A Tilted-Ring Fit of NGC 3741 using TiRiFiC

&

Data Collator and Polyphase Filter Bank for the Phase-II Upgradation of ORT

Thesis submitted in partial fulfillment of the requirements of
BITS C421T/422T

By:
Deepthi Gorthi
ID No: 2010A3B5136P

Under the supervision of:
Dr. Jayaram Chengalur
Dean NCRA Faculty
Senior Professor
National Centre for Radio Astrophysics

Birla Institute of Technology and Science Pilani,
Pilani Campus

May 12, 2015

Acknowledgement

I am very grateful to Birla Institute of Technology and Science, Pilani, Rajasthan for creating and upholding a flexible system in which students can get practical experience in the field they are interested in. Without such a provision, I would not have got this valuable insight into the scientific research culture in India or the opportunity to work amongst them.

I would like to thank Dr. D.D. Pant, HOD of the Physics department at BITS, Pilani for trusting my skills and allowing me to pursue to my interests even though I did not start my project with a clear goal in mind.

Dr. Jayaram Chengalur, thank you for giving me this opportunity to intern at National Center for Radio Astrophysics and work with you. You gave me two well-thought-out projects, through which I could both contribute to science in my own small way and improve my knowledge of the subject. You were one of the most patient and knowledgeable teachers I had the honour of learning under, and your guidance was invaluabe to my progress on both these projects.

My sincere thanks to Dr. Tapomay Guha Sarkar for being a link between the BITS thesis system and my supervisor at NCRA, and ensuring a smooth evaluation process for this work. A big thanks for also guiding me and giving me the impetus to take interest in my work during the tough times.

I would like to thank Hemant Lokhande and Reena for taking care of my accomodation and making my stay here a smooth process. Students at NCRA, thank you for making me feel at home, your company has made my stay here very memorable. A special thanks to my parents, Kaustav Majhi and Himanish Ganjoo for their constant support and encouragement during my tenure as a project student.

Certificate

This is to certify that the Thesis entitled, **Tilted Ring Fit of NGC3741 using TiRiFiC & Data Collator and Polyphase Filter Bank for the Phase-II Upgradation of ORT** and submitted by **Deepthi B. Gorthi** ID No. **2010A3B5136P** in partial fulfillment of the requirement of BITS C421T/422T Thesis embodies the work done by him/her under my supervision.

Signature of the Supervisor

DATE: Name

Designation

Tilted Ring Fit of NGC3741 using TiRiFiC & Data Collator and Polyphase Filter Bank for the Phase-II Upgradation of ORT

Supervisor: **Dr. Jayaram Chengalur**
Semester: Second Session: Final Evaluation
Name of Student: **Deepthi Gorthi** ID No: **2010A3B5136P**

Abstract

Part I contains the details of the development of a Data Collator and a Polyphase filter bank for the Ooty Radio Telescope. Signals received by the telescope are digitized and routed to 8 HPC nodes such that each node gets a single time slice from all the dipole receivers. Each HPC node is an independent unit and is connected to 11 ethernet links. The data collator collects the data from all the 11 ethernet links and arranges them in time order in a shared memory, so that it can later be fourier transformed by the Polyphase Filter Bank and divided into spectral channels for analysis. The 11 ethernet links function simultaneously, requiring the data to be buffered by 11 different parallel processes. This requirement was addressed by using Open MP threads and this report explains in detail the code that performs this operation. The second program down the pipeline operates on this collated data and computed the frequency spectrum. The advantages of a polyphase filtering technique over a simple DFT or FFT algorithm is explained and the implementation of the filter bank using Intel IPP functions is discussed.

Part II discusses rotation curves of galaxies, their significance and the software tools available to plot rotation velocity diagrams accurately. An understanding of rotation curves from the perspective of a bachelors student and their importance in futuring science has been presented, followed by why rotation curves of dwarf galaxies, in particular, are interesting. The tilted-ring fitting process for kinematical modelling is introduced; refering to Begeman's pioneering work (Begeman, 1987). The advantages and disadvantages of kinematical modelling of disk galaxies using a newly available public software tool called TiRiFiC (Józsa et al., 2012) has been explained- its precedence over other softwares, and its flexibilities have been discussed, followed by an analysis of the results obtained from fitting the dwarf galaxy NGC3741 using this software. These results have been compared to the rotation curve obtained by Begum et al. (2005) and the performance of TiRiFiC has been discussed.

Contents

Part I

DATA COLLATOR AND PFB FOR THE ORT

Chapter 1

Introduction

The Ooty Radio Telescope, built in 1970 was designed to be run on completely analog electronics. Changes in technology since have demanded that the telescope be digitised to keep up with research trends. This project is part of the ongoing Phase II of the upgradation process.

The telescope is made of 22 modules, each module containing 12 elements or 48 dipoles (a group of 4 dipoles is called an element). Each module is digitized separately by an ADC box. The ADC data is packetized into 1600 timesamples for 2 elements. The data is 3 bit quantized and for every 20 time samples there is an additional 4 bit code. A header of 32 bytes is also added to each packet of the 1600 timesamples. With a sampling clock of 76.8 MHz frequency, the data rate from all the 264 elements comes out to be 7.742 Gbps. This data is multiplexed over 8 HPC nodes, so each node gets a data rate of 0.968 Gbps. There are 11 ethernet links for every node, hence each ethernet link sees a rate of about 90 Mbps. The code developed in this project is to be run at each HPC node separately.

The code takes the data from each ethernet link and rearranges the UDP packets in time order, based on the time stamp in the UDP header. It strips the packet of the header and collates the data from all the 11 ethernet links into one large shared memory, from where another program down the pipeline processes it.

The program has been tested at speeds of even 150 Mbps, much greater than the expected data rate at the ethernet link, and was found to be working satisfactorily. In the following chapters I have explained the structure and design of the code in more detail.

Chapter 2

Data Collator

The data collator program is run at each of the 8 HPC nodes, on the host computer. Every HPC node is connected to 11 ethernet links. The ethernet links work parallely, transmitting data from all the 264 elements. Hence to collate the data without any loss, each of the links has to be serviced by a different process.

An instance of a computer program being executed is called a process. A computer program can have a single process or multiple processes in the form of threads. So there were two ways of servicing the ethernet links: one by writing a seperate program for each link and another by using a multiple threaded program. The multithread program was a more convenient solution for collating the data since the threads could write the received data into a buffer that was shared amongst them before copying the buffer into a shared memory for futher processing.

The program receives data from the 11 ethernet links, collates them in a large circular buffer and copies this buffer periodically into a shared memory. The 11 ethernet links are serviced by 11 threads, the threads create one socket each, read the data coming into the socket and buffer it in a local private memory. Data coming through each link is buffered separately. A 12th thread scans for buffers that are ready to be copied and writes the buffered data into the right place in a temporary memory, while a 13th thread copies this to shared memory periodically.

The program uses Open MP threads for parallelization since they are easier to implement than POSIX threads. All threads are required to do the same work- arrange the recieved packets in time order, since the collating is done by a different thread. Open MP threads are task based and easier to implement

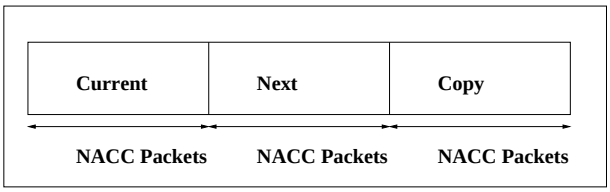

Figure 2.1: Structure of each of the circular buffers. NACC is the number of UDP packets that can be accomodated. The buffers are circularly shifted every time the current buffer gets filled.

when fine control over the threads is not necessary. Moreover, they are useful for scaling a given code. For instance, this program written in Open MP can be easily re-used for fewer links; say when one of the ethernet links is not functional. Most importantly, Open MP threads are portable. They are supported by multiple compliers across various platforms making the program nearly platform independent.

The UDP transmission protocol is being used to send the data from the bridge cards to the ethernet links. UDP protocol differs from TCP/IP in a number of ways, the primary being that UDP does not ensure time order. This means, packets that are sent out in one order are not received in the same order at the reciever. UDP protocol sends packets through the shortest route available at the time. This results in greater speed, but at the cost of accuracy.

Where both accuracy and speed matter, users usually write their own code to order the recieved data and still use UDP protocol. The Ooty Radio telescope also does the same. Hence, the received data has to be buffered locally and arranged in time order, before it can be copied into the shared memory.

2.1 Circular Buffer

The local circular buffer is different for each of the links. Each thread reads the UDP packet at its socket and checks for the time stamp in the UDP header, that has been written by the ADC right at the telescope. Depending on this time stamp, it is copied into the right place in its own circular buffer. The circular buffer is 3*NACC packets deep. See fig 2.1. The buffer is divided into three segments called *curr*, *next* and *copy*, each being NACC packets deep. At any given time, the packet recieved can be copied into either *curr* or

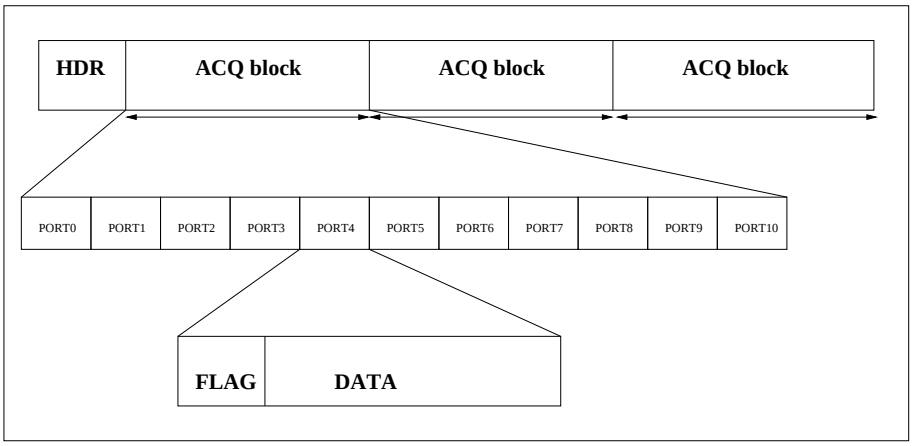

Figure 2.2: Structure of the shared memory. The number of ACQ blocks can be varied depending on the requirement. Each ACQ block accomodates all the ethernet links. The size of the data block for each ethernet link is fixed by the FFT parameters. The flag block is fixed proportionaly.

next. Hence the circular buffer is capable of time ordering packets that lie in this range only. Packets with time stamps beyond this range are discarded. Once the *curr* buffer is filled or the threshhold times out, the *curr* buffer is marked to be copied and the buffers are shifted circularly. That is, the *curr* becomes the *copy* buffer, the *next* buffer becomes *curr* and the previously *copy* buffer becomes the *next*. The program also flags the lost and unrecieved packets. Data statistics are shown on screen periodically.

2.2 Shared Memory

The 12th thread which is periodically checking for filled buffers copies the marked *copy* buffer into the shared memory and resets it for the next switch. The shared memory is divided into ACQ blocks with a header that contains information about each of them. (See fig 2.2) Each ACQ block is further divided into 11 segments for the 11 ethernet links. Each of the segments contains the actual data from the link along with flag content. Flag contains information about the number of packets recieved and the number lost. The actual size of data in the ACQ blocks is fixed by FFT parameters that will operate on this data later. Nevertheless, this program constraints the data block to be a multiple of NACC and UDP packet size. That is, the number of UDP packets that can be written into a data block must be a multiple of NACC.

Chapter 3

Program Delineation

The program is a stand-alone routine, and can be run on each of the host nodes connected to the telescope. The program creates the memory space required for the circular buffers and shared memory and frees the local memory before exiting.

