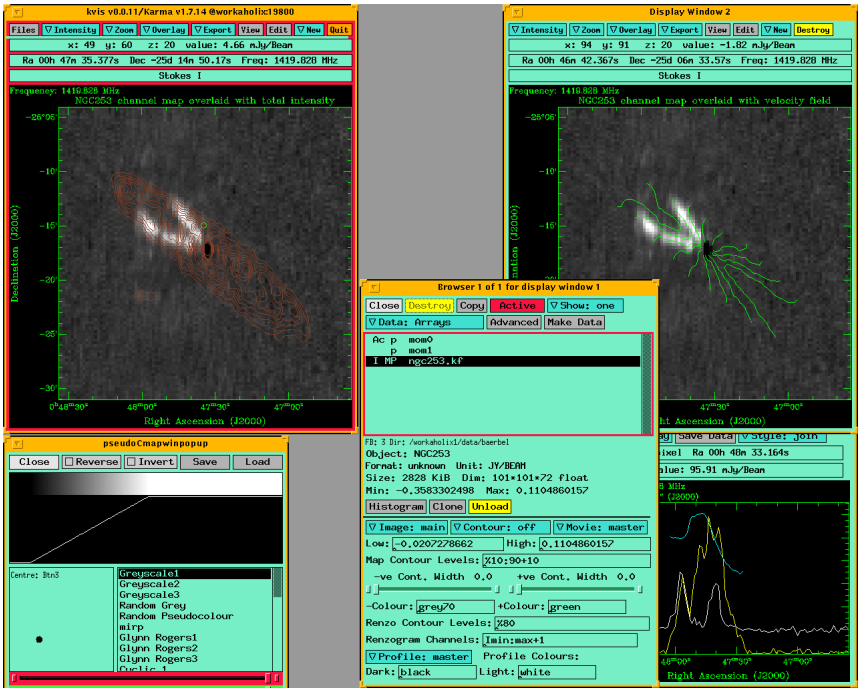


FIGURE 1.6: KVIS dataset viewer with multiple windows.
Image Credit: The Karma Home Page

1.3 Line Emission and the H I Line

Electromagnetic radiation travels across the vacuum of space at the speed of light - $2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$. Astronomers learn about the universe by decoding messages buried in the radiation from radio sources. These radio sources produce either continuum emission or line emission. Continuum radiation covers a wide range of frequencies and thus, forms a continuous spectrum. Thermal radiation and synchrotron emission are examples of continuum emission mechanisms. Line radiation is emitted at only one specific wavelength. This makes it easy to map line emissions to either red or blue which is indicative of redshift or blueshift - a useful piece of information astronomers decode from line radiation to determine whether a source is moving away from us or towards us. Objects moving towards us will be observed at a shorter wavelength ("bluer") and an object will appear to have a longer wavelength ("redder") if it is moving away from us. This phenomenon is called the Doppler effect and is useful in determining the relative line-of-sight velocities of two nearby objects if the rest frequency is sufficiently well determined. Measuring the redshift and the blueshift of objects provides a means to measure the kinematics of astronomical objects.

Electromagnetic radiation comprises a diverse spectrum of wavelengths with the visible spectrum being a portion of the entire range. The other wavelengths are far too short (too blue) or far too long (too red) to see. The entire electromagnetic spectrum covers a wide wavelength range from gamma rays, the size of an atomic nucleus to radio waves, many meters long. A diagram of the electromagnetic spectrum, showing

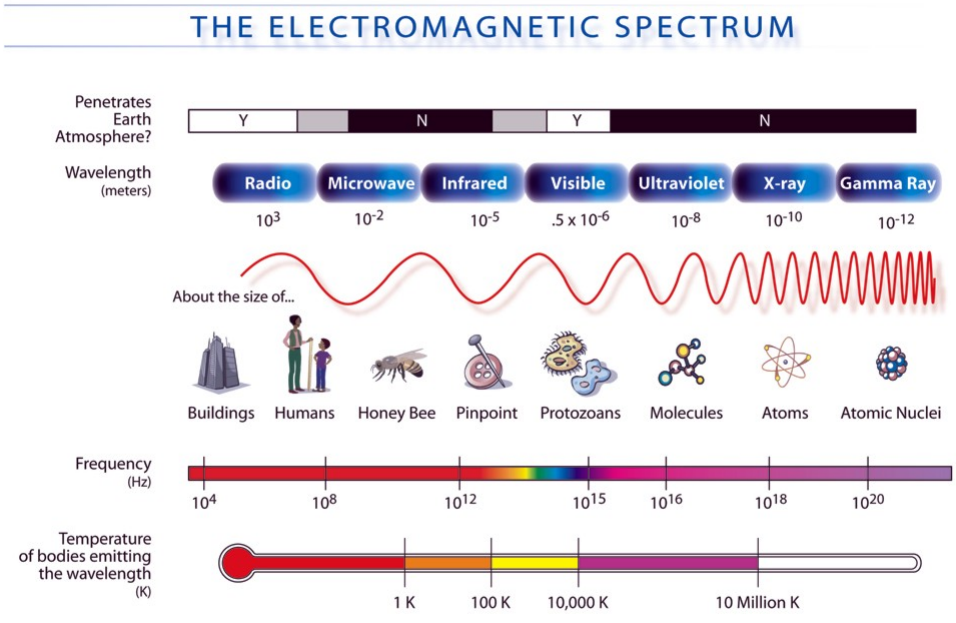


FIGURE 1.7: The Electromagnetic Spectrum
Image Credit: NASA.

various properties across the range of frequencies and wavelengths is shown in Fig.

1.7. The radiation from extraterrestrial bodies travels through gas and clouds which contain atoms. These atoms absorb and release energy in the form of photons. Electrons of an atom occupy discrete energy levels. Every time there is an energy change, we have an associated frequency or wavelength of light emitted. As the electron of an excited atom changes its energy level from higher to lower, the atom releases photons at a specific wavelength which produces line emission. Just as each atom can only give off specific colours (radiation at a specific frequency), each chemical element can only absorb specific colours and the colours absorbed are at the same wavelength as those emitted. The absorption and emission lines, thus, emerge as a result of energy transfer from the electrons of an atom as shown in Fig. 1.8 below. The various spectral lines

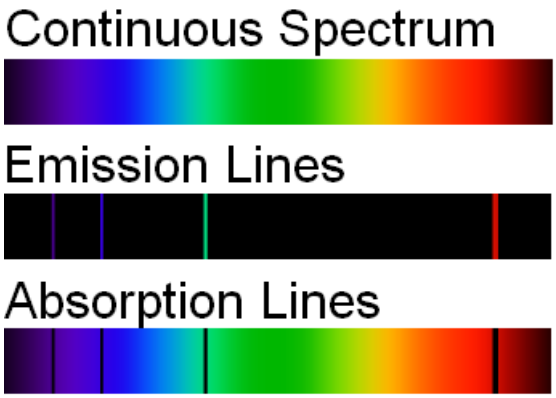


FIGURE 1.8: Spectral Lines.
Image Credit: Public domain.

arising from the transition of the electrons of an atom from higher energy states to a definite energy state form a spectral series. Using the Rydberg formula, wavelengths of spectral lines can be determined for chemical elements which are hydrogen-like. Every atom has its unique signature with respect to the emission line and absorption line. This complimentary phenomenon of absorption and emission is very useful to astronomers in telling the constituents of radio objects out in space and even derive in more detail the percentage quantities of chemical elements detected. This is how we are able to tell that hydrogen is the most abundant element in the universe.

Neutral hydrogen is an electrically neutral atom with one proton and one electron. The common notation used in astronomy is H I where I is the Roman numeral denoting neutral atoms. H I is located throughout galaxies as clouds and does not emit radiation at visible wavelengths. In its lowest energy state, the proton and electron spin in opposite directions. However, as the hydrogen atom gains small amounts of energy through collisions with atoms, it may align the spin of its electron to that of the proton. Without external excitation, an electron tends to always fall to the lowest possible energy state. Thus, the slightly excited hydrogen electron after being left in the excited state spontaneously flips its orientation back to the lower energy state. This transition is highly forbidden and it takes roughly 10 million years for the electron to return to its low energy state. However, because of the abundance of the H I atom in the universe, it is also very likely that some collections of it are in this

excited state. The photon released as a result of the spin flip transition is detected as a 21 cm wavelength signal.

1.4 Galaxies and Galaxy Parametrisation

1.4.1 Introduction

Galaxies are gravitationally bound systems that comprise stars, gas, dust, stellar remnants and dark matter. They are the building blocks of the universe. Conselice et al. (2016) estimates at least a total of two trillion galaxies in the visible universe. Despite advances in technology, the sensitivity levels in our current telescopes cannot detect the bulk of the radiation emitted from galaxies in the cosmos. Galaxies are typically classified as either elliptical, spiral, lenticular or irregular according to their morphology (Mo et al., 2010) and our own galaxy (the Milky Way) is one of the many spirals in the universe.

Spiral galaxies are characterised by a rotating disk of stars and gas. By measuring the speed at which the stars and gas move, we can calculate the amount of mass necessary to hold them in circular orbits, in essence measuring the mass of the galaxy. From Kepler's second law, it is expected that the rotation velocities decrease with distance from the centre. Instead, the rotation curves of disk galaxies remain flat as the distance of the radius increases from the centre as seen in Fig. 1.9 below. According to the standard cosmological theory, the most massive constituent of a galaxy does not interact electromagnetically. This unknown material that drives the motion of the gas, dust, clouds and stellar content is detected only through its gravitational influence. Therefore, rotation curves present indirect evidence of this unknown material called dark matter (e.g. Persic et al., 1996). This unseen material neither emits nor absorbs any radiation and accounts for about 90% of the mass of a galaxy; till date, astronomers are unable to provide an explanation of this material and much of what is known about Dark Matter is what it is not.

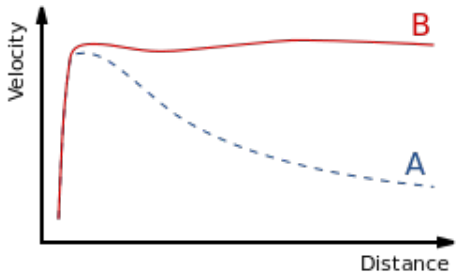


FIGURE 1.9: Rotation curve of a typical spiral galaxy: predicted (A) and observed (B). Dark matter can explain the 'flat' appearance of the velocity curve out to a large radius.
Image Credit: Public domain.

1.4.2 H I in Galaxies

After the Big Bang, most of the matter in the early universe was ionised hydrogen and helium. Gas cooling resulted in the hydrogen becoming neutral leading in the "Dark Ages", in which no light was emitted. After some time, astronomical sources began forming and started ionising the universe. This period is called the epoch of reionization (EoR). The luminous sources which began to form during the period of reionization include galaxies, stars, quasars and clusters of galaxies. Right after EoR, the early luminous sources then began to emit radiation at different wavelengths which we measure with our instruments. Information about these early sources is, however, buried in the H I signal that was produced during the EoR. This 21 cm signal is very weak and thus can only be detected using specially built instruments such as Precision Array for Probing the Epoch of Reionization (PAPER) (Parsons et al., 2010). This is different for local galaxies.

Disk galaxies are known to contain more H I than ellipticals. Disk galaxies contain a thin rotating disk of gas and stars with spiral arms. The spiral arms are better seen when the disk is face on but an edge-on disk reveals the central bulge and the edges. As seen in Fig. 1.10, the H I disk in spiral galaxies is typically much more extended than the bright optical disks (Boomsma et al., 2008).

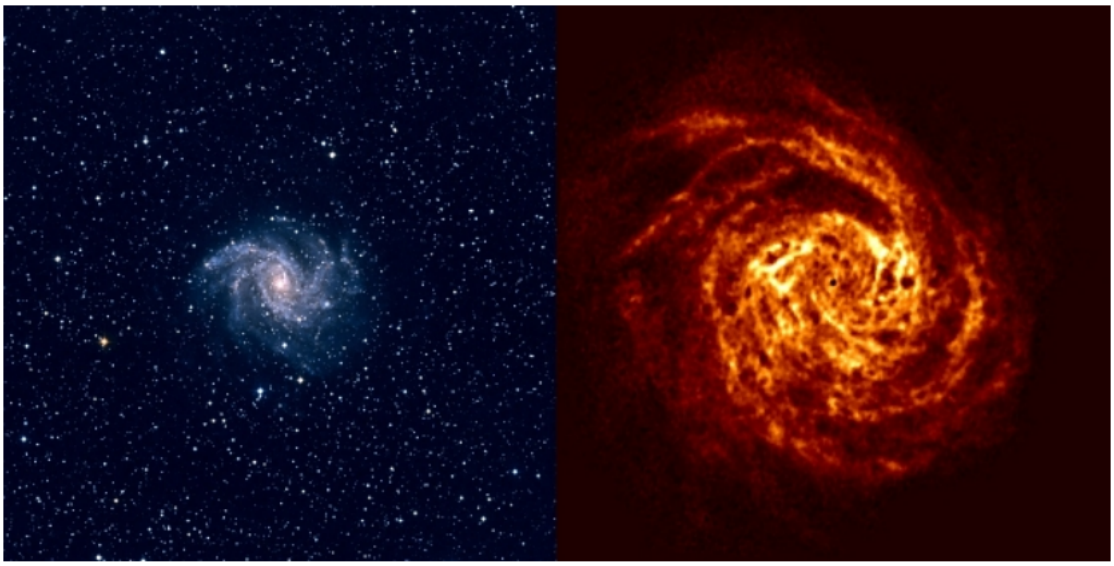


FIGURE 1.10: Comparison between the optical image (left) and the total H I image (right) for the spiral galaxy NGC6946 from Boomsma et al. (2008). The images are of the same scale.

Hydrogen is the most abundant chemical element in the universe. While most of it is ionised, the typical H I content of a galaxy varies by 4 orders of magnitude between $10^7\,M_\odot$ and $10^{11}\,M_\odot$ (Zwaan et al., 2005). Since neutral hydrogen is abundant in galaxies, we can observe it through the 21 cm signal despite the fact that it is a forbidden transition. This makes H I useful in tracing the distribution of galaxies in the universe and mapping out its structure.

1.4.3 Galaxy Parametrisation

Two of the apparent observable features that describe a galaxy are its brightness and radial size. The brightness denotes the amount of energy emitted in the form of light by an astronomical source. This, however, differs from the luminosity which measures the rate of emission of the energy. The total flux, which relates to the measured brightness of a source, is the integrated value of the flux density per unit area of the source. Flux density is measured in units of $\mathrm{W\,m^{-2}\,Hz^{-1}}$. Since the radio signal emitted from astrophysical sources is very small compared to terrestrial sources, radio astronomers use the Jansky (Jy) to express flux densities with $1\,\mathrm{Jy} = 10^{-26}\,\mathrm{W\,m^{-2}\,Hz^{-1}}$. Under the assumption that the light is emitted in all directions equally, brightness (B) and luminosity (L) are related such that

$$B = \frac{L}{4\pi D^2}, \tag{1.3}$$

where D^2 is the square of the distance from the source. In addition to brightness, measured in $\mathrm{W\,cm^{-2}}$, a source can also be measured in magnitudes. Its unit is mag and is a relative scale as it compares the brightness of one star to another using the other as a reference star. Light from galaxies and extended objects spread out into patches in the sky. Thus, by measuring the mag/arcsec^2, we measure the amount of emitting energy per square arcsecond of the galaxy that we would be observing from a star of a particular magnitude.

Another morphological classification of galaxies, adding to the earlier classification in Sect. 1.4.1, is their ellipticity and orientation. The ellipticity measures the axial ratio of the galaxy projected onto the sky. Extracting parameters, such as brightness, total flux, shape and ellipticity, to describe a galaxy is referred to as Galaxy Parametrisation.

1.4.4 Kinematic Parametrisation of Galaxies

Rotation curves, Tully-Fisher relations - a test for galaxy formation (Tully and Fisher, 1977) - and angular momentum distributions are useful parameters and relations that are critical to understanding the formation and evolution of galaxies. Several approaches have been used to extract these parameters from a galaxy disk. It is imperative that the rotation curves are derived with precision and accuracy from the innermost part all the way to the outermost part of the rotating disk. Previously, simple symmetry operations were employed to derive rotation curves from disk galaxies. Currently, the tilted-ring model (Rogstad et al., 1974) is the foremost approach used to parametrise a disk galaxy. Besides the rotation curve, the disk galaxy can be fitted considering other kinematic parameters such as the systemic velocity, rotation velocity etc. The next section elaborates on the tilted-ring model and the H I higher order analysis.

1.5 H I Higher Order Analysis

1.5.1 Introduction

H I disks in spiral galaxies typically contain kinematic information that enable us to answer questions about the evolution and morphology of a galaxy. However, due to projection effects and the internal structure of galaxy disks, H I disks often show complex features that are difficult to parametrise and extract kinematic information through mere visual inspection of data cubes. Therefore, methods have been developed to help in easily extracting these parameters to understand how galaxies form and evolve.

H I disks are known to typically have warps and/or asymmetries (García-Ruiz et al., 2002). Warps basically reveal that outer regions of the H I distribution show stronger irregularities as opposed to what is seen in the inner regions. By using simple conic section symmetry operations, one can parametrise the disk of a galaxy; this would be possible without taking into account these warps. Rubin and Ford (1970) as well as Bosma (1978a) corroborate this by demonstrating how parameter information such as rotation curves and flatness are retrieved from an H I disk using symmetry operations. To explore more, advanced methods have been developed to extract more parameters that describe the velocity structure and the distribution of neutral hydrogen in H I disks.

Proposed by Rogstad et al. (1974), the tilted-ring model (TRM) is a primary tool for H I higher order analysis. This model relies on the premise that the disk of a galaxy can be represented by a series of concentric rings each holding their own orientation and kinematic parameters (see Fig. 1.11).

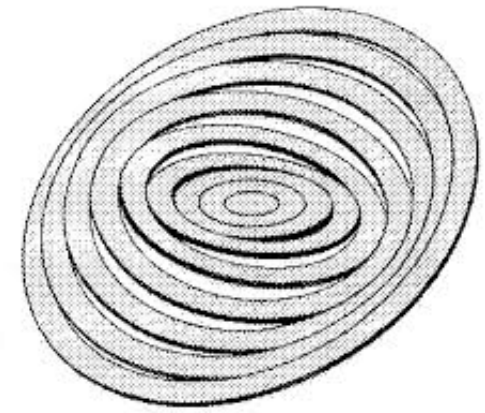


FIGURE 1.11: tilted-ring model of M83 by Rogstad et al. (1974).

Despite the fact that the underlying geometry of the model is similar to earlier approaches to H I disk parametrisation, the TRM is unique in its own way and offers the space to use more parameters to describe the velocity structure and hydrogen distribution. The TRM parametrises disks at different radii using two orientation parameters (inclination and position angle), central position, surface brightness (thickness) and

rotation velocity. It can be fitted to a velocity field (2D fitting) or directly to a data cube (3D fitting).

1.5.2 Velocity Field Fitting

Velocity field fitting is also known as 2D fitting. This is by far the most common approach to fit a tilted-ring model to an H I disk. The velocity of a rotating disk at a sky position of (x, y) can be described with the simple analytical formula (Rogstad et al., 1974; Begeman, 1989)

$$v(x, y) = v_{sys} + v_{rot} \times sin(i) \times cos(\theta), \tag{1.4}$$

where $(x,\ y)$ are the pixel coordinates on the sky, v_{sys}, v_{rot}, i and θ are the systemic velocity of the galaxy, rotational velocity, inclination and angle between the major axis and the (x, y) pixel coordinates respectively. This is related to the position angle (PA) and image coordinates through the relation

$$cos(\theta) = \frac{-(x - XPOS) \times sin(PA) + (y - YPOS) \times cos(PA)}{r}, \tag{1.5}$$

where r is the radial distance from the central coordinates $(XPOS, YPOS)$. This analytical representation can then be fitted to the velocity field in order to retrieve the parameters $XPOS$, $YPOS$, v_{rot}, v_{sys}, i and PA as function of radius r. Figure 1.12 is a Hermite velocity field extracted from NGC 5102.

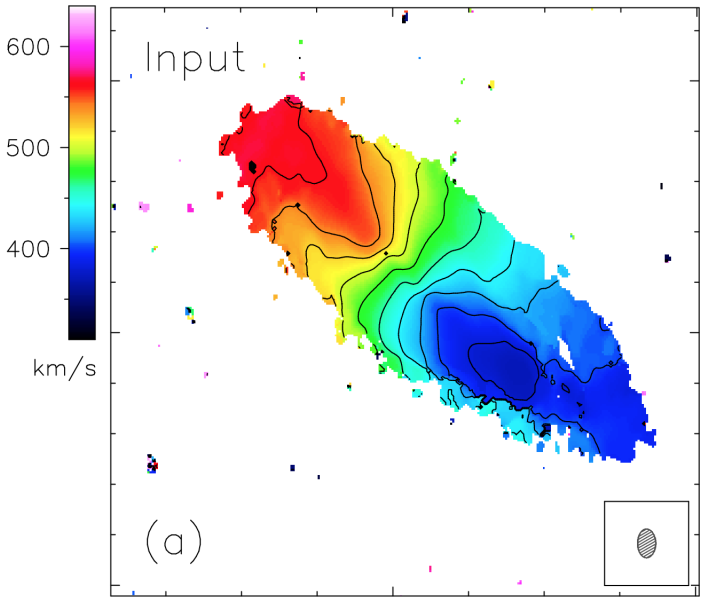


FIGURE 1.12: Hermite Velocity field of the LVHIS galaxy NGC 5102 by Oh et al. (2018).

Velocity field fitting offers the option to use simple fitting techniques. Single Gaussian Fit, Intensity Average Velocity, Peak Velocity, Gaussian-Hermite moments,

among others, are some of the different fitting techniques. The fitting technique used depends on the quality of data and the type of observation. Using the appropriate technique, one can extract the tilted-ring parameters at low computational expense. This justifies why it has been adopted as the standard tilted-ring modelling tool for H I kinematics analysis, especially for well-resolved galaxies with inclinations lying on either side of the halfway point, i.e. 60° inclination. The current analytical representation features systemic and rotation velocities but one can fit more parameters by adding features such as radial velocity or streaming motions to the representation (albeit one would then speak of an extended tilted-ring model). Despite the argued leverage of speed and stability inherent in the velocity field fitting, the approach has several problems.

Creating a velocity field involves choosing one velocity from a velocity profile. The data usually retrieved from interferometric observations have a 3D shape consisting of two spatial axes and one spectroscopic or velocity axis. Since the fit is performed on the velocity field, the data have to be compressed to 2D in order to use this method; thus, losing information. One such key information which is lost using this approach is the surface density distribution which can only be retrieved from an elliptical fit of an integrated moment map. Since the velocity is the only reference attribute used in 2D fitting, the fit to the TRM is subject to retain any errors which may be present in the velocity field, hence, compromising the fit quality. As mentioned in the previous section, warps typically occur in H I disks. These warps can cross the line of nodes. Displaying the line profile of warped galaxies where the warps cross the line of nodes reveals peak intensities which are misrepresented as singular values in the velocity field. The resultant is an erroneous fitting to the model.

From Eq. 1.4, it is easy to see that the velocity field carries very little information about the rotation curve around the minor axis. Therefore, one would usually exclude a given angle around the minor axis from the fit. The angle that is excluded is usually in the range of 10° to 30°. In addition to this, there are limits to the inclination angle beyond which velocity field fitting is not reliable. There is no standard or set inclination angle range within which the velocity field fitting works best. However, it is a usual convention to not apply the velocity field fit to a galaxy where the angle of inclination i lies below 45° and above 75° (Begeman, 1989), outside of which the goodness of fit using chi-square minimisation fails. The reason for this upper limit is mainly because projection effects become pronounced at high inclinations and hamper reliable retrievals of rotation curves using velocity fields (Sofue and Rubin, 2001). However, Oh et al. (2018) demonstrate that well-resolved galaxies with inclination lying between 20° and 70° can also be fitted using 2DBAT. Added to the above is the problem of beam smearing and projection effects which occur purely from errors in the observing instrument (Kamphuis et al., 2015). Beam smearing, according to Bosma (1978b), produces biases when the data has less than seven resolution elements in the Holmberg radius.

Existing software which use velocity field fitting includes the GIPSY routine

known as ROTCUR (Begeman, 1987), KINEMETRY (Krajnović et al., 2006), VELFIT (Spekkens and Sellwood, 2007) and 2DBAT (Oh et al., 2018).

1.5.3 Direct Fitting

Direct fitting outrightly accounts for most of the shortfalls encountered in velocity field fitting. Since no compression is done using this tilted-ring approach, the information in the data cube is preserved. Beam smearing resulting from errors in the observing instrument and projection effects are also reduced in direct fitting. Direct fitting is also referred to as 3D fitting.

In 3D fitting, a TRM is constructed from a cloud of point sources and used to parametrise a model data cube generating a mock observation. The parametrised disk model is then convolved to obtain the resolution of the observed data cube. This is done iteratively and ends when the fit quality, using chi-square minimisation, is good enough. Since each process requires the generation of a data cube and convolution, 3D fitting generally is computationally expensive and has poor throughput compared to 2D fitting. According to Józsa et al. (2007), an additional downside to the 3D fitting is its inability to account for inhomogeneities in the gas distribution. Despite these and the fact that the fitting becomes complex, the process creates the avenue to construct models that cover any form of complexity facilitating the search for extraplanar gas and explains its accompanied kinematics (Gentile et al., 2013; Kamphuis et al., 2013; de Blok et al., 2014). Figure 1.13 represents a galaxy model projected onto a data cube to be compared with the observed galaxy.

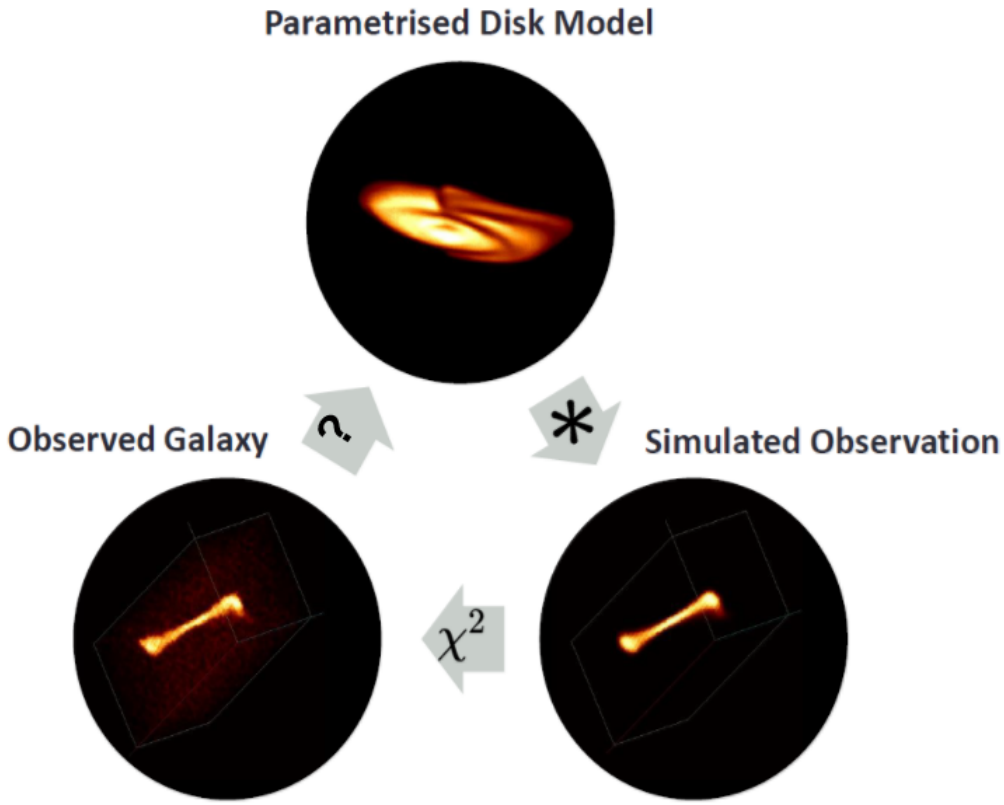


FIGURE 1.13: Direct modelling of disk galaxy

With respect to 3D fitting, the publicly available software called **Ti**lted **Ri**ng **Fi**tting **C**ode (TIRIFIC) offers this possibility (Józsa et al., 2007). Aside all the advantages that direct fitting has over velocity field fitting, TIRIFIC enables the user to fit more parameters. Visualisation tools like KVIS (Gooch, 1996) can then be used to compare models created from the observed data cube. In order to fit the model, TIRIFIC uses chi-square minimisation with emphasis on the golden section method. Other available software includes 3D-BAROLO (Di Teodoro and Fraternali, 2015) and GALACTUS (Peters et al., 2017).

1.5.4 Automated Kinematic Modelling

The TRM has significantly helped in providing a 3D tool to parametrise the most abundant kinematic tracer, H I. However, whether it being 2D or 3D, the parametrisation of the H I observations is still a manual process due to the fact that it requires a guided iterative process by varying the parameters along the fitting process. There exists already hundreds of galaxies from surveys such as the Westerbork H I Survey of Irregular and Spiral galaxies (WHISP; van der Hulst, 2002), the Local Volume H I Survey (LVHIS; Koribalski et al., 2018), DiskMass (Bershady et al., 2010), ATLAS 3D (Serra et al., 2012) and the Bluedisks Project (Wang et al., 2013). To interactively parametrise these galaxies will be very laborious especially considering that the Square Kilometre Array (SKA) South Africa (SA) MeerKAT projects would produce data which exceeds the global internet traffic by ten times. In view of this, efforts have

been made to automate the parametrisation in both 2D and 3D. Despite the advantage of speed offered by automated modelling, there are instances where interactive fitting is more suitable. A typical case is described in Sect. 1.5.3 where the science goal is the search for extra-planar gas. Another such case is when modelling cylindrically symmetric models or very faint emission for which automated modelling simply fails. Examples of such available automated kinematic modelling software include Fully Automated TiRiFiC (FAT; Kamphuis et al., 2015) and 2D Bayesian Automated Tilted-ring fitter (2DBAT; Oh et al., 2018). FAT fits tilted-ring models to H I data cubes of well-resolved galaxies; it is an IDL/GDL wrapper around TiRiFiC. A practical application of FAT is found in Wang et al. (2017) which investigates the star formation properties of the Local Volume H I Survey where tilted-ring models and rotation curves were derived. 2DBAT, on the other hand, implements Bayesian fits of 2D tilted-ring models in order to derive the rotation curves of the galaxy.

1.6 The TIREDIT Code

A TiRiFiC default file editor, developed by George Heald, provides a graphical user interface which allows tilted-ring parameters to be modified in an interactive manner. TIREDIT[4] is a python script which uses Tkinter, a python wrapper around the Tk framework, in the GUI implementation.

Written in 2011, TIREDIT opens in a simple window and provides the following functionalities:

- Select a TiRiFiC `.def` file name.
- Read the values from the file.
- Plot the values of tilted-ring parameters.
- Change the values of multiple parameters using the mouse.
- Write changes to an output `.def` file.

TIREDIT is currently no longer being supported but the code has TODO comments which would have been useful feature improvements. Also, the program hangs when the chosen tilted-ring parameter has less points than specified in the number of rings: a minor bug but worth noting. Aside other desirable features such as linking to a text editor, history in viewgraph, and many others, the GUI framework in which TIREDIT is developed currently does not have an active development community. These make the current version of TIREDIT unsuitable for use with large complex disc models as well as for further development. This warranted the development of a new graphical interface.

[4] http://gigjozsa.github.io/tirific/tiredit.py

1.7 Thesis Motivation

MeerKAT and the SKA will have sensitivities and data rates far exceeding any existing observing instrument which will in turn enable astronomers to probe the universe with acute precision.

Following the high data rates to be produced from these telescopes, automated pipelines have been developed to process the data which will also contain lots of detections with a wealth of kinematic and morphological information buried in these detections. Despite this, automated kinematic modelling does not converge quickly enough when using complex models. The reasons for the slow convergence can be attributed to the processing power of processors, but more specifically to the mathematics behind the fitting algorithm being used. In the case of detections of complex sources, manual interventions will still be required to perform in-depth analysis of the data since there are too many possibilities (e.g. bars, arms, halos, radial motions, shifting centres, flares etc.) to consider. The already existing TiRiFiC would be a useful tool to parametrise the neutral hydrogen kinematic tracer. One issue however, arising when using these tools has to do with its ease of use. Learning how to use the tool and using it should not be what takes most of the time.

There are more than **twenty** parameters to be fitted in each iteration seeking the goodness of fit when using TiRiFiC. Having to edit the parameters in a parameter file for each iteration becomes quite uncomfortable. This work presents a graphical user interface **Ti**lted **Ri**ng **Fi**tting **G**raphical User Interface (TiRiFiG) which allows users of TiRiFiC to perform the fitting process in a more interactive manner.

1.8 Thesis Outline

The remaining two chapters with their contents are summarised as follows:

- **Chapter 2: TiRiFiG Implementation**
  TiRiFiC's layout is given here and the problem being solved is stated. The key software design objectives as well as functional requirements for TiRiFiG are elaborated with implementation details. Five design goals were specified, but the thesis details implementation for three which were realised and the remaining two of which one was partially achieved are earmarked for future work.

- **Chapter 3: Summary and Future Outlook**
  A synopsis of the work presented here is given and final remarks as well as future prospects are discussed.