3.1 Data Types and Structures

I have used a number of structures in the code, for easy data management. These structures can be found in the header files 'roach_udp.h' and 'memspace_udp.h'.

SockPar
This structure contains all the variables that a thread requires to collect data from the ethernet link and arrange it in time order. It contains:

1. fd: A socket is created for each ethernet link, to enable the program to collect data. This variable is the socket file descriptor.
2. sockaddr_in addr: This structure contains the address of the server that the client needs to connect to. In this case, it contains the address of the bridge card that transmits data to the HPC node.
3. ubuf[UDP_PAYLOAD]: This array temporarily carries the data in one UDP packet before it can be copied into the circular buffer.
4. sbuf[NSOCKBUF]: See SockBuf. This is an array of structures of type SockBuf. It is the circular buffer in which data is collated in time order.

The program fixes the value of NSOCKBUF (the number of buffers in the circular buffer) to three. Pointers to these 3 buffers are called *curr*, *next* and *copy*.

5. *curr*, *next*, *copy*: Pointers to the three buffers that constitute the circular buffer. Pointers simplify the switching which has to be done when *curr* is filled. Each of these buffers is NACC packets deep.
6. switch_thresh: The threshhold value that determines how long the program has to wait for *curr* buffer to get filled. When this parameter times out, *curr* is emptied with the unreceived packets flagged.
7. stat: See SockStat. This structure keeps track of the statistics of packets got, lost and discarded. These statistics are periodically shown on screen while the program is running.

SockBuf
This structure defines the buffer that is used store the data in the right time order. Each of the buffers in the circular buffer (*curr*, *next* and *copy*) are of this format. It contains:

1. data: This is a pointer to the actual data that the buffer will hold. This field is allocated memory of size NACC*UDP_DATA during program execution i.e, it can hold NACC number of UDP packets.
2. flag: This is a pointer to the metadata, which contains information about what packets have been received and how packets have been lost.
3. start: This variable contains the timestamp of the first packet in the buffer. Only packets with a timestamp greater than this are copied into the buffer.
4. stop: This contains the timestamp of the last packet in the buffer. Start and stop together control the range of timestamps are allowed to be copied into the buffer.
5. idx: An index is assigned to each buffer in the circular buffer to keep track of the buffer that has been copied last.
6. count: Keeps count of the number of packets received.

SockStat
This structure keeps track of the statistics of the packet arrival. Variables that start with 'd' hold the differential values (reset at every call to report

statistics) while the other variable hold the absolute values (values since the beginning of the program). Its members are:

1. start,dstart: These variables of structure type timeval record the start time of acquisition.
2. total,dtotal: Keeps track of the total number of packets sent.
3. got,dgot: Number of packets received.
4. lost,dlost: Number of packets that were never obtained.
5. bad,dbad: Number of packets that came completely out of sequence, i.e, packets that could not be copied into either *curr* or *next* buffers.
6. rate,drate: Contains estimate of the data rate in Mbps.
7. log_rate: Frequency (in terms of number of packets) with which the statistics have to be output.

AcqShmHdrType
This structure defines the header of the shared memory.

1. wr_blk: The ACQ blocks are filled sequentially. This variable contains the block number for the next write operation.
2. once_filled: This boolean variable is set when all the ACQ blocks have been filled at least once.
3. acq_mode: This can be either Sky Data or Simulation Data. The FFT programs that will act on the data, down the pipeline will differ based on this parameter.
4. blkinfo[MAX_ACQ_BLOCKS]: See AcqBlkInfoType. This holds the data and header of each block.

AcqBlkInfoType
This the structure of each ACQ block.

1. blknum: Contains the sequence number of the block.
2. seq:
3. flag: Pointer to flags (per UDP pkt) for this block char.
4. data: Pointer to data for this block.

AcqSizeType
This structure contains the sizes of various elements in the shared memory. These values are determined by the program that operates on the collated data down the pipeline.

1. tile_time_req: requested data chunk length per fft thread (sec)
2. tile_time: actual data chunk length per fft thread (sec)
3. fft_blksize: size of each fft block (bytes)
4. fft_blocks: num fft_blocks per tile (must be multiple of sta)
5. nblock: number of acq blocks in the shm
6. shm_size: total size of the acq shm (bytes)
7. mode: SimData or SkyData
8. npkt: UDP packets per acq shm block
9. flgsize: size of flag info in one acq shm block (bytes)
10. datsize: size of actual data in one acq shm block (bytes)
11. blksize: size of one acq shm block (data+flags)(bytes)

3.2 Functions

The following sections describe the execution of the code in detail. The execution of the code is broadly divided into three parts; data collection from the ethernet port (see 3.2.2), collation of data from all the ports (see 3.2.3) and copying data into the shared memory (see 3.2.4).

3.2.1 main

This is the first function that gets executed. This sets up the shared memory into which the data needs to be finally copied, initializes the sockets required to recieve data from the ports and initiates 13 threads to for executing the rest of the code.

3.2.2 acquire_socket_data

This function is executed parallely by 11 threads that service the 11 ethernet ports connected to the computer. Each thread receives data from its assigned

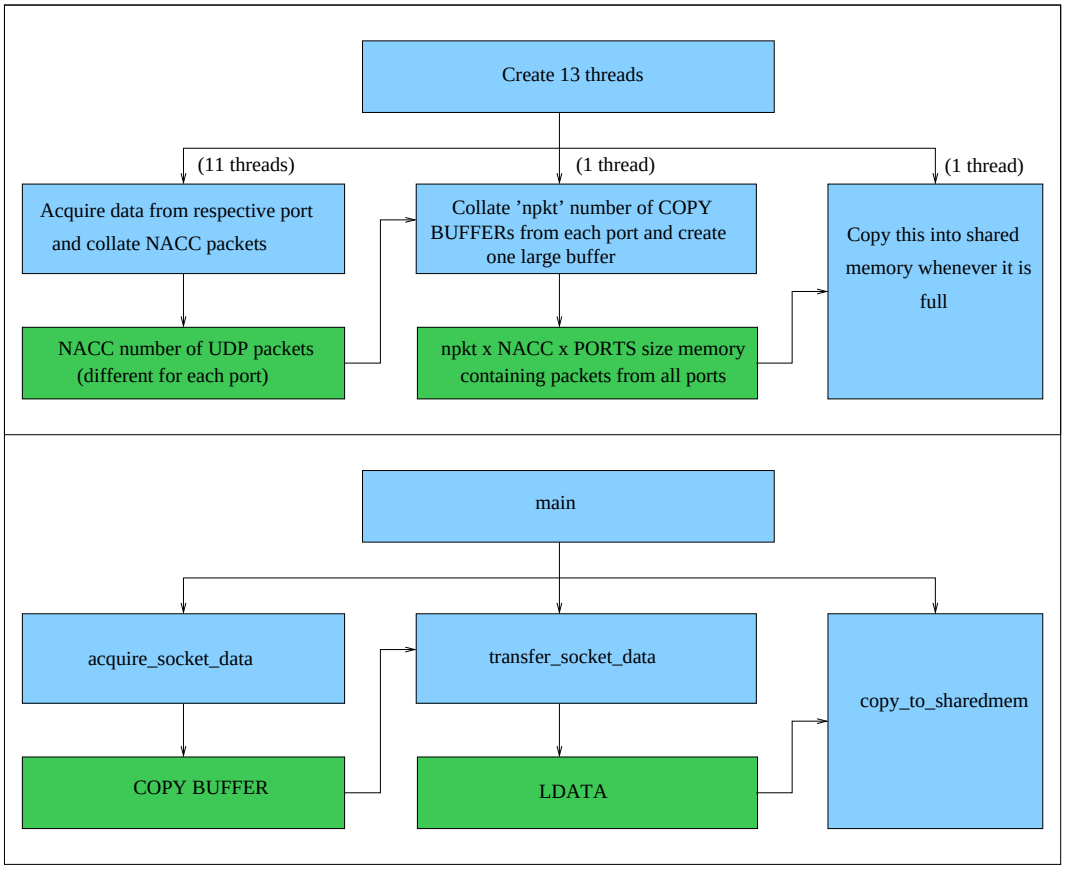

Figure 3.1: Flow-chart describing the working of the collator. The first part shows the sequence of functions performed and the second diagram shows the corresponding function in the code which performs the task. The blue coloured blocks are processes and the green coloured blocks are memory segments.

port and collates NACC number of UDP packets into a circular buffer (see figure 2.1). The code checks for the timestamp in the header of the UDP packet, discards the header and copies the data into its corresponding slot in the circular buffer. For a more detailed description of the function refer to section 2.1.

This function calls three other functions that i)Initialise the socket data ii) Initialise the statistic measures to keep track of data recieved and iii) Output the statistics to screen periodically.

init_socket
This function initialises the socket parameters and the circular buffers required for recieving and collating data. It initializes the variables of structure SockPar (refer to section 3.1).

init_sock_stat
This function initializes the variables in the structure SockStat (refer to section 3.1). It is called by the function init_socket.

report_socket_stat
This function is called everytime NACC number of packets are received by the program. It updates and prints the data statistics to screen.

3.2.3 transfer_socket_data

This function combines the data from all the threads/ports into one large buffer that is called *ldata* in the code. The buffer is ordered exactly like the shared memory (see figure 2.2) and is copied as it is into the shared memory in the next step.

This function executes linearly and checks for filled local buffers from each port sequentially. Given that the local buffers are only 3*NACC packets deep, it is possible that a few buffers from each port are missed if it does not collect data at a rate faster than they are getting filled. In the current form of the code, the rate of data arrival and rate of copying are balnced by the UDP packet size, value of NACC and size of the *ldata* buffer. Changing either of the parameter might affect the rate of data transfer and result in a loss of packets.