Chapter 2

TiRiFiG: A Graphical 3D Kinematic Modelling Tool

"*There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.*" ~C. A. R. Hoare

2.1 TiRiFiC Layout and Old Workflow

2.1.1 Introduction

Using the tilted-ring model (TRM), TiRiFiC enables astronomers to model spectroscopic data cubes of galactic disks. The underlying concept used for the fit in TiRiFiC has been tested many times and is proven to produce accurate results. Since the software uses the direct fit method of the TRM for galaxy parametrisation, the discussed advantages of direct fitting over velocity field fitting apply. TiRiFiC creates model data cubes from tilted-ring parametrisations of a galaxy and adjusts them until the fit quality of the data cubes is satisfied given that a local minimum is reached. The goodness of fit can also be justified if there is no variation in the data points in successive fits or by visually inspecting the produced model data cubes against the observed galaxy. The fit quality is estimated using the chi-square test which is then automatically minimised using an optimisation algorithm. For more complex models, TiRiFiC is used to produce model data cubes whose fit quality is estimated by complementing chi-square minimisation with visual inspection.

2.1.2 Old Workflow

TiRiFiC runs as a standalone command-line routine in a terminal. The input data cube, number of rings, tilted-ring parameters, fit constraints and other options are specified by the user in the TiRiFiC parameter file or alternatively, via the command-line. After the tilted-ring parameters are defined, a cloud of point sources of equal flux is constructed and the point sources are gridded to a model data cube. Consecutively, the model data cube gets convolved with a Gaussian beam in order to achieve

the same resolution as that of the observation. At this point, the model data cube gets compared to the observation and the fit quality is evaluated using the chi-square. In cases where the model is complex, the user may also compare data cubes with a visualisation tool to assess the fit quality. This process is repeated until the fit quality cannot be further improved. A comprehensive description of TiRiFiC including examples and exercises can be found on its website[1].

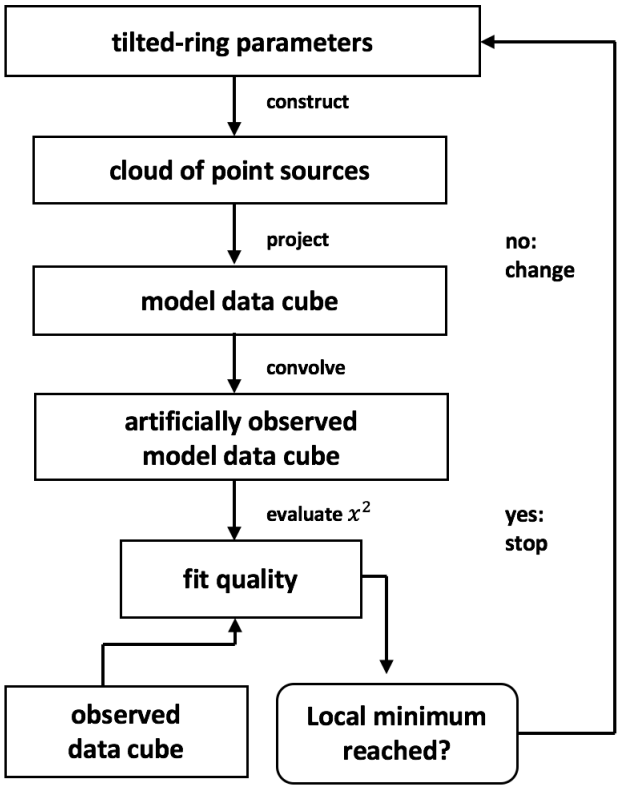


FIGURE 2.1: Model construction and data cube fitting using TiRiFiC.

The flow chart in Fig. 2.1 summarises the typical steps taken when fitting a TiRiFiC model to data cubes. In practice, the evaluation of the fit quality is often made via visual inspection and the parameters are changed manually, but all the other steps are intrinsic to TiRiFiC. Figure 2.2 is an overview of the general workflow when using TiRiFiC to create a complex model of a galaxy requiring manual intervention. Each iteration entails the use of at least three different programs (namely a text editor, TiRiFiC, visualisation software). Obviously, this can become very cumbersome, particularly when handling complex models by manually editing the parameter file which is uncomfortable and confusing.

[1] http://gigjozsa.github.io/tirific/index.html

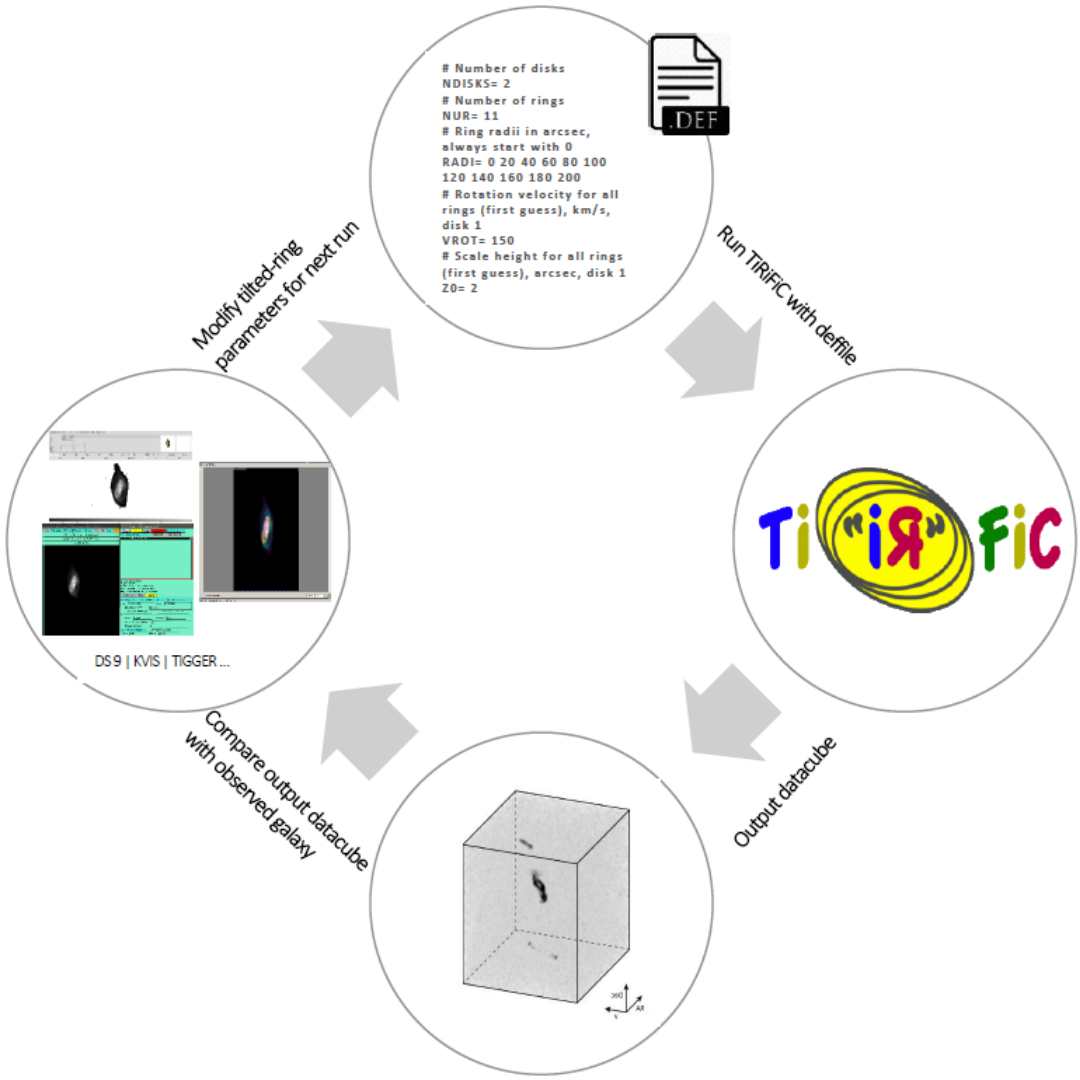


FIGURE 2.2: TiRiFiC workflow.

2.1.3 Problem Statement

One distinguishing feature of TiRiFiC is that it offers the users the avenue to model complex features. There are more than twenty parameters per radial node that can be used to construct a model. Additionally, several disks can be combined into a single model. All these parameters are handled in the `.def` file, which is a simple text file where the parameters are defined one by one with their respective values. Editing the text file, especially when the model is complex (for example, when including haloes, multiple disks, spiral arms and other complex structures), is complicated without a visual aid. Currently, much time and labour is being invested in adjusting the tilted-ring parameters for TiRiFiC.

To ease the editing of the input parameters, a parameter file editor called TirEdit programmed and made available by George Heald helps to visually interact with the tilted-ring parameters in a viewgraph before TiRiFiC is run. TirEdit, however, is limited in many ways and does not provide much flexibility when editing the parameters or interacting with the parameters in the file (see above). In order to mitigate

this cumbersome file editing process of extended models, I have, in collaboration with my supervisors, developed a graphical user interface (GUI) for TiRiFiC: TiRiFiG. TiRiFiG is written as a Python code which utilises the Qt toolkit. The following sections contain the descriptions of the design objectives, implementation and illustration of how TiRiFiG is used.

2.2 Design Objectives and Language Selection

Before I go into the specifics of the design of TiRiFiG, I summarise the software engineering concepts that have been used in this work.

2.2.1 Software Engineering Concepts

Software Engineering principles are a set of rules and procedures which ensure the production of efficient and reliable software. Software development has been classified into distinct cycles which are collectively known as the Software Development Life Cycle (SDLC). The SDLC defines a series of structured software engineering stages which are used to design and build software products. The SDLC stages can be summarised as follows:

- Requirement Analysis: The user, system and functional requirements of the system are collected.
- Design: Logical designs including Unified Modelling Language (UML) diagrams and pseudo codes are produced.
- Development: The logical design is implemented in code and the software is produced.
- Testing and Integration: All the different pieces (functions, classes and modules) of the software are tested individually for errors and also tested as a black box to verify interoperability of the pieces. Several test methods exist but they are all classified as either white-box or black-box testing. Most importantly, tests enable users and developers alike to ascertain the completeness of goals and design objectives before the software is deployed.
- Maintenance: Bug fixes and feature enhancements characterise this phase. Software not maintained any longer is decommissioned and the life cycle ends at that point.

2.2.2 Unified Modelling Language (UML)

UML is an approved ISO[2] standard graphical language for producing software blueprints. Although it is popularly used in software systems, UML is also used to model non-software systems such as the process flow in a manufacturing unit. As a language,

[2]International Organization for Standardization

it provides the vocabulary and rules for producing a conceptual representation of a system and serves as a bridge between requirement analysis and development (coding) of the SDLC. This enables developers to visualise, specify, construct and document components of a software (Booch et al., 2005). It is easy to create any model once a good grasp of the concept has been understood. I now give a brief description of the basic building blocks.

2.2.3 UML Basic Building Blocks

There are three distinct building blocks of UML, viz.: things, relationships and diagrams. I now briefly describe relationships and diagrams.

Relationships

Relationships define the various ways in which elements can be connected to each other. UML relationships can generally be put into four different categories. I describe the relationships which are used in the TiRiFiG implementation in the discussion of the class diagrams in the section below.

Diagrams

UML diagrams are graphical representations of objects joined by a relationship in a sequence. It includes collaboration diagrams, sequence diagrams, object diagrams, class diagrams, activity diagrams, use case diagrams, state diagrams, component diagrams and deployment diagrams. Class diagrams and activity diagrams are discussed in detail since they are used in TiRiFiG implementation.

2.2.4 Class Diagrams

Class diagrams are common in systems developed under the object-oriented programming paradigm. They present a static view of the system by showing the different classes and the relationship(s) between them. A class diagram takes the shape of a rectangle with three compartments each representing class name, attributes and functions (methods). Attached to the functions and attributes are visibility symbols, `(+, -, #)` that tell whether an attribute or a function is public, private or protected. This attribute/function scope specification (visibility) differs (only in naming) as one moves across different programming languages. Python, for instance, uses the keywords *local* and *global* to indicate object visibility. The diagram also illustrates relationships between classes. The relationships that can exist between classes include:

- Inheritance: A class (referred to as subclass) can inherit attributes and functions (methods) from another class (called superclass in this context). This generalisation relationship is shown as an open arrow on the class diagram with the arrow head pointing to the superclass.

- Aggregation: A type of association relationship that specifies a whole and its parts. The two classes are peers (conceptually at the same level) and equally important. As such, this relationship is deeply semantic and any part can exist outside of the whole or may be shared by several wholes. The "whole/parts" relationship is only used to depict a "has a" property. It is represented with a line having an open diamond on one end pointing to the class that represents the whole.

- Composition: It is a type of aggregation but differs in the sense that there is a link between the lifetime of the whole (composite) and its parts. The composition aggregation establishes a strong ownership where the "part" object cannot exist when the "whole" object dies. One part belongs to only one whole and the composite manages the creation and destruction of its parts. Figure 2.3 below illustrates an example of a composition. It is represented with a line with a closed diamond on one end pointing to the parent class.

- Association: A plane association means that the classes are connected to each other and it is represented with a line with a word describing how the classes are connected written on it.

Multiplicities help to set the boundaries for whatever relationship exists between objects. In this case, they determine how many instances of a class can exist for zero to many instances of another class and vice versa. They are written as numbers at both ends of the relationship connecting the classes. You can show multiplicity of exactly one (1), zero to one (0..1), many (0..*), one or more (1..*) or determine an exact number (for instance, 2). This is important in our case because the classes used in implementing TiRiFiG are connected via an association relationship.

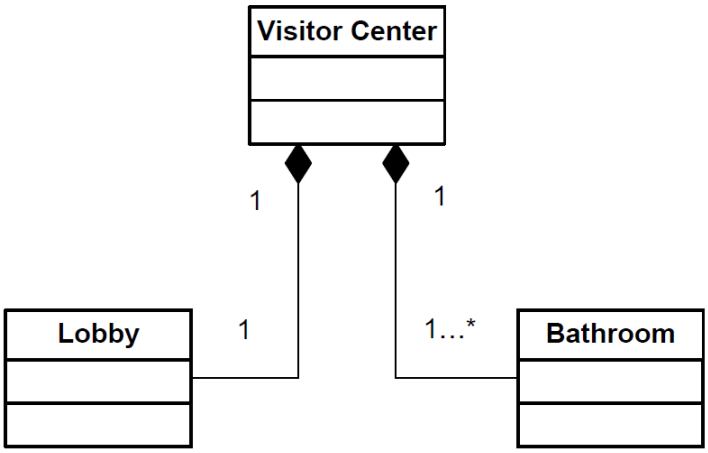


FIGURE 2.3: An example class diagram.

The example class diagram shown in Fig. 2.3 is an abstraction of a visitor centre. The centre has a lobby and a bathroom connected to it in a composite relationship;

thus, the lobby and bathroom will not exist if the centre is destroyed. For the multiplicities represented here, *VisitorCenter* is related to *Lobby* such that one visitor centre can have only one lobby and a specific lobby can only be found in one visitor centre. On the other hand, *VisitorCenter* is related to *Bathroom* such that one visitor centre can have one or more bathrooms but that each bathroom can only be found in one visitor centre.

2.2.5 Activity Diagrams

Activity diagrams are used to model a workflow of a system. They are essentially a flowchart describing the flow of control from one activity to another. The various processes or activities are enclosed in an initial state and a stop state. In the case where decisions have to be made while transitioning from one activity to another, a decision box is used with branches depicting the alternate paths the system will traverse based on the outcome of an evaluated condition. Figure 2.4 below is an example activity diagram showing the various symbols that are used with their annotated meanings.

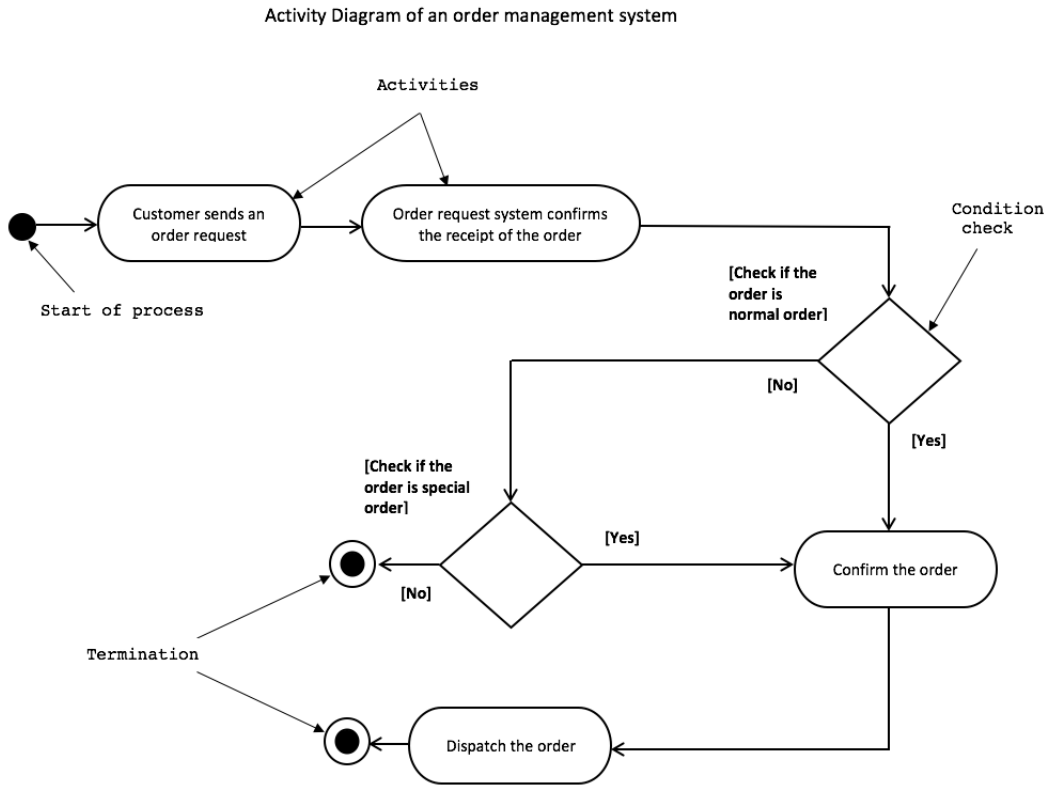


FIGURE 2.4: An example activity diagram.
Image credit: tutorialspoint.

Activity diagrams illustrate the dynamic behaviour of a system. They consist of activity or action states, transitions and objects. Activity states and transitions are discussed in detail below.

- Activity states:
  During the flow of control of a system, several processes occur. Initialising a variable, prompting user for input, making a call to a function, creating or destroying objects, just to mention but a few, are such processes. These different processes are called activity states. Activity states take the form of rectangles with round corners and actions or expressions written in them as shown in Fig. 2.4 above. It is also worthy of note to indicate that all activity diagrams have a start or initial state (represented with a shaded circle) and an end or final state (represented with a bullseye).

- Transitions:
  Action states move from one to another using transitions. The transitions determine the flow of control from one action state to another and in which direction. Once the process in one state is complete, it is exited to be passed on to another state until a stop state is encountered. Branching, forks and joins are also forms of transitions that are used in the activity diagrams. Forks and joins will not be discussed as they are not used in TiRiFiG's activity diagram. Branches, however, are simply a notational convenience that allow us to model decision making scenarios. Transitions are represented with a directed line connecting the states whilst branches are diamonds with different transition paths connected to it.

2.2.6 Design Goals

The prime goal of this research project is to produce an interface that is easy to use, offering a much better way to interact with tilted-ring parameters before TiRiFiC runs. The following design goals (DG) were set to achieve this prime objective:

- DG 1: The interface will accept as input a TiRiFiC `.def` parameter file and produce a viewgraph.
- DG 2: Data points of the plotted parameters in the viewgraph should be movable by click and drag mouse actions.
- DG 3: It should be possible to run TiRiFiC from the interface.
- DG 4: There should be a link to a visualisation tool to view H I data cubes.
- DG 5: The system should be Python based, portable, maintainable and open source.

These design goals will be implemented by different functions from different interrelating classes. Details on the listed DG above follows in the subsections below.

DG 1

Tilted-ring parameters specified in the `.def` file should be read and profiled for plotting with four parameters (**INCL, PA, VROT, SBR**) loaded by default on the first run.

The viewgraph will have an option to add parameters to the viewgraph and remove plotted parameters as well. The reason for choosing these four parameters to be loaded by default is because they are most often radially fitted in the TRM model. The user should also be able to determine from a *preference menu* how many parameters are loaded by default after the parameter file is specified. The default layout of parameters in the viewgraph after the parameters have been read will be **n rows** and **1 column** but there should be the option to alter arrangement of plots in the grid by specifying a preferred layout by entering a row and column configuration from a menu. The interface should also provide the following popular commands: open, save, save as, undo, redo and exit.

DG 2

Each data point in each parameter's viewgraph should respond to left-click, double-click and drag mouse movements. This will allow the user to modify the values using mouse movements other than typing values from a text editor. Nonetheless, there should also be the option to edit values from a text editor and have the new entry synced to values displayed on the viewgraph. The interface should synchronise actions on the viewgraph with the loaded parameter file whilst updating the file's content and also allow for the manual editing of the parameter file with any preferred text editor. During the mouse drag, the points should be allowed to be dragged beyond the current values of the plotting range. As such, the interface should also be able to do a live adjustment of the scale while points are dragged beyond the plotting range.

Additional functionality will also be to perform a group move of data points. The group move will allow two or more points in a parameter's viewgraph to be moved together after selecting the desired data points while holding down the `Ctrl` key. The double-click of data points on the viewgraph should pop up a menu that will prompt the user to provide a desired value for the node in focus. Since the mouse actions modify the values of the data points, a provision should be made for the user to be able to determine the precision points of the new values. By default, the precision points of the new values are fixed to the existing precision which was read from the parameter file. Thus, by providing a visual representation of the parameters that can be immediately edited, the complexity of editing the input file containing the parameters is significantly reduced.

DG 3

After the user has finished modifying the parameters in the viewgraph, it should be possible to call TiRiFiC from within the interface to create a TRM from the tilted-ring parameters and have the model fitting started. The implementation of this functionality would be to either use a progress bar to indicate the progress of TiRiFiC running in percentage terms or to call TiRiFiC to run from an open terminal.

DG 4

The user should be able to view the parametrised data cube and the observed galaxy from a data cube visualiser provided from the interface. This can be done by either using legacy visualisers, such as KVIS or DS9, or writing an embedded visualiser in the interface.

DG 5

PYTHON will be the preferred programming language for this project since it is used by most astronomers, allows for easy code maintenance and runs on all machine architectures. The source code will be publicly available on `GitHub`[3].

2.2.7 TiRiFiG UML Diagrams

Two UML diagrams are used to describe TIRIFIG, a class diagram and an activity diagram.

Class Diagram

A high-level class diagram for TIRIFIG is shown in Fig. 2.5. This displays five classes, their relationships (association and composition) and their accompanying multiplicities. Table A.2 in Appendix A provides a comprehensive (low-level) list of all the classes and their accompanying attributes (with data types) and functions.

The relationships illustrated in Fig. 2.5 are composition and association. The following class pairs have composition relationships: (`MainWindow`, `SMWindow`), (`MainWindow`, `GraphWidget`) and (`MainWindow`, `ParamSpec`). The `MainWindow` object can have zero or many `GraphWidget` objects; thus, an infinite number of viewgraphs can be created, each representing a different tilted-ring parameter after a `.def` file has been opened and loaded. The `MainWindow` object can have zero or one `SMWindow` and/or `ParamSpec` object. (`MainWindow`, `Timerthread`) and (`TimerThread`, `GraphWidget`) class pairs illustrate association relationships with `TimerThread` class triggered by `MainWindow` class which in turn updates the `GraphWidget` class.

Activity Diagram

TIRIFIG's high-level activity diagram shown in Fig. 2.6 below has seven states. The input to the software is a parameter file which contains all the tilted-ring parameters as well as the path to the data cube of the observed galaxy. The tilted-ring parameters are read and indexed to be displayed on a viewgraph. After successful load of the parameter file, the parameters, **VROT, SBR, PA** and **INCL** are displayed on the viewgraph in a `4x1` grid configuration. More graphs can be added by selecting different parameters which have been profiled on load of the text file. The user can shift the data points to desired positions and save changes to the parameter file before starting

[3]https://github.com/gigjozsa/TiRiFiG

TiRiFiC. This flow of control from activity to activity iterates until the branch "fit quality" evaluates to "yes".

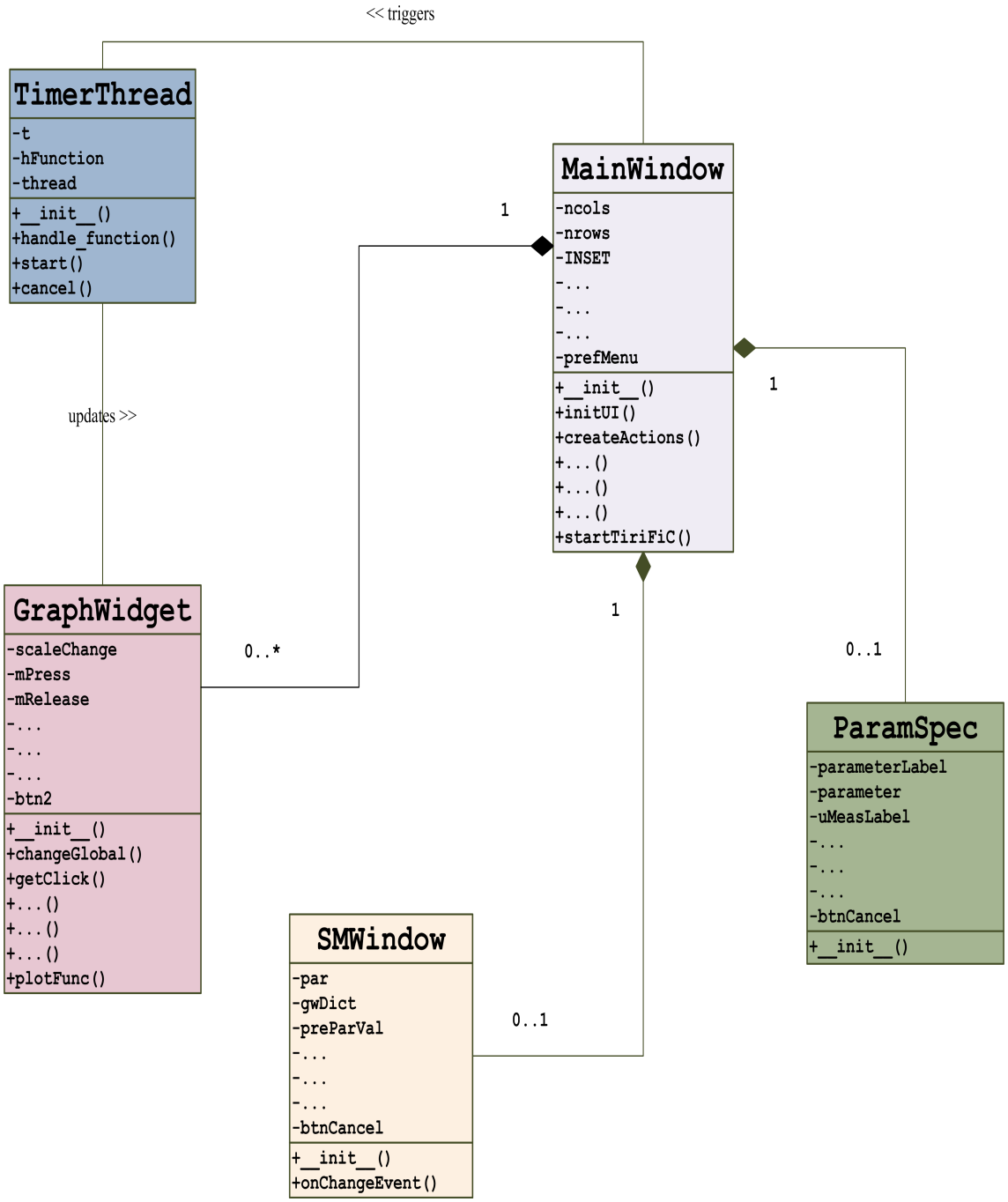


FIGURE 2.5: The class diagram for TiRiFiG.

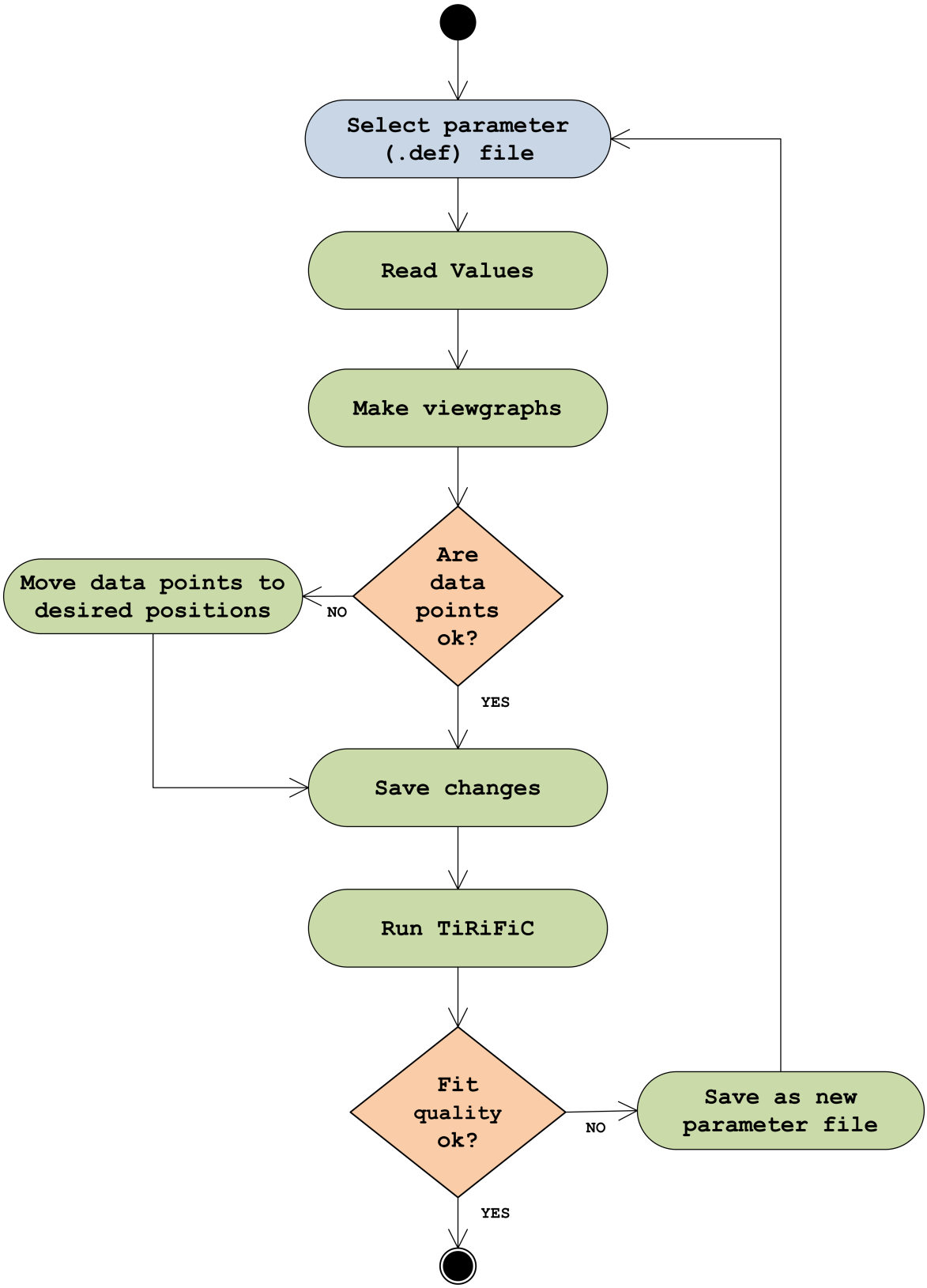


FIGURE 2.6: Activity diagram showing the process flow for TIRIFIG.

2.2.8 Language Selection

The PYTHON programming language is the preferred language for developing programs by astronomers. It harnesses the large user and developer community to improve software and to ensure continued maintenance. PYTHON is a dynamically typed interpreted language and thus much of the information about a written program can only be determined at run time. This makes it perform slower on standard benchmarks compared to statically typed compiled languages. Despite this, in PYTHON it is generally much easier to develop simple, productive, readable code. Also, it has C libraries as part of its own library that make code execution much faster. An example of such a library is the famous `NumPy` PYTHON library. It is hence a natural choice to base TIRIFIG on PYTHON.

A GUI framework provides the structure to support the development of applications that are driven by graphics and icons other than texts and commands. In our search for which GUI to consider, we wanted a framework which provides access to classes that allows for creation of modern widgets, has active community development and adopts a style which makes it fast and flexible to write code. The traditional PYTHON GUI framework, TkInter, was first explored in the development of TIRIFIG's interface. TkInter has the advantages of being lightweight and easy to use. However, it has some limitations which puts PyQt and newer GUI frameworks ahead of it. Some of the limitations include:

- An added layer of complexity, since its widgets are not PYTHON objects. This makes its development non-pythonic (code conventions of the PYTHON community), thus, making it cumbersome to debug errors.
- The inactive development community. This has however been attributed to the fact that it is a fairly stable language compared to new GUI frameworks.
- The lack of support for newer image formats, namely `PNG` and `JPEG`.
- A limited number of core widgets as opposed to the range of widgets available in other frameworks.