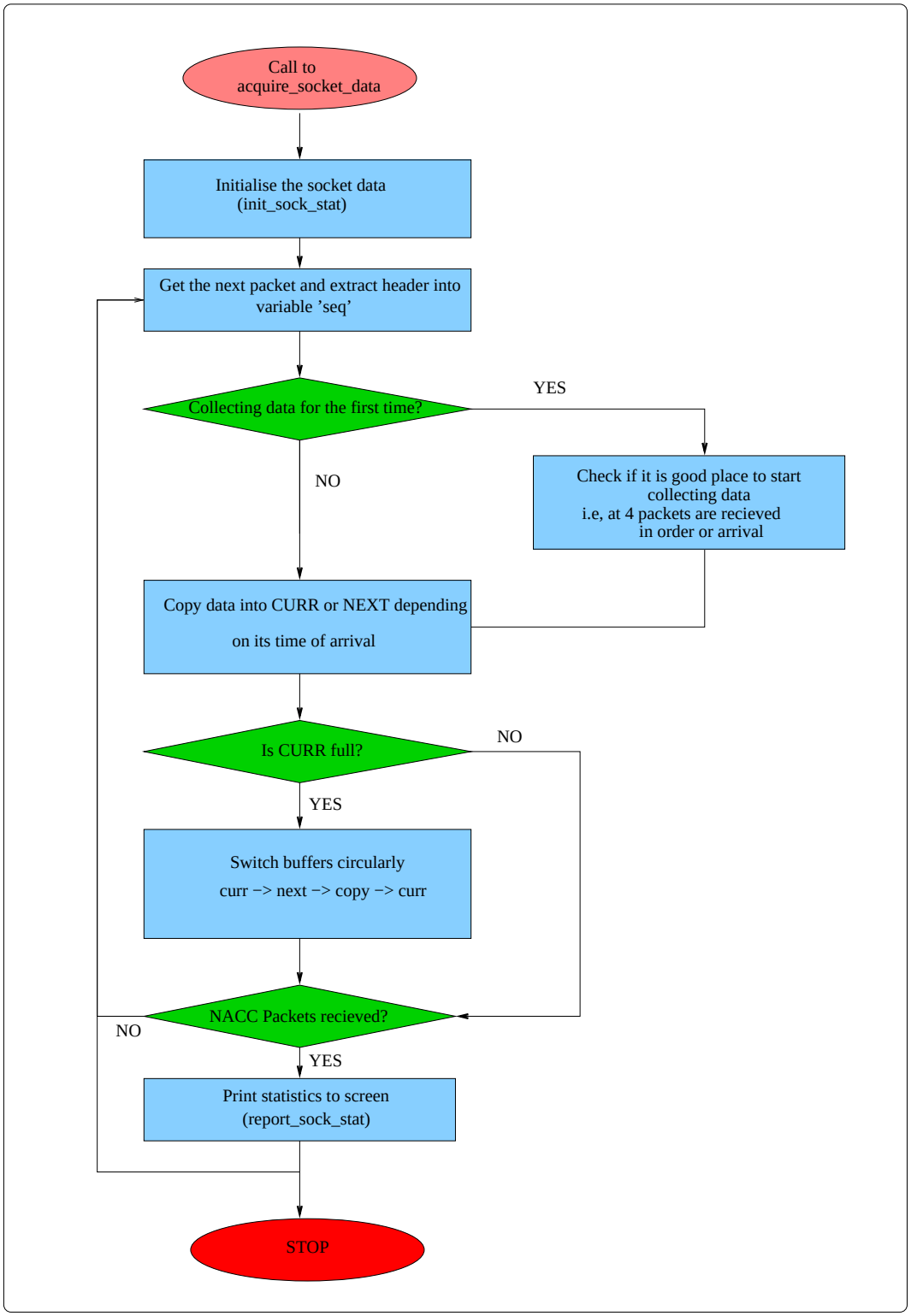

Figure 3.2: Execution flow inside the function acquire_socket_data.

init_asize
The size of *ldata* is determined by the program that operates on the data next. The parameters of the shared memory contained in the structure AcqShmType are initialised by this function (see paragraph(3.1).

3.2.4 copy_to_sharedmem

This function copies *ldata* into the shared memory. Since it is not an atomic process and takes time to execute, this process needs to be run by a separate thread that does not interfere in any other data collation process.

The data stored in this shared memory is converted into a frequency domain signal by another program down the pipeline. This program uses a polyphase filtering technique to perform this computation in a robust way. The polyphase filtering method and the program structure is discussed in the next two chapters.

Chapter 4

Polyphase Filtering Technique

In digital signal processing, an instrument or software that performs a fourier transform computes the discrete fourier transform (DFT) of the input signal. The algorithm that is used to compute the DFT in an efficient way is called the fast fourier transform (FFT) algorithm. DFT is performed on a discrete time domain signal and results in a discrete frequency signal. For an input signal $f(t)$, say the discrete time samples are given by $f_n = f(nT_s)$ where T_s is the time sampling interval. Its DFT is computed using the formula:

$$F_m = \sum_{n=0}^{N_0-1} f_n e^{-2\pi jmn/N_0} \tag{4.1}$$

where N_0 is the number of samples f_n within the time period T_0 of the input signal, if the signal is periodic; or the total number of samples being used to compute the DFT, if the signal has no periodicity. F_m the m^{th} frequency sample (also called frequency bin), i.e. $F(m\omega_0)$, where ω_0 is the periodicity of the frequency signal given by:

$$\omega_0 = \frac{2\pi}{T_0} \tag{4.2}$$

As evident from the above equation, the fourier signal is computed only at frequencies that are multiples of the above fundamental frequency, ω_0. This in turn means that, depending on the sampling frequency and the number of points in the DFT, it is possible that an input tone appears in more than one frequency bin. This is called leakage loss. If the input tone is not very strong, this effect can go unnoticed but in astronomical signal this effect can

become important in the presence of Radio Frequency Interference (RFI). Strong RFI can drown the signal in nearby bins, leading to loss of data.

Moreover, notice that by taking a finite signal in the time domain, we are effectively multiplying the infinite input signal with a tophat function in the region of interest before computing its fourier transform. This becomes equivalent to convolving the fourier transform of the input signal with the fourier transform of the tophat function, that is the sinc function. Fortuitous combinations of N_0, T_s, and the input frequency can be such that the zeroes of the sinc function coincide with the bin centres of all other frequencies, in which case the problem is non-existent. But in a general case, the frequency domain bin centres lie at non-zero locations on the sinc function, thus making the single-bin frequency response non-flat in nature. This is called scalloping loss.

The polyphase filtering technique helps in over-coming these two significant drawbacks of DFT/FFT. The solution to DFT leakage involves suppressing the side-lobes of the aforementioned sinc function by changing the single-bin frequency response of the DFT to approximate a rectangular function. This is achieved by, roughly speaking, multipying the infinite input signal with a sinc function instead of a tophat function.

In the actual implementation, since we cannot take an infinite signal as the input to any computation, we take instead a block of time signal that is say, P times the actual length that we require, say M. This block of data is multiplied with a sinc function of the same length to get a flat single-bin frequency response. This multiplied signal is then folded to obtain the original length (M), before computing the fourier signal through a normal fourier transform. Mathematically, the process can be represented as follows, if $L = P \times M$:

$$y(l) = \sum_{p=0}^{P-1} f(pM + l)h(pM + l) \tag{4.3}$$

where the sub-filter coefficients $h(l + pM)$ correspond to what are called P-tap polyphase sub-filters. The N such polyphase sub-filters that make up the above operation, together with the FFT filterbank stage that follows this, are collectively called a 'Polyphase Filter bank'. The only difference between the sub-filters is their phase response, which is why this structure is called a 'polyphase' filter bank.

Another way to view the mathematical interpretation would be to work back-

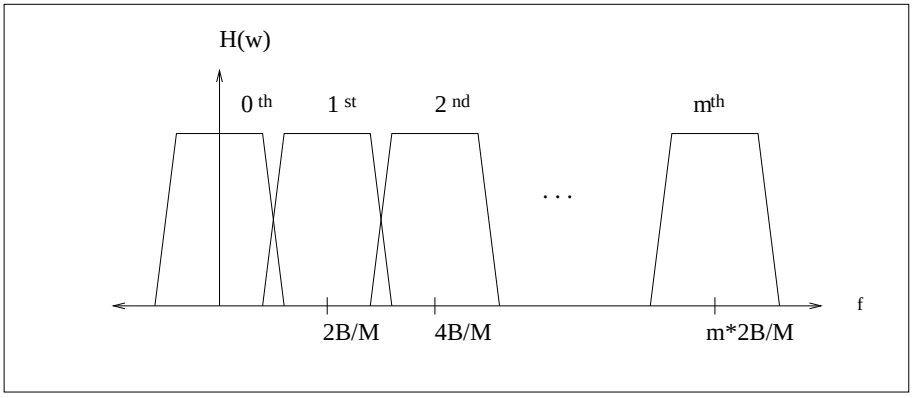

Figure 4.1: Expected frequency response of the polyphase filtering technique. An input signal of length M in the time domain, will result in a frequency resolution of $\frac{2B}{M}$ which is where we want the bin centres to be fixed.

wards from the output. We want the filter bank in the frequency domain to be a series of tophat functions that looks like figure 4.1. We consider M such subbands in the frequency domain to account exhaustively for all the point in one slice of the time domain that we will consider. The m^{th} subband can be written as,

$$h_m(n) = h(n)e^{-j2\pi mn/M} \tag{4.4}$$

where $h(n)$ stands for the tophat function at zero frequency. For the input signal mentioned above- $f(n)$, the output of this m^{th} subband will be,

$$\begin{aligned} Y_m(n) &= \sum_{i=0}^{L-1} h_m(i) f(n-i) \\ &= \sum_{i=0}^{L-1} h(i) e^{-j2\pi mi/M} \cdot f(n-i) \end{aligned} \tag{4.5}$$

We can again break this as $L = P \times M$ as we did above, which gives us,

$$Y_m(n) = \sum_{l=0}^{M-1} \sum_{p=0}^{P-1} h(pM+l) f(n-pM-l) e^{-j2\pi ml/M} \tag{4.6}$$

Putting $h(pM+l) = h_l(p)$ and $f(pM+l) = f_l(p)$ in the above equation we

can see a reduced expression which can be condensed immediately to look like equation 4.4.

$$\begin{aligned} Y_m(k) &= \sum_{l=0}^{M-1} \sum_{p=0}^{P-1} h_l(p) f_l(k-p) e^{-j2\pi ml/M} \\ &= \sum_{l=0}^{M-1} y_l(k) e^{-j2\pi ml/M} \end{aligned} \tag{4.7}$$

You can see that equation 4.7 is exactly same as equation 4.3 which we obtained from the time domain analysis.

In the next chapter I have discussed the implementation of a PFB with Intel IPP functions. This program operates on the data collected by the data collator program in the shared memory. The end result of the PFB computation is a frequency domain signal that has a flat response across the channel and provides excellent supression of out-of-band signals like RFI.

Chapter 5

PFB- Implementation with Intel IPP

Intel Integrated Performance Primitives (IPP) is a multi-threaded software library of functions for multimedia and data processing applications, produced by Intel. The functions take advantage of the processor features and perform the computation more efficiently that a program written for the same purpose in a high level language like C. The IPP functions allow different sizes of data, like the datatypes in C, for more efficient computing. I chose to write this program using IPP (version 8.2) since:

1. IPP has an inbuilt complex datatype and most of its functions can directly work with these complex numbers. Hence, the task of defining structures to handle complex numbers is reduced.
2. Arithmetic operations can be performed on arrays. The O(n) time order loops that are required to perform simple additions and multiplications (during PFB computations) are implemented in IPP to be much faster process processes, making the program more efficient.
3. Fourier transforms can be computed using an inbuilt function. Though the function does not execute in zero order, it is more efficient than the FFT algorithm written in C.

The PFB program reads data from the shared memory that was created and populated by the data collator program. It multiplies a chunk of 512*16 data-points with a similar length sinc function before folding it to a 512 point signal and computing its fourier transform. The following sections describe the functions implemented in more detail.

5.1 Structures, Macros and Variables

5.1.1 Macros

The data handling is controlled by four macros:

1. Ldata: The length of one data segment for which we wish to compute the fourier transform.
2. fold: The number of times you want to fold the input signal.
3. L: The length of the input signal that is taken according to the fold value, i.e. Ldata*fold. This is the length considered for multiplication with the window function.
4. Lmem: The integral number of Ldata units that are present in the shared memory. Presently, this value is just being used as a stopping factor to exit the computation and can be ignored if the process has to be run continuously till manual interference shuts it down.

The other macros in the program handle the shared memory segment and are not relevant to the polyphase filtering process.

5.1.2 Structures

The format of the data stored in the shared memory is defined in a structure called 'Input'. Currently, the structure only holds an array of real numbers of length L into which data from the shared memory is loaded. It can be modified in the future to accomodate more complex datatypes.

5.1.3 Variables

Some of the variables that hold important data are:

1. B: Contains the bandwidth of the input signal. This can be fixed based on the telescope settings that are used to observe the signal.
2. h[L]: Holds the sinc function that is used to weight the input signal before folding.

3. D: Carries the width of the windowing sinc function. This width is tuned such that the corresponding fourier domain top-hat function has a width exactly equal to the frequency bin width.
4. y[Ldata]: Contains the final output after the polyphase filtering process.