In view of the limitations listed, the GUI was developed using PyQt which is a PYTHON wrapper around the Qt framework. PyQt is only one of the frameworks for GUI programming in PYTHON. Table A.1 in Appendix A summarises equally decent GUI toolkits that can be used. So, why PyQt then?

- PyQt combines the successful cross-platform C++ Qt framework with the cross-platform PYTHON programming language.
- PyQt transcends beyond being just a GUI toolkit and comprises a rich set of over 620 classes that cover graphical user interfaces, XML handling, network communication, SQL databases, Web browsing and other technologies available in Qt.

- The Qt classes also make use of a signal and slot mechanism that allows for communication between objects.
- Qt includes a QtDesigner, a graphical user interface, that allows developers to create and add GUI tools easily.
- PyQt automatically generates code for the interface built from the QtDesigner. This feature makes PyQt very suitable for rapid prototyping.
- Python as a programming language already supports the ability to build complex and large applications in a very simplistic manner.

Thus, PyQt offers a programmer the ability to exploit the power of Qt with the simplicity of Python. PyQt builds successfully in Python 2 and Python 3. The `QtGui` module which contains a substantial set of GUI widgets and the `QtCore` module responsible for handling event loops and Qt's signals and slot mechanism are the main modules used in developing TiRiFiG's interface. PyQt is distributed under a dual license: GPL license (versions 2 and 3) or a commercial license. PyQt is available for the major existing operating systems (Windows, Unix/Linux and Mac OS X) making it a suitable selection to achieve our portability design goal.

2.3 Implementation

In our first attempt at implementing the system, we experimented on the rough idea we had of using Tkinter to build a prototype and having a sense of what the interface will look like and how its functions will be implemented. With the prototype in place, the set goals could now be implemented with a clear mind iterating back on the features and functionality that was achieved in the Tkinter example (prototype). This approach of building a system around a prototype is referred to as Rapid Prototype Model; in which a software model built from the early stages is not part of the final deliverable. But instead, it allows users to evaluate the developer's proposal for the design of the eventual product. Interaction and user interface designs are particularly better developed using this approach.

The functions in TiRiFiG can be summarised into the following user interaction (UI) modules: `input module` and `display module`. These exclude other modules which work behind the `UI module`. In Python, modules are simply scripts containing Python code. Such scripts can be imported in other scripts to use their classes, functions and variables. TiRiFiG has been written using one custom module and imports classes from other modules in the standard library and the `PyQt4` module. Modules used in this context represent independent functions from the different classes that work together to achieve the goals as specified in Sect. 2.2.6. Details of the available features and functionalities that are used in TiRiFiG are listed below.

2.3.1 UI Modules

Input Module

The input module comprises five functions from the `MainWindow` class, namely:

```
def getData(self) -> list
        """Loads data from specified .def file in open dialogue box.
        """
def strType(self, var) -> str
        """Determines the data type of a variable.
        """
def numPrecision(self, data) -> int
        """Determines and sets floating point precision.
        """
def getParameter(self, data) -> None
        """Fetches data points of specified parameter.
        """
def openDef(self) -> None
        """Opens data, gets parameters values, sets precision and sets
        scale.
        """
```

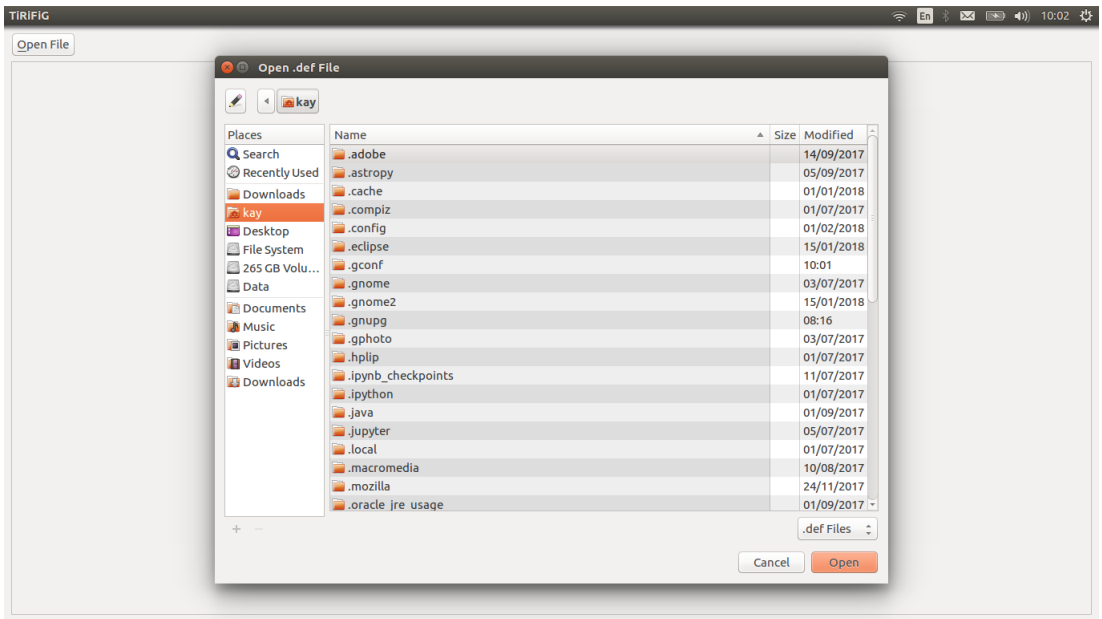


FIGURE 2.7: Input module popping up a dialogue box to prompt user to select parameter file.

The input module opens the gateway for other modules to now become useful in the sense of what they were built for. The only input to this module is a parameter file which contains tilted-ring parameters and fitting constraints. The input module addresses some of the features discussed in **DG 1**. When TiRiFiG is started, the `getData` function is what prompts the user from an open dialogue box to specify the parameter file. This function is invoked by clicking the **open** command on the *file*

menu. Alternatively, the **Open File** button on the TiRiFiG window can be clicked to produce the same result. The file is opened and the parameters are read. At this point, the `getParameter` function is invoked and the read values are formatted and profiled for plotting on a viewgraph. This function makes the code smart enough to parse the various parameters irrespective of the order or format in which parameters are specified. Parameters with values less than the value specified for the number of grid nodes (NUR) are extended as well. However, parameters with values which are not integer or float type will break the code. Both the `strType` and `numPrecision` functions are called within the `getParameter` function and all four functions are called within the `openDef` function. An example interface of the prompt is shown in Fig. 2.7. The dialogue box invoked by the module only shows `.def` extension in the file type dropdown. After the values have been formatted, four graph widget objects, each representing the values for the different tilted-ring parameters - **INCL, PA, VROT** and **SBR** - are created from the `GraphWidget` class. Each graph widget object houses their individual viewgraph.

Display Module

The display module is the centrepiece of the GUI. This module is invoked immediately after the input module starts and finishes successfully. It contains eight functions from the `GraphWidget` class and five functions from the `MainWindow` class. The functions include:

```
def getClick(self, event) -> None
        """The x-data is captured when the left mouse button is
        clicked on the canvas.
        """
def getRelease(self, event) -> None
        """The x-data is captured when the left mouse button is
        released on the canvas. The new data point is added to the
        history and mouse pressed is assigned 'None'.
        """
def getMotion(self, event) -> None
        """The x-data is captured when the left mouse button is
        clicked while moving on the canvas.
        """
def undoKey(self) -> None
        """Deletes the last item in the history list when 'Ctrl+z' is
        pressed and re-draws graph.
        """
def redoKey(self) -> None
        """sends the viewgraph forward in history if an undo
        action(s) has/have been performed.
        """
def showInformation(self) -> None
        """Displays a messagebox that informs user there's no previous
        action to be undone.
```

```
        """
def firstPlot(self) -> None
        """Plots data from file. Produces view graph from historyList.
        """
def plotFunc(self) -> None
        """Plots data from file. Produces view graph from historyList
        or parVals.
        """
def SMobj(self) -> None
        """Instantiates the scale manager window and pops it.
        """
def editParaObj(self) -> None
        """Instantiates the parameter specfication class and connects
        btnOK button to editParamDef function.
        """
def paraObj(self) -> None
        """Instantiates the parameter specfication class and connects
        btnOK button to paramDef function.
        """
def setRowCol(self) -> None
        """Specifies the number of rows and columns in the grid layout.
        """
def openEditor(self) -> None
        """Opens parameter file in preferred text editor.
        """
```

The display module produces a viewgraph of the tilted-ring parameters and allows the users to modify them interactively. Figure 2.8 below is an example of the interface generated when a parameter file is successfully read. There are two viewgraphs (**VROT** and **SBR**) visible and a vertical scroll bar on the right indicating there is more content is below.

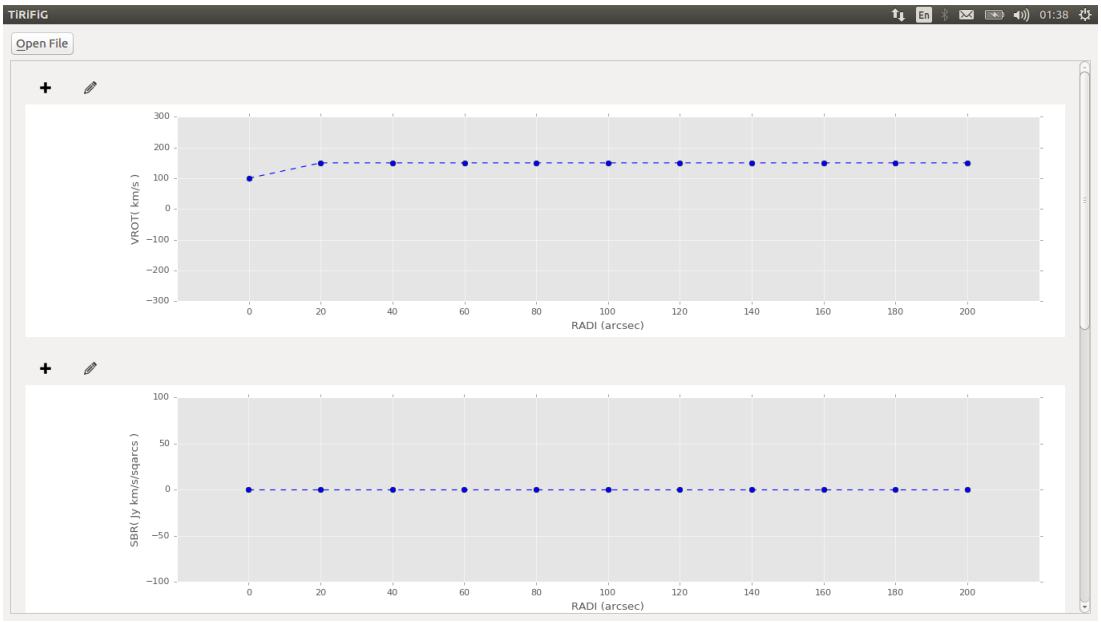


FIGURE 2.8: Viewgraphs displaying VROT and SBR with PA and INCL visible when scrolled down.

The display module addresses the second DG. Any of the data points in the viewgraphs can be dragged to a different value position or set to a value by entering the new value in an input dialogue box after double-clicking a specific data point. The `getClick`, `getRelease` and `getMotion` functions are used to achieve the drag feature and the input dialogue box is obtained from the double-click feature. Since the points can be dragged beyond the current plotting range, the `SMobj` function provides a Scale Manager that gives the user the option to determine the scale behaviour of the viewgraph as the data points are being dragged. It enables the user to set boundaries for the points being plotted for a specified parameter on the *x-axis* and the *y-axis*. The two options used to determine the scale behaviour from the Scale Manager window are *Beyond Viewgraph* and *Free*. The former increases or decreases the scale of the graph only when a data point is being dragged beyond the highest or lowest value on the viewgraph; this is the default option. The latter, on the other hand, adjusts the scale behaviour based on how close or wide apart the highest and lowest points are to each other. The Scale Manager can be accessed on the *preference menu*.

Each viewgraph has two quick access buttons, *plus sign* (left) and *pencil* (right), that allow the user to add more parameters in the viewgraph or change the parameter being plotted in a particular viewgraph. The dialogue box displayed when adding a new parameter has an editable drop down menu of all variables profiled from the input file which the user can select or alternatively type a new variable. The user can also change the layout of how the graphs are arranged in the window to any row and column configuration. This feature is accessible on the *preference menu* by clicking the *Window Specification* command. The new layout is entered as two digits (r,c) separated by a comma where r and c represent row and column numbers respectively. The functions that handle these features are from the `MainWindow` class,

viz: `editParaObj`, `paraObj` and `setRowCol` functions. The `editParaObj` and `paraObj` functions perform the editing and addition of the parameters in the viewgraph whilst the `setRowCol` function is responsible for the arrangement of the graph widgets. Depending on the number of parameters plotted in the current viewgraph, the user can enter any row number and column number combination as long as the product of the two does not fall short of the current number of parameters in the viewgraph. If the user enters a combination whose product is less than the number of parameters currently plotted, an information alert pops up indicating that the combination should at least match the number of parameters. The configuration of the layout in Fig. 2.8 has been changed to a `2x2` grid configuration in Fig. 2.9.

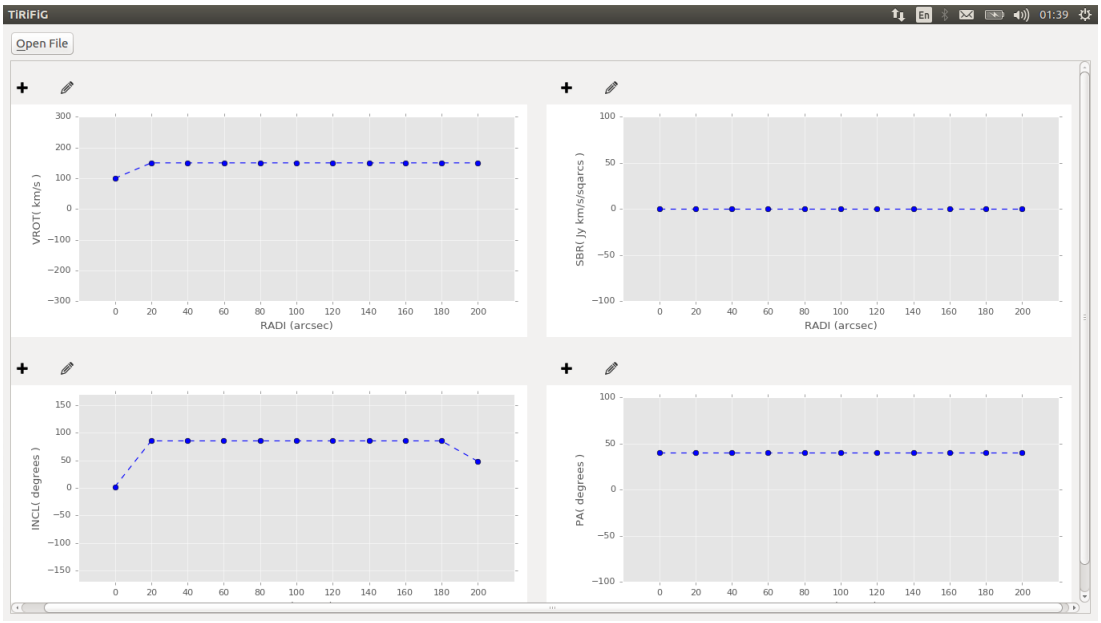


FIGURE 2.9: Viewgraph displaying VROT, SBR, PA and INCL visible in a 2x2 grid configuration.

The display is what provides the user the visual perspective of the tilted-ring parameters before TiRiFiC is called to produce a model data cube and fit. There is also, however, a provision for users to edit the parameters in a text editor. The action on the text editor is synced automatically to the plot of values in the viewgraph. The `openEditor` function is responsible for handling this feature. This function allows the user to open the parameter file being displayed on the viewgraph in a text editor. Actions in the text editor are synced with the data displayed on the viewgraph. Data points which are modified in the text editor are automatically updated in the viewgraph. The sync is handled by the `slotChangeData` function which is called by the `animate` function after it has been invoked by the `openEditor` function. The user's preferred text editor can be specified in the input dialogue box that opens when the *Open Text Editor* command is clicked on the *run menu*. The specific text editor has to be typed as would be called from the terminal. If no text editor is entered, the default text editor is opened.

2.3.2 Other Modules

Run TiRiFiC

This is handled by the `startTiriFiC` function. It uses the system library in PYTHON to make a call to the terminal to start the TIRIFIC software installed on the system. TIRIFIC then runs to produce a model data cube and or also perform the fitting. It can be accessed by clicking the *Start TiriFiC* command on the *run menu*.

Forward and Backward History

All mouse drag actions on the data points are recorded in history. On press of the key combination, `Ctrl + z` or `Ctrl + y`, the `undoKey` and `redoKey` functions allow the user to undo or redo drag events that shift data points on the viewgraph. These functions can also be accessed with the *undo* and *redo* commands on the **file menu**.

Save Feature

The `saveAll` function allows the user to save changes made to a parameter file on the viewgraph to disk. It can be accessed by clicking the *save* command on the *file menu*. Alternatively, the `saveAsAll` function allows the user to create a new parameter file and save to disk. The contents of the input file are read into a list where each index represents a line. As such, the layout of the new output file after a new file is saved is the same as the input file. A dialogue box appears prompting the user to enter the new file name and specify the location. It can be invoked by clicking on the *Save As* command on the *file menu*.

Exit Program

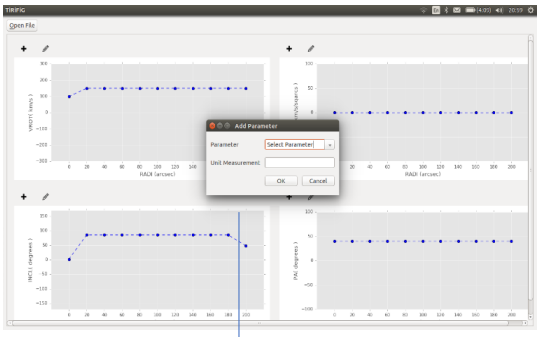
This provides the facility for the user to exit the application. It is handled by the `quitApp` function which is invoked by the *exit* command on the *file menu* or by pressing the key combination `Ctrl + q`.

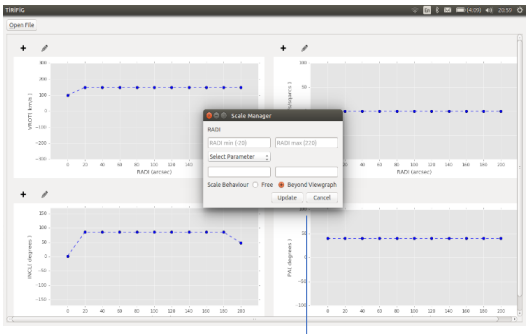
2.3.3 Implementation Summary

The above discusses how the design goals enumerated in Sect. 2.2.6 have been implemented. It is however worthy of note that **DG 4** has not been implemented in the current roll out of this TiRiFiG version. Also, the group move of the data points is not featured in the implementation. Table 2.1 summarises the functions used to implement the UI module and other modules. Figure 2.10 below also shows the other windows available in TiRiFiG.

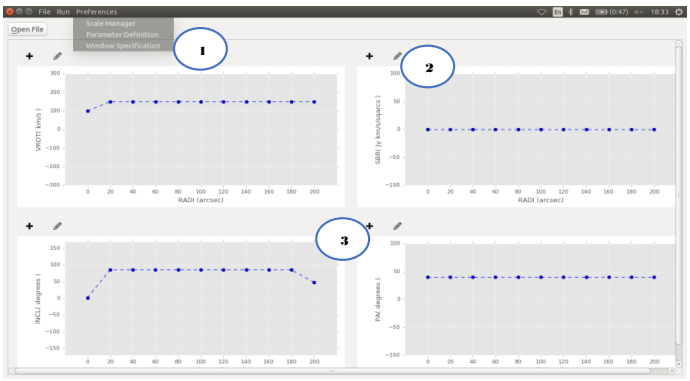
TABLE 2.1: Summary of Functions used in UI module and Other modules

UI Module	**Functions Comprised**
Input Module	def getData def strType def numPrecision def getParameter def openDef
Display Module	def getClick def getRelease def getMotion def undoKey def redoKey def showInformation def firstPlot def plotFunc def SMobj def editParaObj def paraObj def setRowCol def openEditor
Other Modules	**Functions Comprised**
Run TiRiFiC	def startTiriFiC
Forward and Backward History	def undoKey def redoKey
Save Feature	def saveAll def saveAsAll
Exit Program	def quitApp



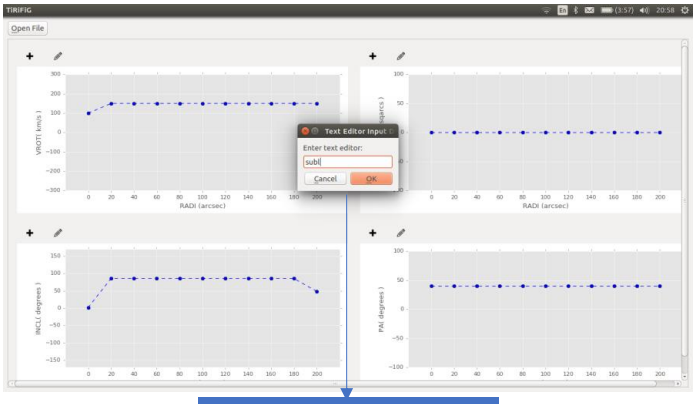


Add parameter dialogue box prompting user to enter new parameter to be plotted



Scale manager window for determining scale behaviour.

1) menu bar with preference menu showing menu options
2) edit currently plotted parameter
3) add parameter to be plotted



Text editor dialogue box

FIGURE 2.10: Other windows in TiRiFiG.

2.4 Testing

As was discussed in Sect. 2.2.1, software testing is an indispensable step in the software development life cycle and several test methods exist to validate implemented software against design specifications. Specific unit tests were not written for the different functions and modules discussed. Instead, the tests performed on different units were done in parallel with development by running each piece of code for any new function implemented while fixing any errors encountered. Specific tests included successful selection of error-free parameter files, successful reading of tilted-ring values from file, plotting the values of the often fitted parameter files on initial file upload, adding new plots in window for other tilted-ring parameters, running TiRiFiC from the interface, and all other implemented functions from the design. The main test objectives were to ensure that:

- The various functions written in the various classes work independently.
- The modules work together as a single unit without breaking.
- The complexity encountered using the traditional TiRiFiC is alleviated in the current user interface design of TiRiFiG.

The parameter files from TiRiFiC's example[4] web page were used to validate the performance and behaviour of both the individual functions/modules and the system as a whole. A test with the parameter file which has the science case considered and demonstrate how TiRiFiG performs is detailed in the next subsections.

2.4.1 Fitting with TiRiFiG

Current Workflow

Figure 2.11 shows a schematic diagram of the key components of the graphical user interface. It includes a data cube visualisation tool (yet to be implemented in code), TiRiFiC and the interactive tool all bundled together with various functionalities available in the menus. Figure 2.2 in Sect. 2.1.2, summarises the old workflow of TiRiFiC and highlights the difficulty in editing the parameter file in a text editor. Despite there being no the visualisation tool currently, the current implementation of TiRiFiG is nonetheless useful since the new workflow plugs in a visual aid to enable the user to interact visually with the tilted-ring parameters.

[4] http://gigjozsa.github.io/tirific/examples.html

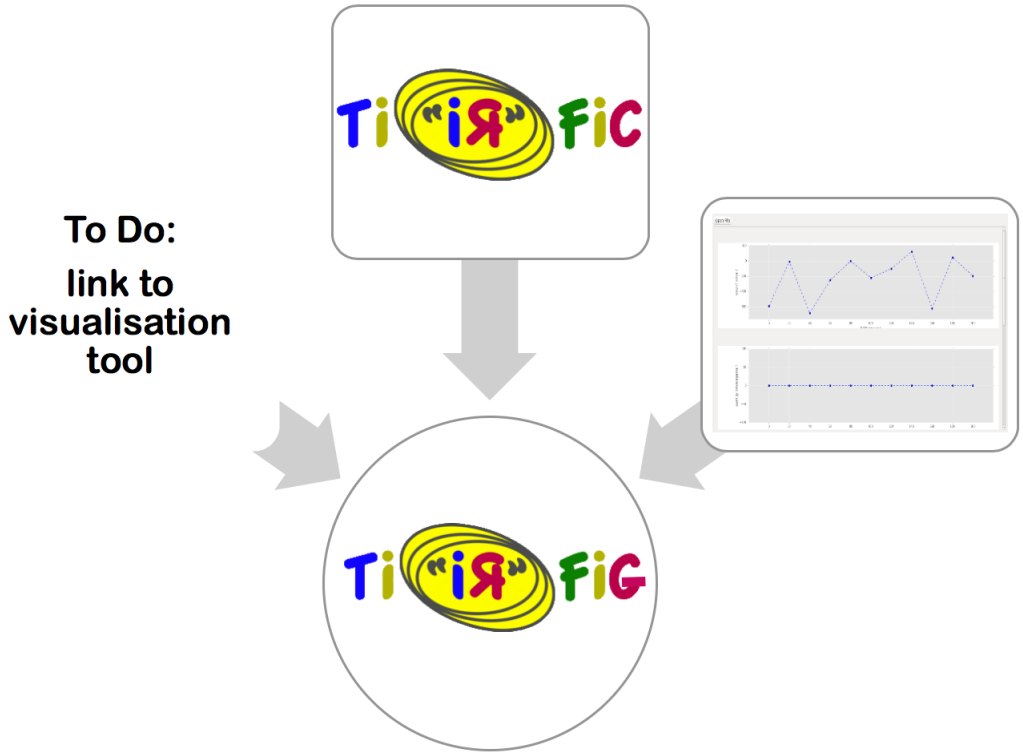


FIGURE 2.11: TiRiFiG Functionality Overview.

Illustration with TiRiFiG

Figure 2.13 below, from Józsa (2007), shows a bump in the rotation curve at 120" radius. Using TiRiFiG and TiRiFiC's extended capabilities (compared to the 2007 version) as well as progressing insights to the H I distribution we modelled the extra-planar gas in NGC 5204 and removed the bump in the rotation curve produced in Józsa (2007). In order to answer the question about the kinematics of the extraplanar gas, we considered the following cases to obtain tilted-ring parametrisations of the spiral galaxy NGC 5204:

- A warp with variable dispersion.
- Radial motion.
- Radial motion varying with height above the plane.
- Tangential motion varying with height above the plane.
- Radial and tangential motions together.

The motivation for the scientific cases was gotten from Schmidt et al. (2014) which is incidentally a reproduction of which analyses the kinematics of the dwarf irregular galaxy UGCA 105. The tilted-ring parameters modelled, as enumerated above, map to the following TiRiFiG parameters: **SDIS, VRAD, DVRA, DVRO** and **DVRA & DVRO**. TiRiFiC was run six times to perform a fit for each of the cases modelled

after adjusting the tilted-ring parameters in the viewgraph provided by TiRiFiG. After TiRiFiC had completed fitting, using the viewgraph proved to be a much more convenient way to adjust the values of the tilted-ring parameters since this had significantly reduced the time spent in modelling.

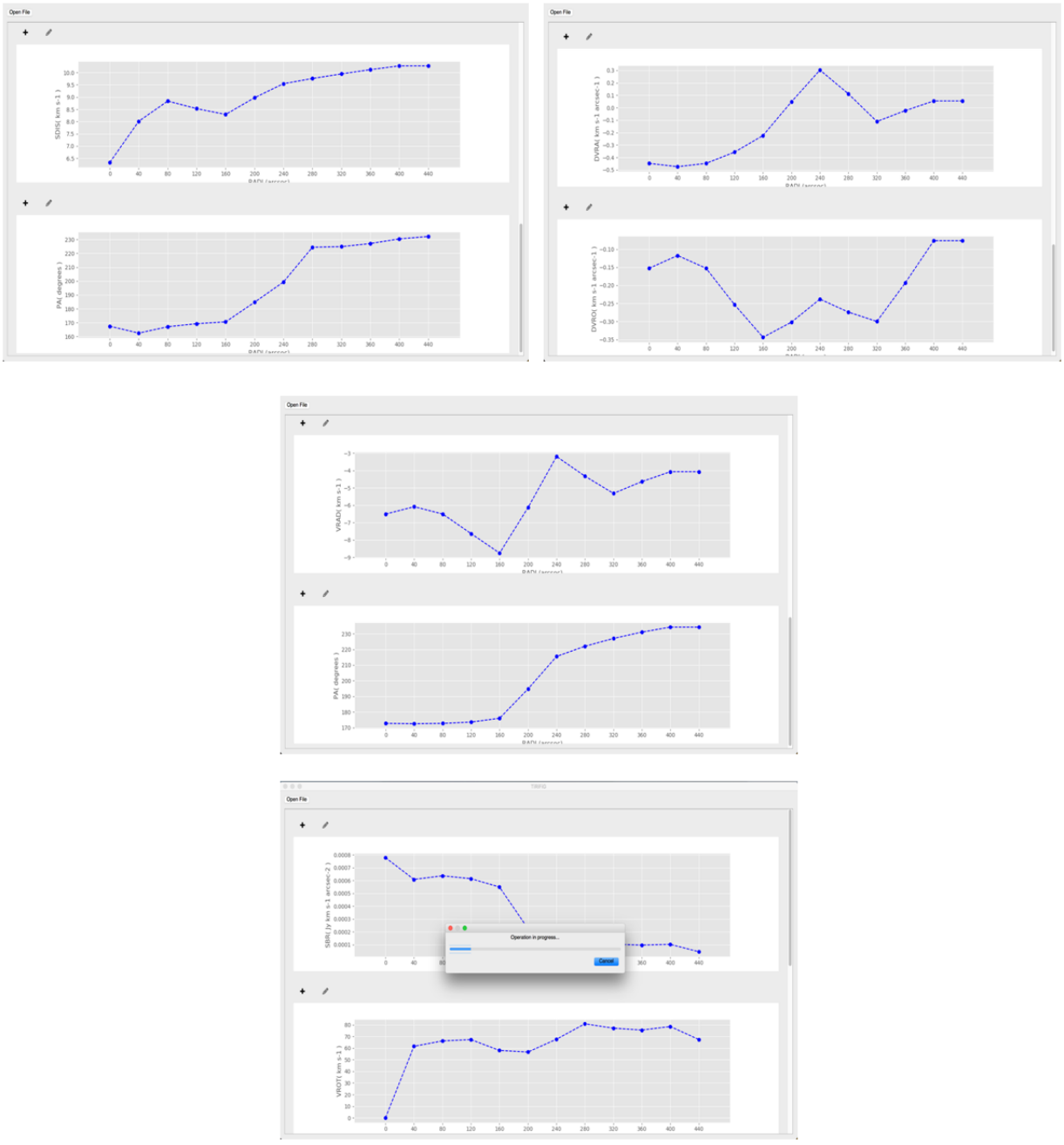


FIGURE 2.12: Modelling with TiRiFiG. Top: SDIS (left) and DVRA, DVRO (right). Centre: VRAD. Bottom: Progress bar dialogue indicating progress of fitting.