5.2 Working

The function that implements the polyphase filtering performs three main tasks in a loop until all the data in the shared memory has be processed. First, it loads a length 'L' of the data from the shared memory into the 'Input' structure. Using an IPP function called 'AddProduct', it simultaneously multiples the input signal and the sinc function, and folds the resultant product [1]. The function structure is:

5.2.1 Folding in IPP

A special function call 'AddProduct' ensures fast and efficient folding. The format of the function is,

```
IppStatus ippsAddProduct_64f(const Ipp64f* pSrc1,
   const Ipp64f* pSrc2, Ipp64f* pSrcDst, int len);
```

pSrc1 Pointer to the source vector that contains the multiplicand

pSrc2 Pointer to the source vector that contains the multiplicant

pSrcDst Pointer to the vector that contains the folded signal

len Length of the above vectors

This function deals with 64bit float integer arrays. Mathematically, the resultant function can be denoted by:

```
pSrcDst[n] = pSrcDst[n] + pSrc1[n]*pSrc2[n], 0<n<len
```

An (Ldata)-point FFT is performed on the weighted folded signal after this to obtain the final answer. The FFT computation is not very straightforward. It requires a few setup instructions in IPP to perform an efficient fourier transform.

[1] For a detailed description of IPP functions see Intel Integrated Performance Primitives, Reference Manual, Volume 1: Signal Processing

5.2.2 Fast Fourier Transform in IPP

As mentioned in Chapter 4, an FFT is just an algorithm to compute the DFT of a signal at a faster rate. The most common algorithm for FFT is called the Cooley-Tukey FFT algorithm and works on a divide and conquer principle. The DFT is broken down in to two parts iteratively; this is done by splitting the input signal into the even time samples and odd samples and multiplying both parts with corresponding common factors known as 'twiddle factors'. [2]

The FFT implementation in IPP requires pre-calculated twiddle factors for faster multiplication. The setup leading to FFT computation requires one to estimate the amount of memory required to store the twiddle factors and compute and populate the array. These tasks also well-defined IPP functions and just need to be called before the FFT function.

First, we need to compute the memory space required for storing the twiddle factors. This is computed by the function:

```
IppStatus ippsFFTGetSize_R_64f(int order, int flag,
   IppHintAlgorithm hint, int* pSpecSize, int*
   pSpecBufferSize, int* pBufferSize);
```

order FFT order. The input signal length is N= 2 order.

flag Specifies the result normalization method.

hint The value to ippAlgHintNone (is not used by this function).

pSpecSize Pointer to the FFT specification structure size value.

pSpecBufferSize Pointer to the buffer size value for FFT initialization function.

pBufferSize Pointer to the size value of the FFT external work buffer.

The FFT specification structure contains data such as the table of twiddle factors. The buffer for the FFT initialization function, (size held in pSpecBufferSize), is the workspace required for computations that will populate the FFT specification structure. The external work buffer space, if allocated, improves performance by avoiding allocation and deallocation of

[2] For the full algorithm refer to Prokais & Manolakis (2007)

internal buffers. After computing the sizes of the required buffers and alloting the memory space, the specification structure needs to be initialized. This is done by the function,

```
IppStatus ippsFFTInit_R_64f(IppsFFTSpec_R_64f**
   ppFFTSpec, int order, int flag, IppHintAlgorithm
   hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer );
```

ppFFTSpec Double pointer to the FFT specification structure to be created.

pSpec Pointer to the area for the FFT specification structure.

pSpecBuffer Pointer to the work buffer.

order, **flag** and **hint** have the same values and meanings as mentioned for the previous function. After this pre-computation setup the FFT can be computed using the function,

```
IppStatus ippsFFTFwd_RToPack_64f(const Ipp64f* pSrc,
    Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec,
    Ipp8u* pBuffer );
```

pSrc Pointer to the vector containing the input function (real signal).

pDst Pointer to the destination where the fourier transform can be stored.

pFFTSpec Pointer to the FFT specification structure.

pBuffer Pointer to the work buffer.

The output of the FFT can be stored in three different ways: **Perm**, **Pack** or **CCS**. These three formats refer to the way the complex and real parts of the solution array are stored. These are stored in a memory saving format and need to be decoded before writing to file. You can use an IPP function to decode this and populate a complex array,

```
IppStatus ippsConjPack_64fc(const Ipp64f* pSrc,
   Ipp64fc* pDst, int lenDst);
```

pSrc Pointer to the vector containing the fourier output in Pack format.

pDst Pointer to the destination where the fourier transform can be stored in complex form.

lenDst Length of the destination vector.

This final result can be written out to file or another shared memory for futher processing. Though it is more efficient to perform the multipying, folding and fourier transform in one step in C, the three nested loops that perform this task take longer to execute than computing the FFT separately in IPP.

Chapter 6

Conclusion

Data Collator program ensures that the digitized signal coming from the telescope to the control room is in same time sequence as it was at the telescope. This is a crucial step in ensuring that the conclusions drawn from analysis of this data are correct. Wrong timing information can lead to patterns where there aren't any, and these might be interpreted as coming from the source. They could also erase patterns that actually exist resulting in loss of information from the source.

The code in its current form ensures lossless time ordering of data from all the 11 ports connected to the host it is running on. The rate of data arrival at the host is about 125Mbps and writes data to the ROM at nearly the same average speed in bursts. The code does not perform any data reduction and merely resequences it if necessary.

I tested the code under the following situations and found it to be operating efficiently.

1. Out of sequence packets at the start of data receival: The code runs only if it can find at least four packets in the correct sequence, if 1024 packets come without any particular sequence, the code abhorts.
2. Out of sequence packets after starting: Packets that were sent slightly out of sequence and completely out of sequence were shifted and copied and dropped respectively.
3. Fewer ports: The code works even if one of the ports connected to the host is nonfunctional. But a better way to solve this problem would be to change the macro 'PORTS' that specifies the number of working ports.

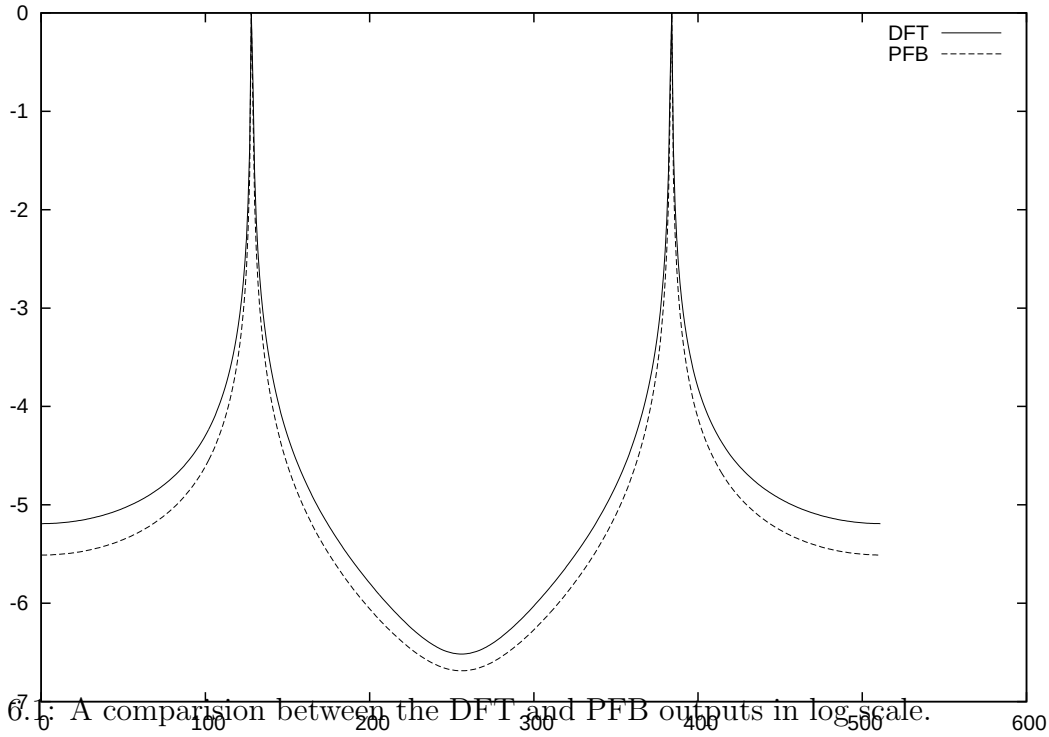

Figure 6.1: A comparision between the DFT and PFB outputs in log-scale.

The PFB code has been tested against a simple DFT for an input cosine of frequency that is a non-integral multiple of the fundamental frequency of the transform. Though the PFB computation takes, on a average, about 1.5 times the resources and time, from the graph it is evident that the output in case of a PFB is much more effective in supressing out-of-band signals.

Both the codes have been commented sufficiently and should be easy to follow in case someone wishes to modify them at a later stage.

Part II

TILTED RING FIT USING TiRiFiC

Chapter 7

Introduction

Uptill the 1960s, scientists believed that galaxies are static objects that are moving away from each other according to the Hubble's law. In the 1960s, an American astronomer- Vera Rubin proposed and found that galaxies, much like planets, rotate about their centers. A rotation curve for a galaxy is a map of the rotational velocity of the galaxy as a function of its radius.

We can theoretically predict what this rotation curve should look like. We can model the galaxy as a dark matter sphere with a uniform distribution of particles of density ρ. Balancing the gravitational force and the centrifugal force at every point of this sphere we can get a rough picture of the expected trend in radial velocity.

If the velocity field is given by $v(r)$; for a differential element $\mathrm{d}m$ inside the sphere we can write,

$$\frac{(\mathrm{d}m)v^2(r)}{r} = \frac{G\rho(\frac{4}{3}\pi r^3)(\mathrm{d}m)}{r^2}$$

$$\Rightarrow v(r) \propto r \tag{7.1}$$

and for an element outside the sphere we can write,

$$\frac{(\mathrm{d}m)v^2(r)}{r} = \frac{GM(\mathrm{d}m)}{r^2}$$

$$\Rightarrow v(r) \propto \frac{1}{\sqrt{r}} \tag{7.2}$$

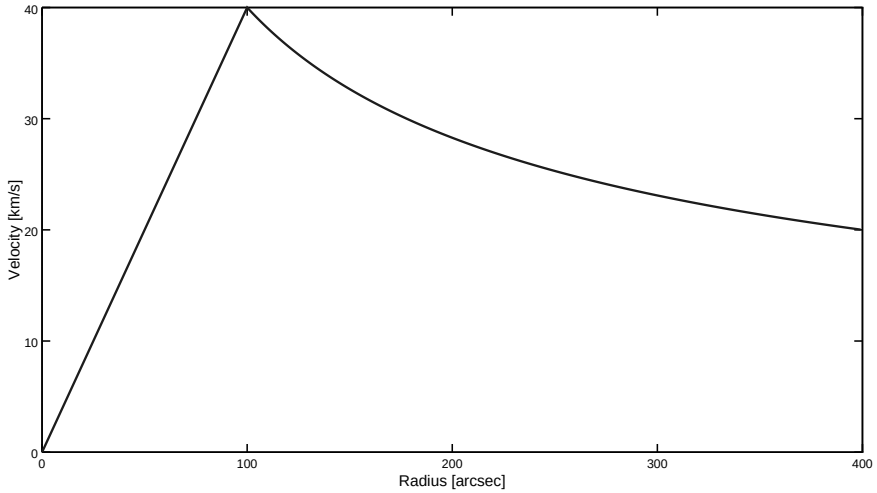

Figure 7.1: Theoretically predicted rotation curve of a galaxy assuming that the galaxy is 100 arcsec wide and the maximum rotation velocity is 40 km/s, at the outer edge of the galaxy.