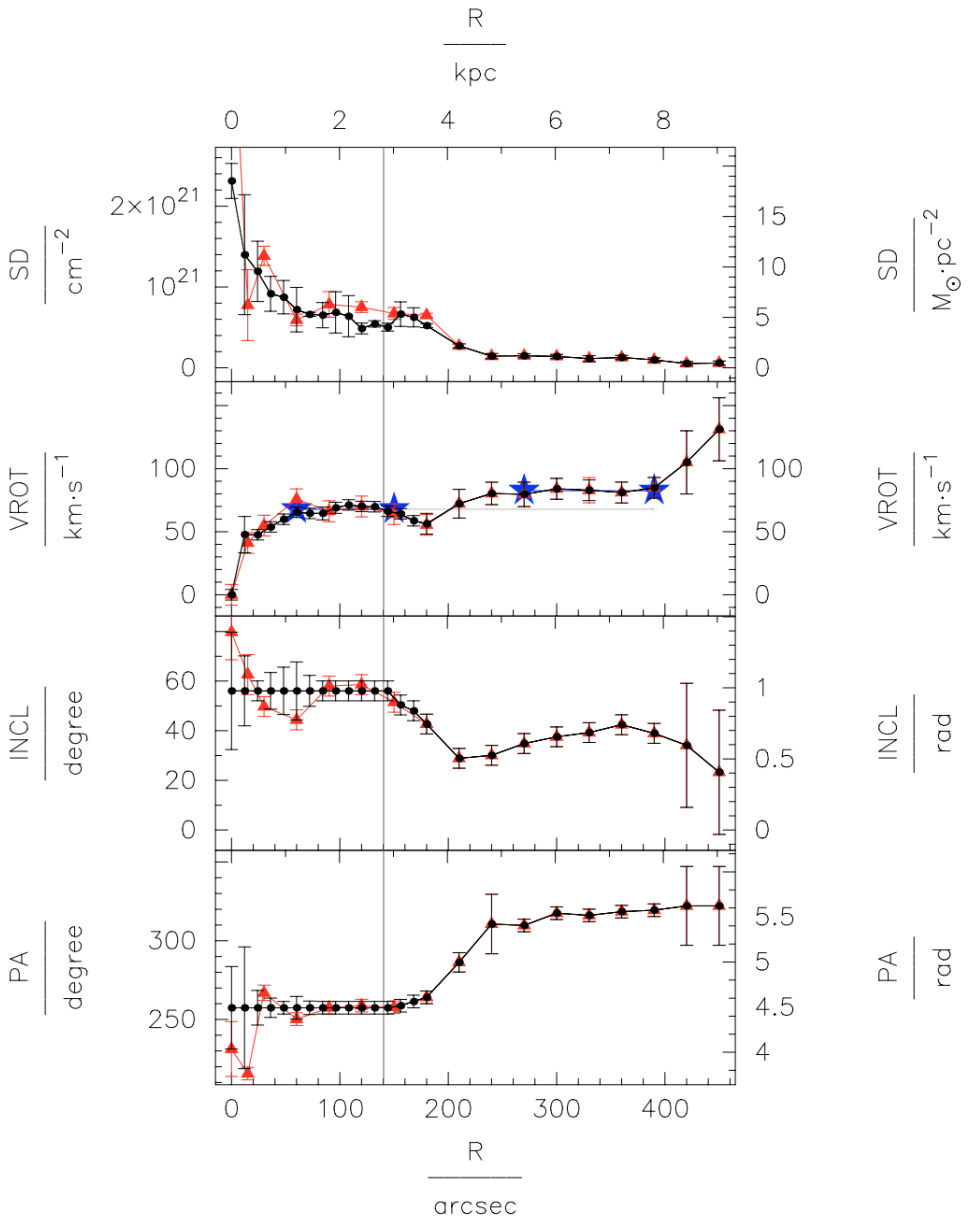


FIGURE 2.13: Tilted-ring parametrisations of NGC 5204 showing surface density, rotation velocity, inclination and position angle as presented in Józsa, 2007

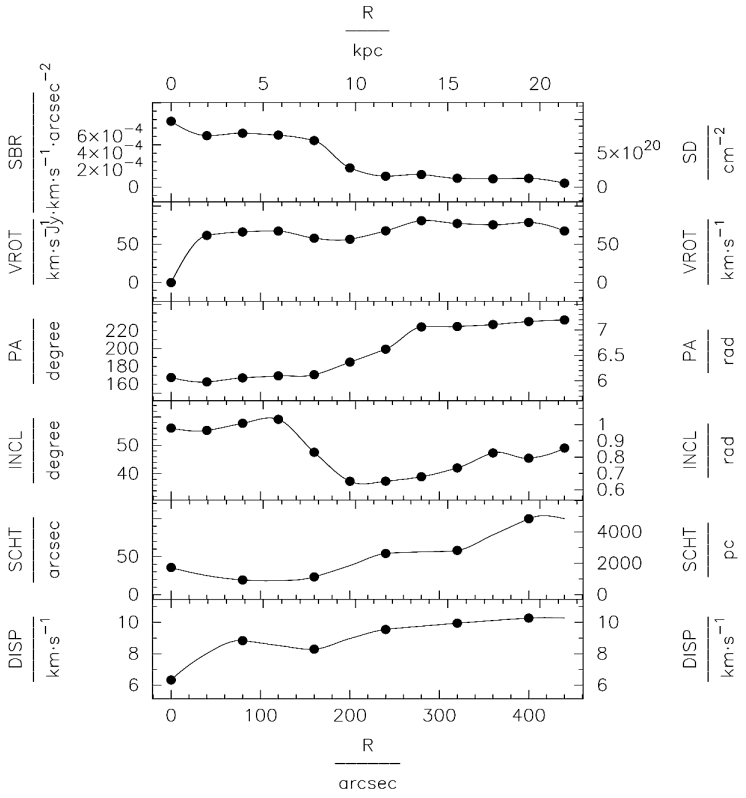


FIGURE 2.14: Parametrisation for case 1: variable dispersion.

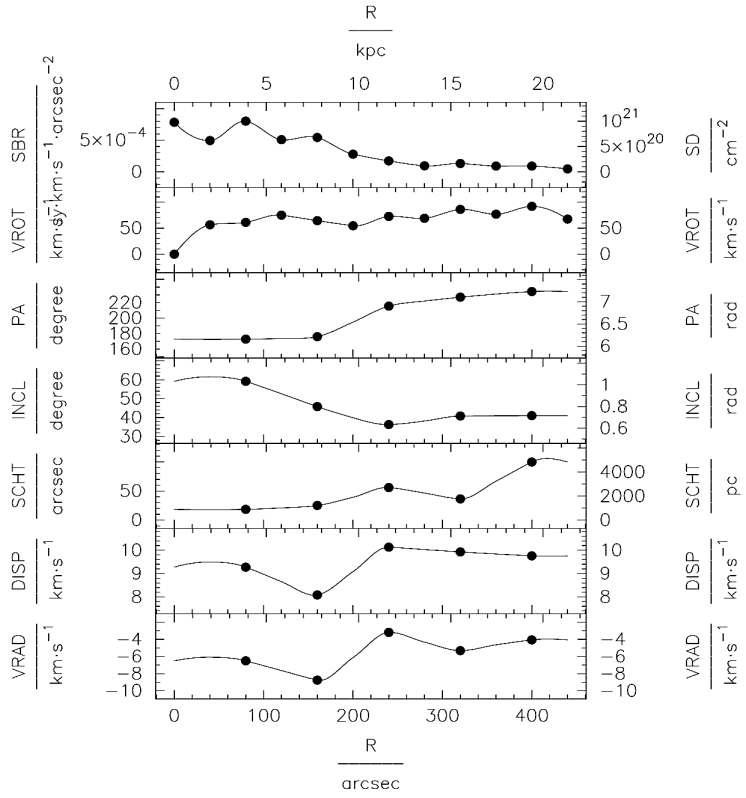


FIGURE 2.15: Parametrisation for case 2: radial motion.

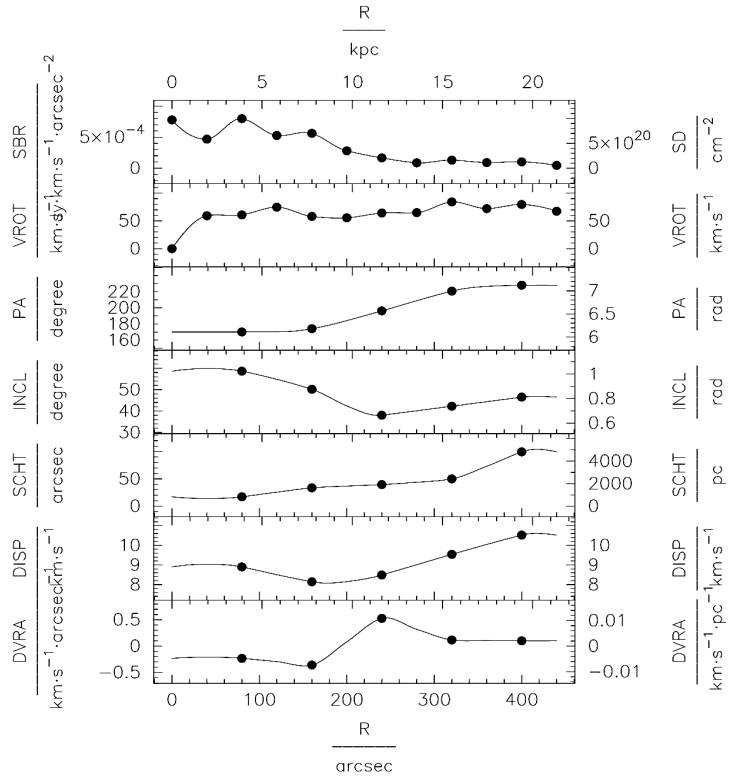


FIGURE 2.16: Parametrisation for case 3: radial motion varying with height above the plane.

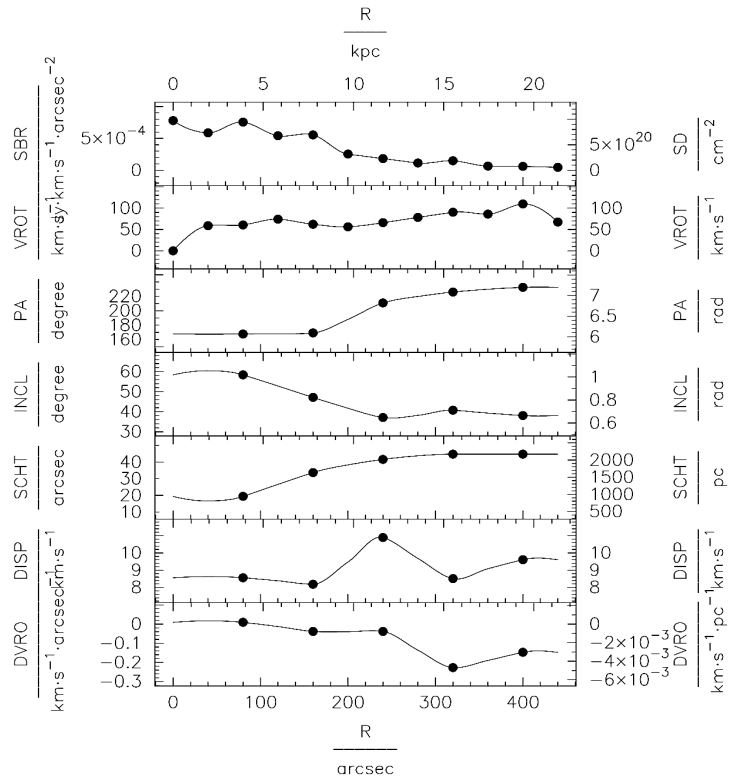


FIGURE 2.17: Parametrisation for case 4: tangential motion varying with height above the plane.

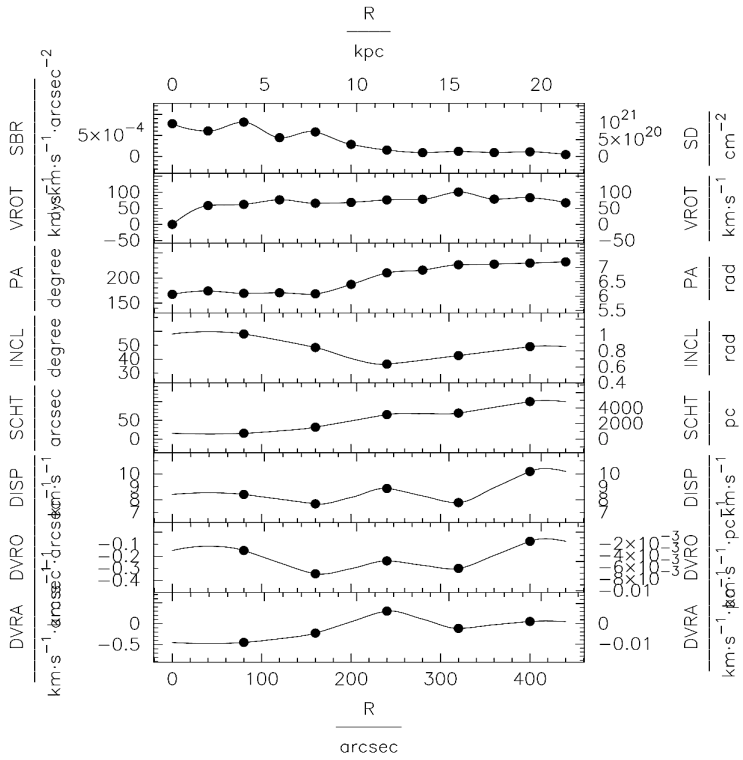


FIGURE 2.18: Parametrisation for case 5: radial and tangential motion together.

2.5 Discussion

The parameters **SBR, VROT, PA, INCL, SCHT, DISP, DVRO, DVR** and **VRAD** plotted in Fig. 2.14, 2.15, 2.16, 2.17 and 2.18 above represent surface brightness, rotation, velocity, position angle, inclination, scale height, dispersion, vertical gradient of rotation velocity and vertical gradient of radial velocity respectively. In Schmidt et al. (2014), they explored a number of kinematic models (the same cases in Sect. 2.4.1 which TiRiFiG was used to model) to search for signatures of gas accretion in more massive galaxies. Their results discuss the vanishing of the decline in the rotation curve when extraplanar gas is accounted for. The results of our parametrisations of NGC 5204 in Fig. 2.14, 2.15, 2.16, 2.17 and 2.18 above also shows vanishing of the bump in the rotation curve. This is also an indication of extraplanar gas which complements the work in Józsa (2007) showing some evidence of lagging extraplanar gas in NGC 2541, NGC 5204 and UGC 3580. The scope of this work is to demonstrate how the visual front end facilitates fitting with TiRiFiC. Further discussions on the kinematics of the extraplanar gas modelling using TiRiFiG are to be found in Twum et al. (in prep.).

2.6 Conclusion

In this chapter, TiRiFiG is described in detail highlighting the key software design objectives and the implementation. An overview of TiRiFiC (the current kinematic modelling software) is given with emphasis on its workflow.

TiRiFiC allows users to model gaseous H I disks of varied complexity. Since it uses the TRM to parametrise disk galaxies, it can become confusing to handle complex models using the parameter the file containing many parameters for each sub-ring. The way around this problem is TiRiFiG - a software that provides a visual front end to users to enable them to interact with the tilted-ring parameters. In Sect. 2.1.3 however, I mentioned TirEdit, a GUI provided by G. Heald, which is also available to edit TiRiFiC default files. Though TirEdit allows for a visual view and interactive update of `.def` files, it has very limited functionality in that it only provides for `.def` files to be displayed and modified on a viewgraph after it has been read and nothing more. By contrast, TiRiFiG provides additional functionalities such as allowing the user to start TiRiFiC from the interface coupled with other handy options to increase the flexibility of interacting with data points on the viewgraph. UML diagrams and the various modules used in the implementation of TiRiFiG are discussed. The outstanding design goal yet to be implemented is to link TiRiFiG to a visualisation tool which allows for data cubes to be viewed right from within the interface.

Chapter 3

Summary and Future Outlook

"*We are not at the end but at the beginning of a new physics. But whatever we find, there will always be new horizons continually awaiting us.*" ~Michio Kaku

3.1 Synopsis of this Thesis

With the SKA and its progenitors coming online soon, deep observations which will require detailed, complex and interactive modelling will be abundantly available. The work presented in this thesis is a visual frontend, called TiRiFiG, that allows users of the 3D kinematic modelling tool, TiRiFiC, to graphically interact with tilted-ring parameters before fitting.

Chapter 1 introduces astronomy as having begun with night star gazing to deeper observations using optical telescopes and finally, radio telescopes. The use of interferometry coupled with increased instrument sensitivity enabled us to achieve better angular resolution in the radio regime. This has helped us improve our understanding of galaxies and other cosmological constants in the universe. Hydrogen is known to be the most abundant element in the universe with neutral hydrogen (H I) uniformly distributed in the disk galaxies. H I has been widely used as a kinematic tracer in the kinematic study of disk galaxies. Some of the kinematic parameters that can be extracted from the H I emission line include rotation curves and gas mass profiles. The foremost of approaches used to parametrise galaxies is the tilted-ring model by Rogstad et al. (1974). This approach parametrises disk galaxies using a series of concentric rings, each ring having its own orientation and kinematic parameters. The TRM can be fitted to a velocity field (2D fitting) or directly to a data cube (3D fitting). Among many available parametrisation tools, TiRiFiC, a publicly available 3D kinematic modelling tool, developed by Józsa et al. (2007) uses the TRM to parametrise disk galaxies.

In Chapter 2, we looked in detail at the design objectives of TiRiFiG as well as its implementation. The window manager is handled by the C++ Qt GUI toolkit which has been wrapped in Python as PyQt. The problem addressed by this interactive interface is the complexity of manually editing the parameter file that arises when using TiRiFiC to parametrise complex models. There can be as many as 20+ parameters to be considered for each ring in the model. The more complex the model

is, the more confusing it is to edit the tilted-ring parameters in the parameter file in a text editor. As such, TiRiFiG has been developed to address this complexity. This very first release has been implemented to mainly allow for users of TiRiFiC to have a visual representation and interaction with tilted-ring parameters before fitting. Thus, the intent is to enhance the ease of use of the 3D kinematic modelling tool by visually interacting with the data points. The chapter concludes with an example fit using TiRiFiG.

3.2 Closing Remarks and Future Outlook

For complex models, TiRiFiG is a great tool for interacting with and modifying tilted-ring parameters for TiRiFiC before a fit is performed; not just before the fit but also during, for smoothing parameters and correcting outliers or adapting ring sizes in the outer parts.