The above figure shows the expected trend of the theoretical prediction. Even accounting for the fact that most galaxies do not have a sharp edge, as assumed in this plot, we can say that the profile well within the galaxy and well outside the galaxy should match this prediction i.e., the rotation curve should fall with distance outside the galaxy.

Vera Rubin (Rubin et al., 1980), studied the rotation velocity profiles of 21 Sc[1] galaxies and concluded that they do not fall after crossing the outer luminosity edge. The abstract of her paper reads,

"All curves show a fairly rapid velocity rise initially and a slow rise thereafter. Most velocity curves are rising slowly even at the farthest measured point. Neither high nor low luminosity Sc galaxies having falling rotation curves."

This peculiar deviation from prediction is accounted for by 'invisible' matter around the galaxy which is called Dark Matter, since it does not interact with electromagnetic radiation of any wavelength.

From this initial discovery till now, we have made a lot of progress in estimating the dark matter content in galaxies. But still we know only little

[1]According to the Hubble classification, these are spiral galaxies with loosely wound spiral arms with a small faint bulge. They can be clearly resolved into individual stellar clusters and nebulae making the rotation velocity observation easier.

about the distribution of the dark matter content inside galaxies. The main problem in determining this is geometry- galaxies are not uniform flat disks as we assumed. They are non-homogenous and most galaxies also exhibit warps. Moreover, the gas inside galaxies has random motion along with a uniform rotation velocity. This means that the spectral red shift observed cannot be assumed to be a direct function of the rotation velocity alone. This is complicated by the galaxy having a finite thickness and varying inclination at different radii.

In 1974, Rogstad et al. (1974) proposed a simple yet powerful technique to obtain rotation curves. This algorithm, which is still in use today, is called the tilted ring model and is explained in more detail in the next chapter. For my thesis I have used a modified version of the same algorithm to analyse the dwarf galaxy NGC3741.

Dwarf galaxies, as the name suggests, are small galaxies. They are predominantly found in galaxy clusters, mostly as companions to larger ones. Though they are the most abundant type of galaxy, they are very difficult to detect due to small size and low luminosity. Scientists believe that dwarf galaxies are the Lego bricks out of which the large galaxies that we see today have formed. The Λ-CDM model predicts that small objects were the first to form, and these coalesced to form massive objects. Further mergers lead to more massive objects, in a 'hierarchical merging' fashion. Moreover dwarf galaxies have more dark matter content than larger ones. This makes dwarf galaxies interesting objects to observe and analyse. Their rotation curves give insight into primodial galaxy formations and the Λ-CDM model itself.

The intention of starting this project was to analyse dwarf galaxies with a new rotation curve plotting software- TiRiFiC (Józsa et al., 2012). Since the performance of the software did not match expectation and due to a heavy time constraint, I had to restrict myself to analysing only one dwarf galaxy. The results obtained from TiRiFiC come close to the published results and do not show any new trends that have not already been accounted for. The rest of the report descibes the algorithm TiRiFiC is built on and its performance.

Chapter 8

Tilted Ring Model

The fact that all galaxies do not exhibit a flat disk-like structure, when viewed edge on, became apparent by the 1960s when scientists discovered a warp in all the major galaxies of the local group[1], including in our own galaxy. But in the 1970s, high resolution 21-cm maps[2] made it evident that galaxy warps are not confined to the local group.

Velocities in astronomy are measured using spectral redshift. Doppler effect tells us that objects moving away from us at high velocities appear 'red-shifted' and objects moving towards us appear 'blue-shifted'. This means that well known emission lines, like the hydrogen 21-cm signal, will appear to have a lower wavelength if the hydrogen gas cloud under observation is moving towards us and a higher wavelength if it is moving away from us. In 1927, Edwin Hubble discovered that all galaxies are moving away from us. He concluded this from the red shifted spectra he obtained from the galaxies under observation.

While observing galaxies for the purpose of plotting rotation curves, the whole galaxy will appear red-shifted as per Hubble's law, but the magnitude of red-shift will vary across the galaxy. The difference in red-shift is a function of the rotation velocity, modulated by inclination, position angle and surface brightness effects. This means that the observed redshift cannot be taken to be a direct function of the rotation velocity alone.

[1] A group of around 54 galaxies, including the Milky Way that form our immediate neighbourhood in the universe.

[2] Radio observations made by measuring the 'forbidden' ortho-para flip in hydrogen whose emission/absorption wavelength is 21cm. This falls into the radio region of the electromagnetic spectrum.

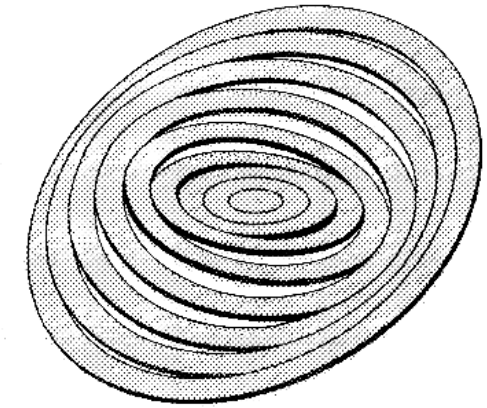

Figure 8.1: The tilted ring model as proposed by Rogstad et al. (1974). Representation inspired from the same paper.

8.1 Parameters

The five main parameters that directly or indirectly influence the radial velocity field that is observed by the telescope are:

1. (X_0,Y_0): The coordinates of the center of the galaxy.
2. V_{SYS}: The systemic velocity of the galaxy, i.e., the velocity with which the galaxy as a whole is moving away from the Sun.
3. V_{ROT}: The actual rotational velocity of the galaxy as a function of radius from the center.
4. PA: Position angle of the galaxy. This is defined as the angle, taken in the anticlockwise direction, between the north direction in the sky and the receding (higher red-shift) major axis of the galaxy.
5. $INCL$: Inclination of the galaxy with respect to the line of sight. Edge-on galaxies have a 90^o inclination and face-on galaxies have 0^o inclination by definition.

If $V(x, y)$ is the radial velocity field obtained directly from observation, we can write it as a function of the above five parameters as:

$$V(x, y) = V_{SYS} + V_{ROT}(R) \times \sin(INCL) \times \cos(\theta) \tag{8.1}$$

where θ is the azimuthal angle, defined using the other parameters.

$$\cos(\theta) = \frac{-(x - x_0)\sin(PA) + (y - y_0)\cos(PA)}{R} \tag{8.2}$$

$$\sin(\theta) = \frac{-(x - x_0)\cos(PA) - (y - y_0)\sin(PA)}{R\cos(INCL)} \tag{8.3}$$

Evidently, the above system of equations cannot be solved analytically. Moreover, the orientation parameters INCL, PA, X_0 and Y_0 cannot be determined in a straight forward way from the observed rotation velocity field $V(x, y)$. Hence, an iterative process called the tilted-ring fitting method is used, where the results improve with the number of runs.

8.2 Algorithm

The tilted ring model was introduced by Rogstad et al. (1974) and is very commonly used to analyse the kinematics of disk galaxies which might contain warps. The description of the algorithm given here is derived from Begeman (1987). The algorithm goes as follows:

1. Divide the galaxy into concentric rings. For effective sampling, consider at least two points per beam on the major axis.
2. Guess an initial value for all the five unknown parameters listed above, for each ring. This need not be a completely random selection. In mostly cases supporting optical data can give an estimate of these parameters, or they can be approximated from the moment zero map. Such a map can be easily generated using available softwares like Duchamp (Whiting , 2012).
3. From these initial estimates, generate a radial velocity profile using equation 8.1.
4. Perform a least squares fit for the parameters V_{ROT}, V_{SYS}, X_0, Y_0, PA and INCL using the observed $V(x, y)$ and the field generated in the previous step.
5. Take the improved values of these parameters as the input for the next step and repeat the above two steps till the accuracy reaches a satisfactory level.

This algorithm can be modified a little to make it more efficient. Observe that V_{SYS}, X_0 and Y_0 are global parameters for all the rings. Ideally, we do

not expect these values to change from ring to ring for any galaxy. Hence, we can first fit for these three parameters by averaging over all the rings and then perform the iterative process for the ring varying parameters. Note that you can also perform the iterative process separately for the approaching and receding sides of the galaxy, to account for asymmetry, if any.

The Tilted Ring Fitting Code (TiRiFiC) software that I have used for my project, implements the same algorithm in a slightly modified form. I have described the TiRiFiC algorithm and its working briefly in the next chapter.

Chapter 9

Tilted Ring Fitting Code- TiRiFiC

The tilted ring model is the standard kinematic model for galaxies. Since its introduction by Rogstad et al. (1974), a number of variations of the original algorithm have come up which try to improve the fitting process over the previous ones. All these algorithms try to analyse the velocity field that has been derived from the observed datacube. Such a method has two problems: (a) The first and more serious problem is that, deriving the velocity field is not always straightforward. In galaxies with large warps, even if the disk geometry is known, the line of sight may intersect the disk twice making it meaningless to assign a single representative velocity field at that point and (b) an intermediate step between the model and the observed data cube- the derived velocity field is used to improve the model, carrying the errors of the first step in to the subsequent fitting process.

The solution TiRiFiC suggests to the above two problems is generating a model datacube that can be optimised against the observed datacube directly and deriving the unknown parameters like position angle, inclination, surafce brightness and the rotation velocity field from this optimised model.

9.1 Modified Tilted Ring Algorithm

TiRiFiC uses a modified version of the tilted ring algorithm that was described in the previous chapter. It applies the algorithm to a datacube instead of the 2D galaxy. Much like the original algorithm, a TiRiFiC model is defined by a set of parameters that vary with radius. But in addition

to these, it also uses a set of global parameters that can change the model datacube. These parameters are:

1. Radius (RAD): An array that specifies the radius of each ring of the model. All the subsequent parameters have to be fixed at each of these radii for the initial run. The modified values are also reported back at each of these radii.
2. Circular velocity (VROT): An array that contains the rotational velocity at each of the above radii. After the fitting process, this array will contain the rotation curve (at discreet points though).
3. Scale height (Z0): The thickness of the galaxy which can be varied for each radii. In addition to this, another global parameter LTYPE can control how the density falls over this thickness.
4. Surface brightness (SBR): The original tilted ring algorithm did not have a provision to fit for the surface brightness. TiRiFiC allows you to fit the model's surface brightness as a function of radius.
5. Inclination (INCL): The array containing the inclination of each ring at the above specified radii.
6. Position angle (PA): Array containing the position angles.
7. Right ascension (XPOS): The RA value of the center of the ring. Though, intuitively, this must be a global parameter it is given the flexibility to vary with radius because for certain peculiar disk geometries, like a bowl shape, the center needs to be shifted for each ring.
8. Declination (YPOS): The DEC value of the center of each ring.
9. Systemic velocity (VSYS): The velocity with which the galaxy is moving away from the sun is also given the provision of varying with distance.

   The global parameters include:

10. CONDISP: Global isotropic velocity dispersion, which includes the instrumental dispersion. Though this parameters is known to vary with radius in spiral galaxies and is also not uniform in the vertical and horizontal directions, it has been made a global parameter to improve the computational speed. See, Scale height in the above list.
11. LTYPE: Vertical distribution of gas density. The user has a choice between Gaussian, $sech^2$, Lorentzian and box type layers.
12. CFLUX: Constant total flux of a single point source. The TiRiFiC model is generated by Monte-Carlo distributing small point sources

with this flux on the various rings and sub-rings. The number of point sources is approximately the total intergrated flux of the model galaxy divided by this number.

The model generated using the specified ring varying parameters at the given radii is an idealistic model that cannot be directly compared to the observed datacube. Practically, the observed datacube is affected by the finite beam-width of the telescope in the space domain and the instrumental dispersion in the velocity domain. To account for these affects, the model cube generated is convolved with a 3D-Gaussian which is a product of the 2D Gaussian in the $x-y$ plane (that is determined by the beam-width of the telescope) and a 1D Gaussian determined by the global velocity dispersion factor CONDISP.

The model thus generated with the above parameters is optimised by minimising the χ^2. The χ^2 is calculated using the deviation at each pixel, weighted by the RMS and quatization noise at that pixel.

$$\chi^2 = \sum_k \frac{(M_k - O_k)^2}{\sigma_k^2} \tag{9.1}$$

where the index k runs over all the pixels in the datacube. M_k is the value of the k^th pixel of the model datacube and O_k is the value of the corresponding pixel in the observed datacube. σ^2 is the noise of the observed datacube at that pixel. It is the sum of squares of the RMS noise and the quantization noise. The weight given to the quantization noise can be controlled by a separate "weight" parameter- W which can be fixed by the user.

$$w_k = \frac{\sigma_{rms}^2 \cdot W^2 + (\sigma_k^q)^2}{W^2} \tag{9.2}$$

The weight parameter gives the flexibility to emphasize on regions of high surface density or low surface density. With a higher weight parameter, the emphasis is more on regions of lower quantization noise i.e, regions of low surface density.

The χ^2 thus calculated is now minimized by changing the ring parameters. The Monte-Carlo technique is used to generate a model datacube and the golden-section search algorithm is used to find the local χ^2-minima in parameter space.

9.2 Model Generation: Monte Carlo Method

The Monte Carlo technique refers to the broad class of methods which use random sampling to obtain the numerical solutions. TiRiFiC uses this method to generate a model datacube.

A number of 'sub-rings' are created between the rings specified by the user. The user can control the number of sub-rings between two rings with a parameter called RADSEP. The parameter values at these sub-rings are radially interpolated from the values specified/calculated at the rings. As mentioned above, in the description of the parameter CFLUX, the generated model consists of a certain number of point sources of fixed intensity (as determined by the user). These points are distributed on the sub-rings using the Monte-Carlo technique to generate the final model datacube.

9.3 Fitting Algorithm: Golden-Section Search

The golden section search is an iterative process for finding the extremum of a piece-wise monotonically increasing or decreasing function. It is used in place of faster converging methods that use derivatives to save computational time and space. For a single variable function $f(x)$ the algorithm goes as follows:

1. Start with an interval [a,b] between which the extremum is known to exist.
2. Calculate $\phi = (-1 + \sqrt{5})/2$, which is the golden ratio. Compute $c = b + \phi(a - b)$ and $d = a + \phi(b - a)$.
3. Evaluate $f(x)$ at c and d.
4. The next interval is [d,b] if $f(c) < f(d)$ and [a,c] if $f(c) > f(d)$.
5. Repeat the steps using the new interval till a minimum required interval length is reached.

In summary, TiRiFiC uses the Tilted-Ring Model to fit for the rotation curve with the following steps:

1. Determine the initial set of ring varying parameters, global parameters and telescope beam widths.
2. Generate a model datacube with the specified number of sub-rings whose parameter values are linearly interpolated between the given rings.

3. Generate enough point sources with the flux intensity determined by the user, so that you can account for the entire integrated flux density of the observed galaxy.
4. Project the point sources on the generated sub-rings using a Monte Carlo technique.
5. Convolve the above model with a 3D Gaussian to account for observational effects like beam width and velocity dispersion.
6. Compare this model datacube with the observed datacube, pixel by pixel and compute the χ^2 deviation.
7. Minimise this deviation using the golden section search algorithm to change the parameter values. Repeat from step 2 with this new set of parameters.

In the next section I have discussed the results of fitting the dwarf galaxy NGC3741 using this improved algorithm.

Chapter 10

NGC 3741

The dwarf galaxy NGC 3741 is a well studied and documented galaxy. The rotation curve has been established with a large accuracy by Begum et al. (2005). My aim in re-fitting this velocity curve was to test the efficiency of TiRiFiC. I took a bottom-up approach to achieve this, moving from the lowest resolution to the highest, though I did not transfer the results of the lower fits to the higher ones. Each resolution has been fit from initial guess parameters obtained from Duchamp (Whiting , 2012).

10.1 Parameter estimation from Duchamp

Duchamp produces the moment-0 map of the galaxy and allows you to broadly estimate the input parameters of TiRiFiC like the surface brightness, inclination and rotation velocity. The details of this estimation are given below:

Radius Duchamp produces a map of the detected intensity in all channels. This allows you to determine the radial extent of the galaxy in arcsec. The resolution at which you need to sample the galaxy is determined by the beam width of the telescope. Typically, the resolution is taken as half the beam width, though with TiRiFiC I could not fit at any resolution finer than the beam width. So in this case, the width of the radial bins is the beam width of the telescope.

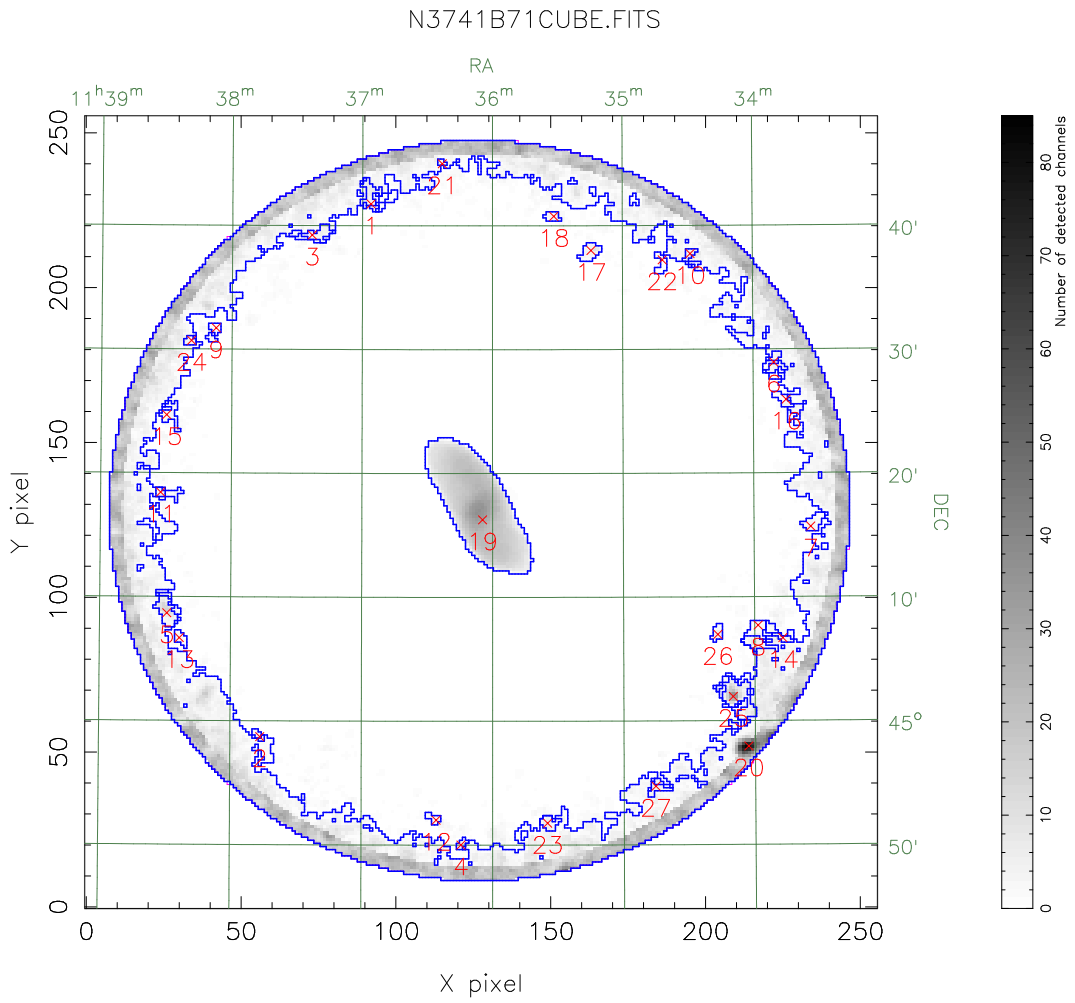

Figure 10.1: The detection map produced by TiRiFiC for the 71" resolution cube. The radial extent of the galaxy can be estimated from the legend as 300" and it has been sampled at the beam width for the purpose of fitting.

Systemic Velocity The red-shift of the center of the galaxy, or the average red-shift of the whole galaxy gives the systemic velocity. Duchamp gives a rough estimate for this if the third spectral axis of the input data cube is in velocity units. Hence if the Z pixels are given in any other units other than 'VEL', the datacube needs to be modified.

Rotation Velocity Duchamp provides two measures of the spectral width, one at 50% of the peak flux and another at 20% of the peak flux. These are measured on the integrated spectrum (i.e. the spectra of all detected spatial pixels summed together), and are defined by starting at the outer spectral extent of the object (the highest and lowest spectral values) and moving in or out until the required flux threshold is reached. The widths are then just the difference between the two values obtained. The rotation velocity is taken as half the width at 20% of the peak flux.

Inclination The tilted-ring model assumes that galaxies are circular disks. This means that the observed elliptical shape of the galaxy is entirely due to projection effects. Neglecting the contribution of the finite thickness of the galaxy to this ellipticity, we can get the ideal inclination.

The observed major axis would be the actual radius of the circular galaxy and the inclination towards or away from the line of sight would produce the observed minor axis. Simple geometry gives, θ the inclination angle as:

$$\theta = cos^{-1}(\frac{MIN}{MAJ}) \tag{10.1}$$

Surface Brightness Duchamp gives the integrated flux over all the spatial and spectral channels. Surface brightness is the intergrated flux per unit area observed. Since we observe the galaxy as an ellipse, the surface brightness can be approximated as the integrated flux over the area of the observed ellipse.

Position Angle This is estimated by 'eye' from the detection map given by Duchamp. For instance, from fig 10.1 we can 'see' that the position angle is roughly around 45^o for this galaxy. By definition, position angle is the number of degrees east of north that the galaxy appears tilted towards. The direction of north is fixed by increasing declination (+y axis in the figure), and direction of east by increasing right ascension (-x axis in the figure).

Center of the galaxy The spatial coordinates of the center of the galaxy are directly reported by Duchamp and can be taken as such.

Scale Height Duchamp does not provide any estimate of this parameter and infering it from the detection map is not straight-forward either. It can be estimated in other ways like solving the vertical hydro-static equilibrium equation (Patra et al., 2014), but for the purpose of this project I used a random guess.

10.2 Results

Resolution of the observation is determined by the beam width of the telescope used to observe the galaxy. In this case beam widths of 71", 40", 20" and 10" have been used. The radial extent of the galaxy keeps decreasing with resolution due to the decreasing area covered by the telescope beam. This effectively means that higher resolution fits can only be performed within the inner rings and only low resolutions fits are available for the larger radii. The coming sections show the results of the fit for various resolutions.

10.2.1 70" Resolution

A radial extent of 300" and a 71" beam width gave only 5 bins across the radius of the galaxy. In the first run, I fit for the average values of all the parameters to account for differences in systemic velocity, coordinates of the center of the galalxy, scale height and other parameters than are uniform across the galaxy, between the estimated values of Duchamp and the tilted ring model of TiRiFiC.