This current implementation does not feature the link to the visualisation tool as described in Fig. 2.11. The immediate future work will be to incorporate this functionality in the current version and to also add the feature that allows the user perform the group move of data points specified in DG 2. Specific software packages have not yet been selected. However, the main idea here will be to allow the user to compare the observed galaxy to the parametrised galaxy right from the interface. Thus, features such as simultaneous display of cubes and overlays would be desirable. Possible visualisation options would consider volume rendering as well. Additional future prospects will be to steer the fitting process using this interface and produce analytical information (moment maps etc.) using custom written commands. TiRiFiG as any package, will be actively maintained and developed on `GitHub`[1]. With the official release, a wider user community base is expected and this will also bring bugs and design flaws to light to make TiRiFiG robust and much more interactive.

[1] https://github.com/gigjozsa/TiRiFiG

Appendix A

Python GUI Frameworks and TiRiFiG Code Implementation Details

This appendix contains two tables. One capturing a list of alternative GUI programming environments which could have been used for developing TiRiFiG and the other a list of all variables, functions and classes making up the code for TiRiFiG.

A.1 Python GUI Frameworks

A list of alternative Python GUI frameworks which could have been used to develop TiRiFiG other than PyQt.

Table A.1: Alternative PYTHON GUI Toolkits
Retrieved from The Python Wiki

GUI Toolkit	Features
pyFLTK	wrapper for FLTK, Fast Light Toolkit cross-platform small and fast access to OpenGL
PyGTK	powerful - aims to be a 1:1 GTK wrapper good GTK docs out there for both C and Python it is a wrapper - sometimes C idioms leak through
Pyglet	new OpenGL-y (3D)
PyGUI	cross-platform, utilizing the underlying system's GUI facilities designed to work smoothly with Python's language features and data types API is documented purely in Python terms (so it is not necessary to read documentation for an underlying toolkit)
PySide	Wraps Qt API compatible with PyQt LGPL License
PythonCard	Wraps WX Easy to use Graphic widget insertion Slightly slow

A.2 TiRiFiG Code Implementation Details

All classes, functions and attributes (and their respective types) implemented in the development of TiRiFiG.

Table A.2: Classes with respective attributes & functions

Class	**Attributes**	**Functions**
TimerThread	t «int» hFunction «function» thread «function»	__init__ handle_function start cancel
SMWindow	par «str» gwDict «dict» preParVal «str» counter «int» parameter «PyQt4.QtGui.QComboBox» xLabel «PyQt4.QtGui.Qlabel» xMin «PyQt4.QtGui.QLineEdit» xMax «PyQt4.QtGui.QLineEdit» yMin «PyQt4.QtGui.QLineEdit» yMax «PyQt4.QtGui.QLineEdit» radioFree «PyQt4.QtGui.QRadioButton» radioViewG «PyQt4.QtGui.QRadioButton» btnUpdate «PyQt4.QtGui.QPushButton» btnCancel «PyQt4.QtGui.QPushButton»	__init__ onChangeEvent

Table A.2 – continued from previous page

Class	**Attributes**	**Functions**
ParamSpec	parameterLabel «PyQt4.QtGui.Qlabel» parameter «PyQt4.QtGui.QComboBox» uMeasLabel «PyQt4.QtGui.Qlabel» unitMeasurement «PyQt4.QtGui.QLineEdit» btnOK «PyQt4.QtGui.QPushButton» btnCancel «PyQt4.QtGui.QPushButton»	__init__

Table A.2 – continued from previous page

Class	**Attributes**	**Functions**
GraphWidget	scaleChange	__init__
	«PyQt4.QtGui.QPushButton»	changeGlobal
		getClick
	mPress «list»	getRelease
	mRelease «list»	getMotion
	mMotion «list»	keyPressed
	mDblPress «list»	showInformation
	xScale «list»	firstPlot
	yScale «list»	plotFunc
	choice «str»	
	unitMeas «str»	
	par «str»	
	parVals «list»	
	parValRADI «list»	
	historyList «list»	
	key «str»	
	numPrecisionX «int»	
	numPrecisionY «int»	
	canvas	
	«matplotlib.backends.backend_qt4agg.	
	FigureCanvasQTAgg»	
	btn1 «PyQt4.QtGui.QPushButton»	
	btn2 «PyQt4.QtGui.QPushButton»	

Table A.2 – continued from previous page

Class	**Attributes**	**Functions**
MainWindow	ncols «int»	__init__
	nrows «int»	initUI
	INSET «str»	createActions
	par «list»	createMenus
	unitMeas «list»	quitApp
	tmpDeffile «str»	getData
	gwObjects «list»	strType
	t «int»	numPrecision
	scrollWidth «int»	getParameter
	scrollHeight «int»	openDef
	before «int»	setRowCol
	numPrecisionY «int»	saveFile
	numPrecisionX «int»	saveAll
	NUR «int»	saveMessage
	data «list»	saveAs
	parVals «dict»	saveAsMessage
	historyList «dict»	saveAsAll
	historyKeys «list»	slotChangeData
	historyKeysDup «dict»	animate
	xScale «list»	openEditor
	yScale «dict»	SMobj
	redo «list»	updateScale
	btn1 «PyQt4.QtGui.QPushButton»	updateMessage
		paraObj
	scrollArea	paramDef
	«PyQt4.QtGui.QScrollArea»	editParaObj
		editParamDef
	fileMenu «PyQt4.QtGui.QMenu»	tirificMessage
	runMenu «PyQt4.QtGui.QMenu»	startTiriFiC
	prefMenu «PyQt4.QtGui.QMenu»	

Appendix B

Software Documentation

This appendix is an additional documentation providing information on the installation and usage of the interactive modelling tool, TiRiFiG.

B.1 Installation

TiRiFiG depends on `PyQt4` and TiRiFiC to run. To install `PyQt4`, I suggest installing Anaconda first and then installing `PyQt4` using, "`conda install pyqt=4`". Download and installation notes for TiRiFiC are on its website[1]. TiRiFiG was developed and is maintained under a `GitHub`[2] public repository. It has been tested on an UBUNTU 16.04 and a macOS High Sierra v10.13.4 operating system. TiRiFiG can be installed from the terminal using `pip` by running "`pip install TiRiFiG`". If the command is run from the cloned repository, "`pip install .`" will also install it. Once installed, TiRiFiG can be added to a PYTHONPATH using "`export PATH=path_to_installation_directory:$PATH`".

B.2 Usage

To start TiRiFiG, run the following command from the terminal:

- `$ TiRiFiG`
- `$ python TiRiFiG-launcher.py`, if it was not installed using `pip`.

With the GUI running, the next steps are:

- Click 'Open' and select a `.def` file to load and visualise.
- Adjust data points for the parameter(s) using the mouse.
- Start TiRiFiC from run menu to perform fitting.

[1] http://gigjozsa.github.io/tirific/download_and_installation.html
[2] https://github.com/gigjozsa/TiRiFiG

Bibliography

Begeman, K. G. (1987). "HI rotation curves of spiral galaxies". PhD thesis. , Kapteyn Institute, (1987).

— (1989). "H I rotation curves of spiral galaxies. I - NGC 3198". In: *Astronomy and Astrophysics* 223, pp. 47–60.

Bershady, M. A. et al. (2010). "The DiskMass Survey. I. Overview". In: *Astrophysical Journal* 716, pp. 198–233. DOI: 10.1088/0004-637X/716/1/198. arXiv: 1004.4816.

Booch, G. et al. (2005). *The Unified Modeling Language User Guide*. 2nd ed. Addison-Wesley Professional. ISBN: 9780321267979.

Boomsma, R. et al. (2008). "HI holes and high-velocity clouds in the spiral galaxy NGC 6946". In: *Astronomy and Astrophysics* 490, pp. 555–570. DOI: 10.1051/0004-6361:200810120. arXiv: 0807.3339.

Bosma, A. (1978a). "The distribution and kinematics of neutral hydrogen in spiral galaxies of various morphological types". PhD thesis. PhD Thesis, Groningen Univ., (1978).

— (1978b). "The distribution and kinematics of neutral hydrogen in spiral galaxies of various morphological types". PhD thesis. PhD Thesis, Groningen Univ., (1978).

Conselice, C. J. et al. (2016). "The Evolution of Galaxy Number Density at z $<$ 8 and Its Implications". In: *Astrophysical Journal* 830, 83, p. 83. DOI: 10.3847/0004-637X/830/2/83. arXiv: 1607.03909.

de Blok, W. J. G. et al. (2014). "HALOGAS observations of NGC 4414: fountains, interaction, and ram pressure". In: *Astronomy and Astrophysics* 566, A80, A80. DOI: 10.1051/0004-6361/201322517. arXiv: 1405.2160.

Di Teodoro, E. M. and F. Fraternali (2015). "^{3D} BAROLO: a new 3D algorithm to derive rotation curves of galaxies". In: *MNRAS* 451, pp. 3021–3033. DOI: 10.1093/mnras/stv1213. arXiv: 1505.07834.

Ekers, R. D. and R. J. Allen (1975). "Interactive Computer Reduction and Display of Radio Super Synthesis Maps". In: *Image Processing Techniques in Astronomy*. Ed. by R. Nieuwenhuijzen and C. De Jager. Vol. 54. Astrophysics and Space Science Library, p. 309.

García-Ruiz, I. et al. (2002). "Neutral hydrogen and optical observations of edge-on galaxies: Hunting for warps". In: *Astronomy and Astrophysics* 394, pp. 769–789. DOI: 10.1051/0004-6361:20020976. eprint: astro-ph/0207112.

Gentile, G. et al. (2013). "HALOGAS: Extraplanar gas in NGC 3198". In: *Astronomy and Astrophysics* 554, A125, A125. DOI: 10.1051/0004-6361/201321116. arXiv: 1304.4232.

Gooch, R. (1996). "Karma: a Visualization Test-Bed". In: *Astronomical Data Analysis Software and Systems V*. Ed. by G. H. Jacoby and J. Barnes. Vol. 101. Astronomical Society of the Pacific Conference Series, p. 80.

Hubble, E. (1929). "A Relation between Distance and Radial Velocity among Extra-Galactic Nebulae". In: *Proceedings of the National Academy of Science* 15, pp. 168–173. DOI: 10.1073/pnas.15.3.168.

Jansky, K. G. (1933). "Radio Waves from Outside the Solar System". In: *Nature* 132, p. 66. DOI: 10.1038/132066a0.

Józsa, G. I. G. (2007). "Kinematic modelling of disk galaxies. II. A case-study of symmetrically warped galaxy disks". In: *Astronomy and Astrophysics* 468, pp. 903–917. DOI: 10.1051/0004-6361:20066165.

Józsa, G. I. G. et al. (2007). "Kinematic modelling of disk galaxies. I. A new method to fit tilted rings to data cubes". In: *Astronomy and Astrophysics* 468, pp. 731–774. DOI: 10.1051/0004-6361:20066164. eprint: astro-ph/0703207.

Kamphuis, P. et al. (2013). "HALOGAS observations of NGC 5023 and UGC 2082: modelling of non-cylindrically symmetric gas distributions in edge-on galaxies". In: *MNRAS* 434, pp. 2069–2093. DOI: 10.1093/mnras/stt1153. arXiv: 1306.5312.

Kamphuis, P. et al. (2015). "Automated kinematic modelling of warped galaxy discs in large H I surveys: 3D tilted-ring fitting of H I emission cubes". In: *MNRAS* 452, pp. 3139–3158. DOI: 10.1093/mnras/stv1480. arXiv: 1507.00413.

Koribalski, Bärbel S et al. (2018). "The Local Volume HI Survey (LVHIS)". In: *MNRAS*, sty479. DOI: 10.1093/mnras/sty479. eprint: /oup/backfile/content_public/journal/mnras/pap/10.1093_mnras_sty479/1/sty479.pdf. URL: http://dx.doi.org/10.1093/mnras/sty479.

Krajnović, D. et al. (2006). "Kinemetry: a generalization of photometry to the higher moments of the line-of-sight velocity distribution". In: *MNRAS* 366, pp. 787–802. DOI: 10.1111/j.1365-2966.2005.09902.x. eprint: astro-ph/0512200.

McMullin, J. P. et al. (2007). "CASA Architecture and Applications". In: *Astronomical Data Analysis Software and Systems XVI*. Ed. by R. A. Shaw et al. Vol. 376. Astronomical Society of the Pacific Conference Series, p. 127.

Mo, H. et al. (2010). *Galaxy Formation and Evolution*. Cambridge University Press. ISBN: 9780521857932.

Oh, S.-H. et al. (2018). "2D Bayesian automated tilted-ring fitting of disc galaxies in large H I galaxy surveys: 2DBAT". In: *MNRAS* 473, pp. 3256–3298. DOI: 10.1093/mnras/stx2304. arXiv: 1709.02049.

Parsons, A. R. et al. (2010). "The Precision Array for Probing the Epoch of Reionization: Eight Station Results". In: *The Astronomical Journal* 139, pp. 1468–1480. DOI: 10.1088/0004-6256/139/4/1468. arXiv: 0904.2334.

Persic, M. et al. (1996). "The universal rotation curve of spiral galaxies - I. The dark matter connection". In: *MNRAS* 281, pp. 27–47. DOI: 10.1093/mnras/281.1.27. eprint: astro-ph/9506004.

Peters, S. P. C. et al. (2017). "The shape of dark matter haloes - II. The GALACTUS H I modelling & fitting tool". In: *MNRAS* 464, pp. 21–31. DOI: 10.1093/mnras/stw2099. arXiv: 1608.05557.

Punzo, D. et al. (2017). "SlicerAstro: A 3-D interactive visual analytics tool for HI data". In: *Astronomy and Computing* 19, pp. 45–59. DOI: 10.1016/j.ascom.2017.03.004. arXiv: 1703.06651 [astro-ph.IM].

Reber, G. (1944). "Cosmic Static." In: *Astrophysical Journal* 100, p. 279. DOI: 10.1086/144668.

Rogstad, D. H. et al. (1974). "Aperture-synthesis observations of H I in the galaxy M83." In: *Astrophysical Journal* 193, pp. 309–319. DOI: 10.1086/153164.

Rubin, V. C. and W. K. Ford Jr. (1970). "Rotation of the Andromeda Nebula from a Spectroscopic Survey of Emission Regions". In: *Astrophysical Journal* 159, p. 379. DOI: 10.1086/150317.

Schmidt, S. et al. (2014). "Structure and kinematics of the nearby dwarf galaxy UGCA 105". In: *Astronomy and Astrophysics* 561, A28, A28. DOI: 10.1051/0004-6361/201118170. arXiv: 1311.4392.

Serra, P. et al. (2012). "The ATLAS^{3D} project - XIII. Mass and morphology of H I in early-type galaxies as a function of environment". In: *MNRAS* 422, pp. 1835–1862. DOI: 10.1111/j.1365-2966.2012.20219.x. arXiv: 1111.4241 [astro-ph.CO].

Smithsonian Astrophysical Observatory (2000). *SAOImage DS9: A utility for displaying astronomical images in the X11 window environment*. Astrophysics Source Code Library. ascl: 0003.002.

Sofue, Y. and V. Rubin (2001). "Rotation Curves of Spiral Galaxies". In: *Annual Review of Astronomy and Astrophysics* 39, pp. 137–174. DOI: 10.1146/annurev.astro.39.1.137. eprint: astro-ph/0010594.

Spekkens, K. and J. A. Sellwood (2007). "Modeling Noncircular Motions in Disk Galaxies: Application to NGC 2976". In: *Astrophysical Journal* 664, pp. 204–214. DOI: 10.1086/518471. eprint: astro-ph/0703688.

Thompson, A. R. et al. (2017). *Interferometry and Synthesis in Radio Astronomy*. 3rd ed. Springer. ISBN: 9783319444291.

Tully, R. B. and J. R. Fisher (1977). "A new method of determining distances to galaxies". In: *Astronomy and Astrophysics* 54, pp. 661–673.

Twum, S. N. et al. (in prep.). "Graphical kinematic modelling of disk galaxies". In: *MNRAS*.

van der Hulst, J. M. (2002). "The Westerbork H I Survey of Irregular and Spiral Galaxies, WHISP". In: Astronomical Society of the Pacific Conference Series 276. Ed. by A. R. Taylor et al., p. 84.

Wang, J. et al. (2013). "The Bluedisks project, a study of unusually H I-rich galaxies - I. H I sizes and morphology". In: *MNRAS* 433, pp. 270–294. DOI: 10.1093/mnras/stt722. arXiv: 1303.3538.

Wang, J. et al. (2017). "The Local Volume H I Survey: star formation properties". In: *MNRAS* 472, pp. 3029–3057. DOI: 10.1093/mnras/stx2073. arXiv: 1708.02744.

Zwaan, M. A. et al. (2005). "The HIPASS catalogue: Ω_{HI} and environmental effects on the HI mass function of galaxies". In: *MNRAS* 359, pp. L30–L34. DOI: 10.1111/j.1745-3933.2005.00029.x. eprint: astro-ph/0502257.

Zwicky, F. (1933). "Die Rotverschiebung von extragalaktischen Nebeln". In: *Helvetica Physica Acta* 6, pp. 110–127.