In the second run, I used a coarse fit- sampling at only double the beam width; and a full fit process in the third run. Subsequent runs at half the beam width did not yield any physically meaningful results. Figure 10.2 gives the results of the final run. The inclination and position angle show the presence of a warp just within the Holmberg radius which is 100″ for NGC3741.

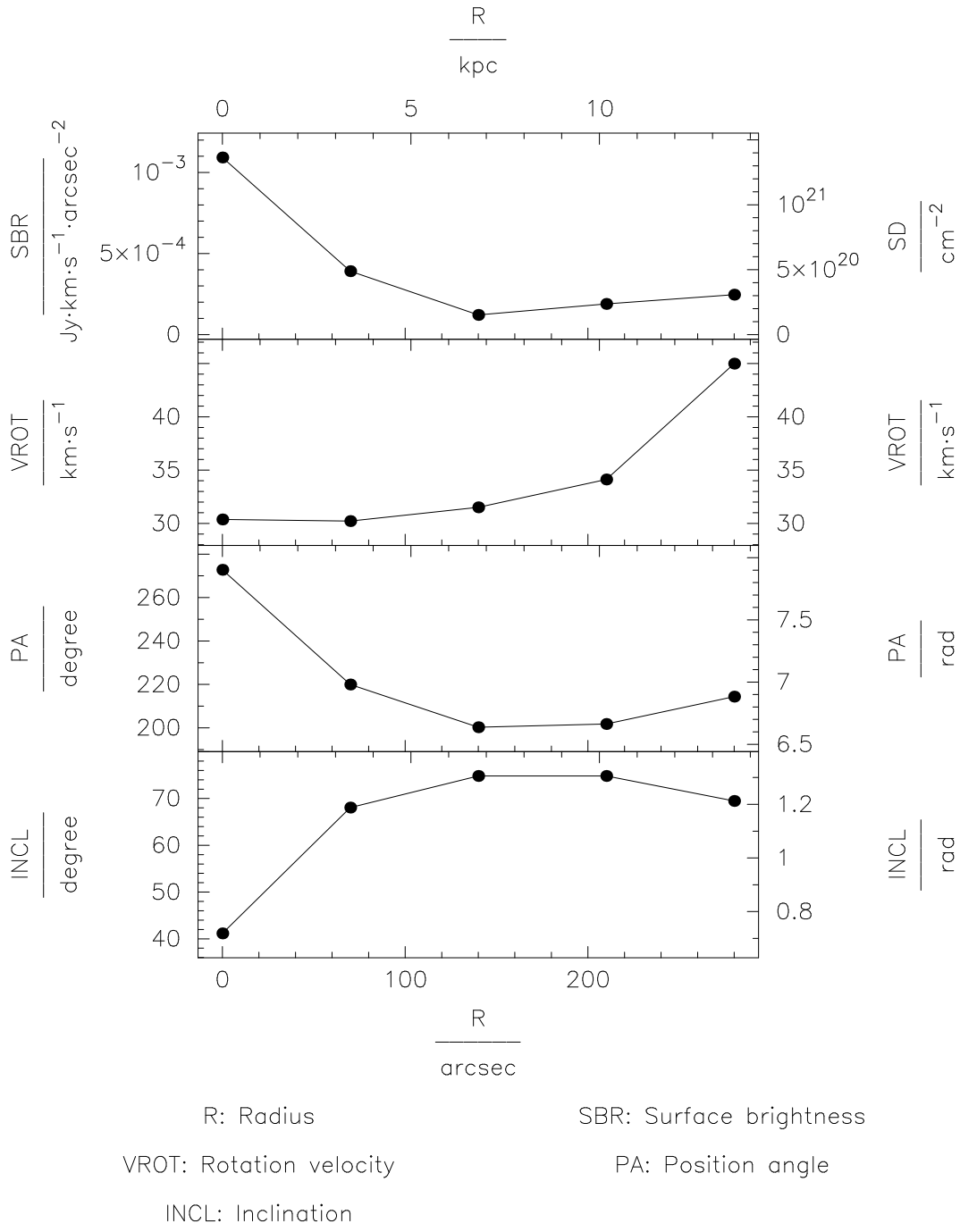

Figure 10.2: Results of the TiRiFiC fit process for a resolution of 71″. Rings were considered at a separation of the beam width at 0”, 70”, 140”, 210” and 280”.

10.2.2 40" Resolution

A similar process, as described for the 71" resolution was adopted for this resolution as well. The same galactic extent is same as that of the previous resolution due to the smaller bin size. A bin at 280" in the previous resolution meant a radial extent of 315" whereas the same bin at this resolution would mean a radial extent of 300". A total of 8 bins have been considered across the profile. Results are shown in 10.3. The wrap can be seen more prominently at 40" this time.

10.2.3 20" Resolution

The radius of the galaxy was considered upto 200" only this time. Though the inclination appears awry, notice that the fluctuation is only between a range of 10^o which coincides with the accepted range given by Begum et al. (2005). The clear warp present in the previous resolutions is not visible anymore. This is because the tilted ring model can distinguish between rotation velocity and inclination only to a certain extent[1]. The rest of the curves show a smooth trend that is similar to the previous resolution results.

10.2.4 10" Resolution

From the rules for warps given by (Briggs, 1990), we know that galaxies are mostly planar upto R_{25} which is $30''$ for NGC3741. Keeping this in mind, the position angle and inclination were forced to remain constant upto 10". Moreover at this resolution, TiRiFiC was unable to yield meaningful results while fitting at the beam width itself. The fit was made coarser to twice the beam width. The extent of the galaxy was further reduced to 150" in this case, with 7 bins across the profile. To take advantage of the higher resolution of this file, though forced to fit at twice the beam width, complementary nodes have been considered for surface brightness, rotation velocity and the geometrical parameters. The results of the final fit are given in 10.4. Even at this lower resolution inclination does not show a smooth trend though the variation is still within the acceptable range given by Begum et al. (2005).

[1]For the χ^2 maps as a function of rotation velocity and inclination, see Fig 2 of Begeman (1987).

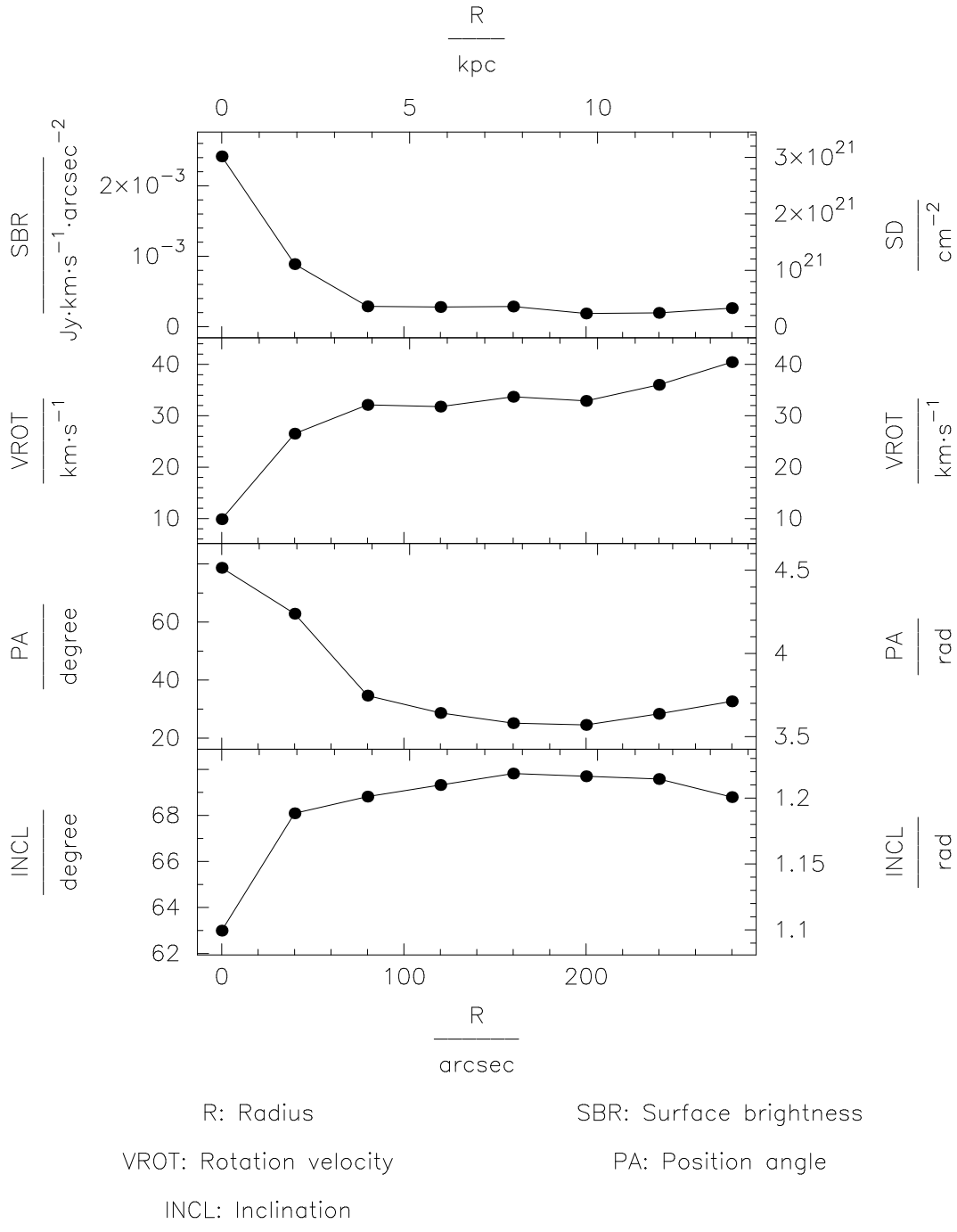

Figure 10.3: Results of the TiRiFiC fit process for the 40" resolution. Rings were considered at intervals of 40 arcsec up till an extent of 280".

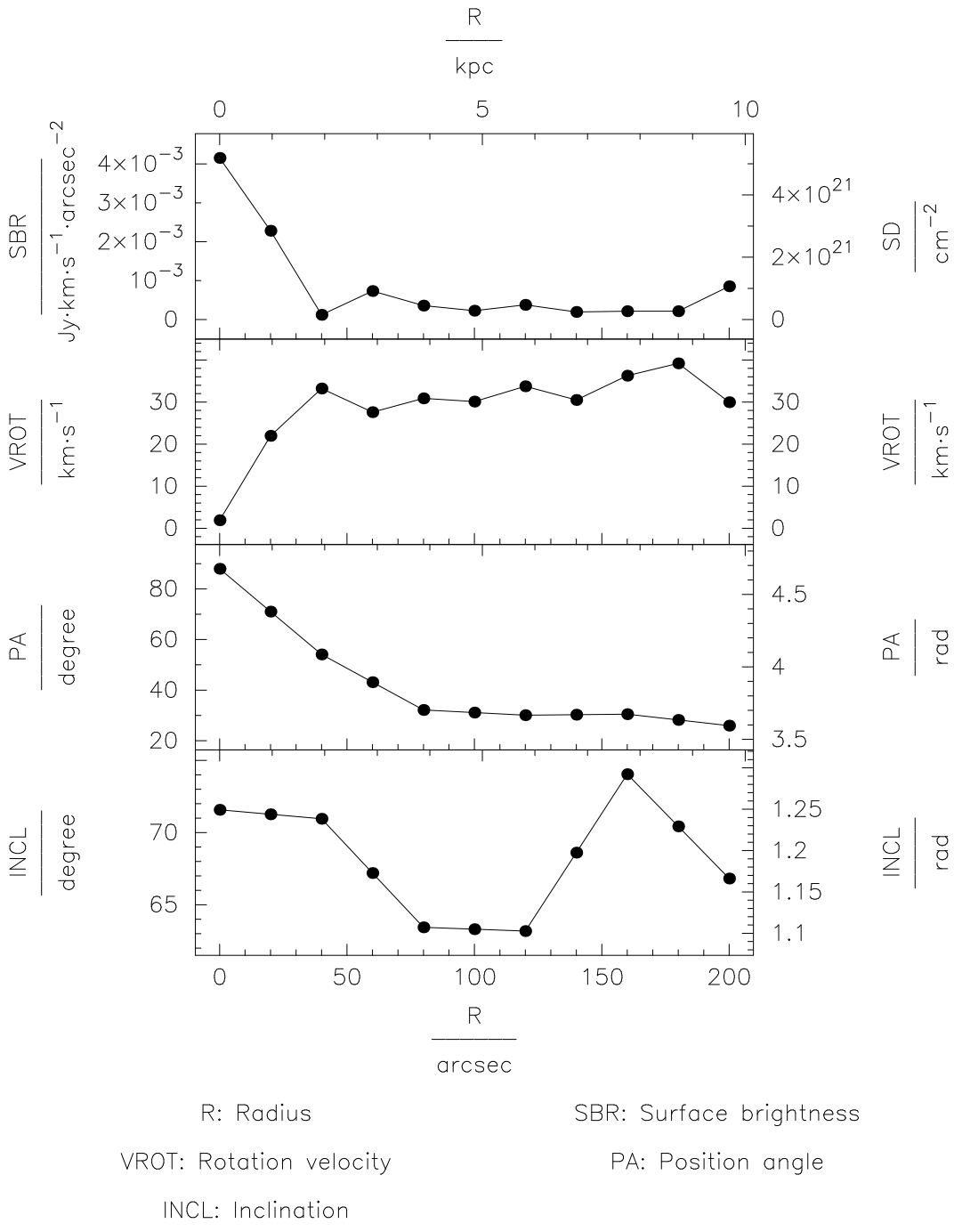

Figure 10.4: Results of the TiRiFiC fit process for the 20" resolution. Rings were taken at intervals of 20 arcsec up till an extent of 200".

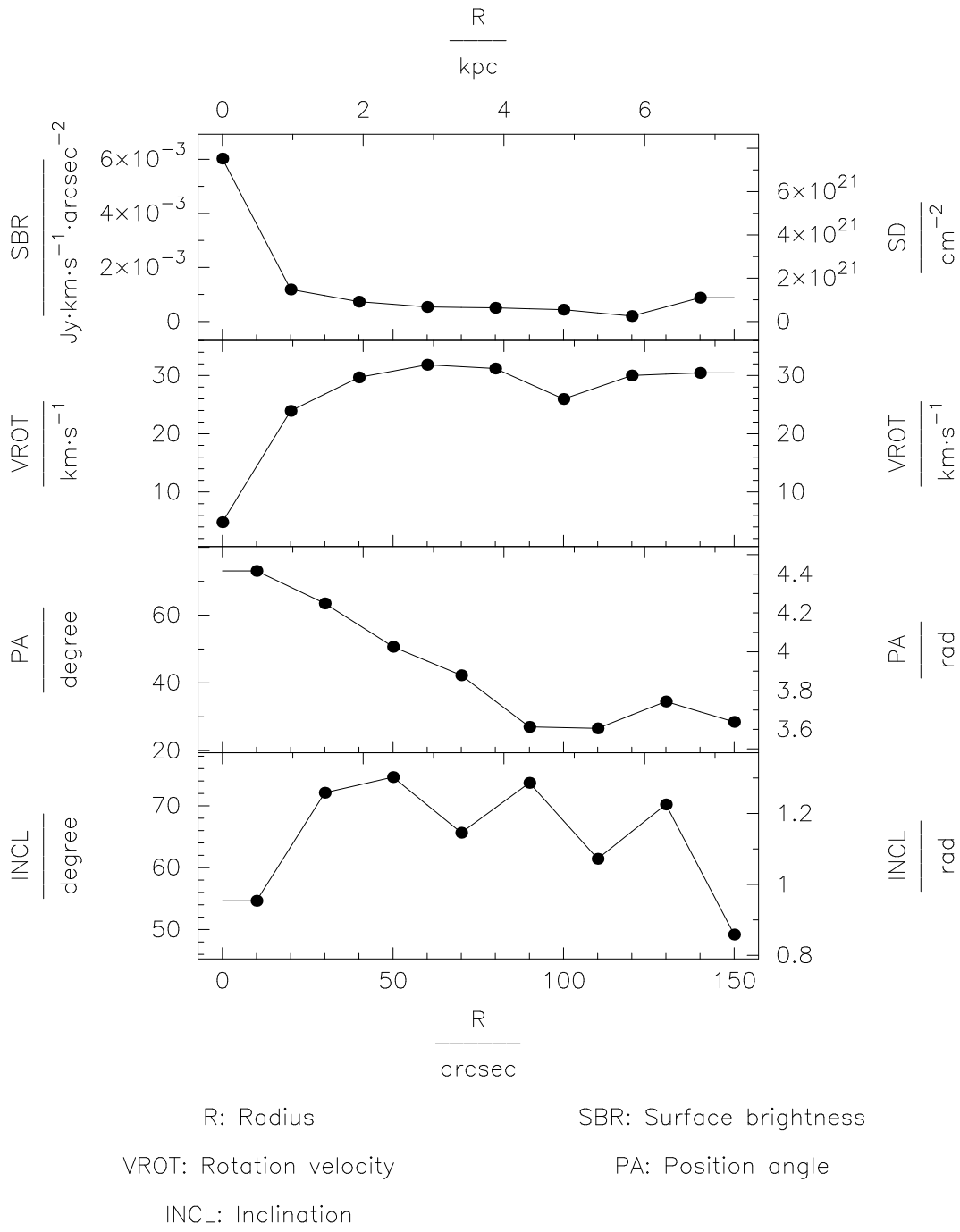

Figure 10.5: Results of the TiRiFiC fit process at 10" resolution. Rings were taken at intervals of 20 arcsec (double the beam width) up till an extent of 150". Note that the rings considered for SBR and VROT and complementary to the rings considered for fitting PA and INCL. This has been done to take advantage of the higher resolution of this data cube.

Chapter 11

Conclusion

The rotation velocity is found to increase throughout the extent of the galaxy. The flattening starts from around 40″ which marks the optical extent (R_{25}) of the galaxy. The increase in velocity after 200" was not present in the rotation curve of Begum et al. (2005) and is probably a result of the fitting process.

The surface brightness profiles for all resolutions do not show much deviation except the edge of the 10 arcsec and 20 arcsec plots. Care was taken to ensure that the galaxy extent does not overshoot half the detected beam (taking into account the finite beam size). In future fits using TiRiFiC, probably a whole beam width at the edge of the galaxy should be ignored.

Position Angle at all the resolutions shows a similar trend. This might be due to a central bar in NGC 3741. The inclination maps on the other hand, do not show any clear trend though the range is restricted between 55^o and 75^o. This haphazardness could be due to the large scale height of the galaxy, but this is merely speculation and has not been verified.

Comparision of the obtained results with published rotation curves of NGC3741 show that TiRiFiC does not perform as well as available standard tilted-ring fitting softwares, at least in the case of this galaxy. The reason for the poor performance might be the thickness of the galaxy (which TiRiFiC predicts is large), but this could not be verified due to the time constraint. It is possible that due to this or some other reason, TiRiFiC does not give good results for this galaxy alone but is efficient in fitting other galaxies.

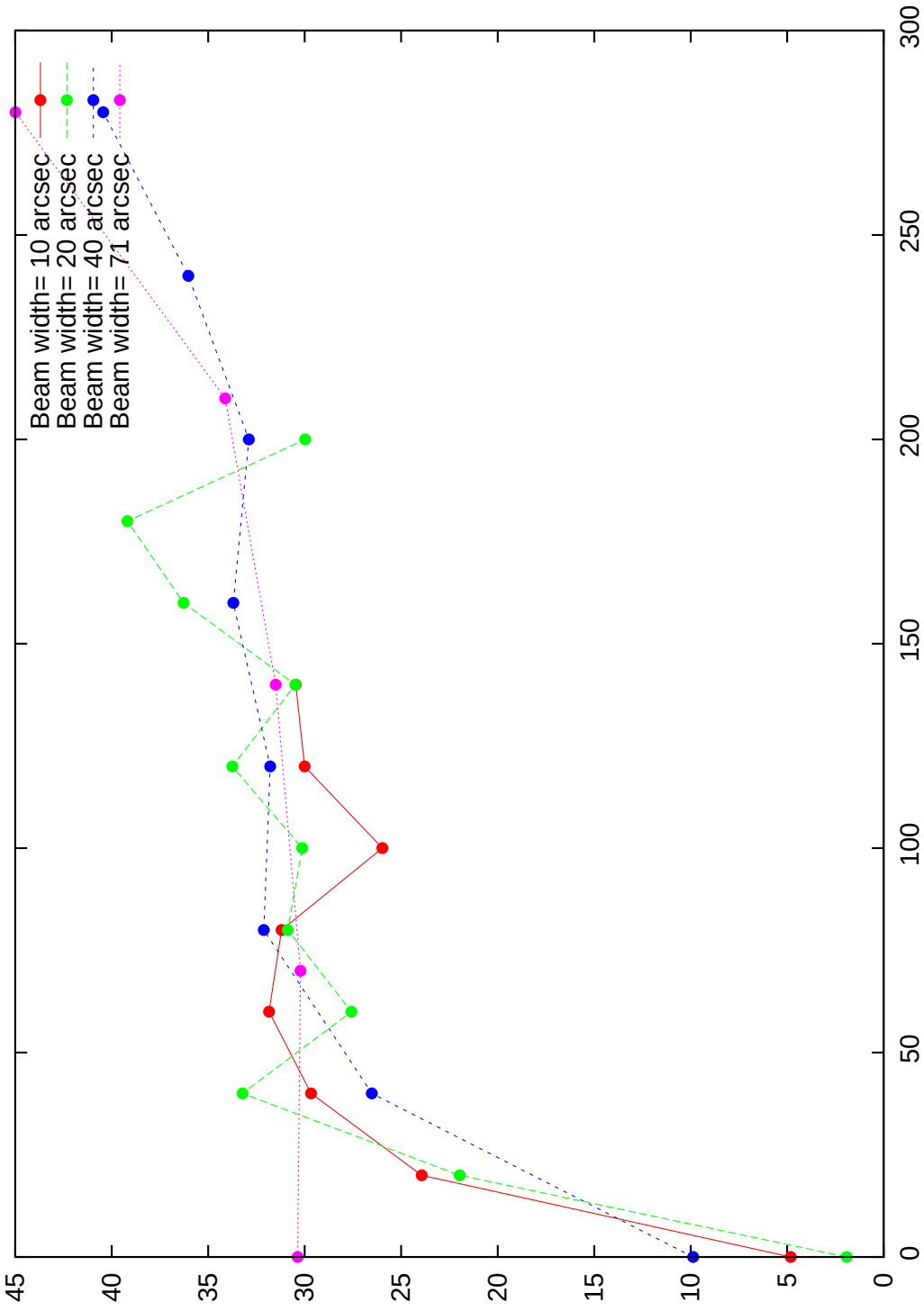

Figure 11.1: Comparision of the rotation velocity curves at various resolutions.

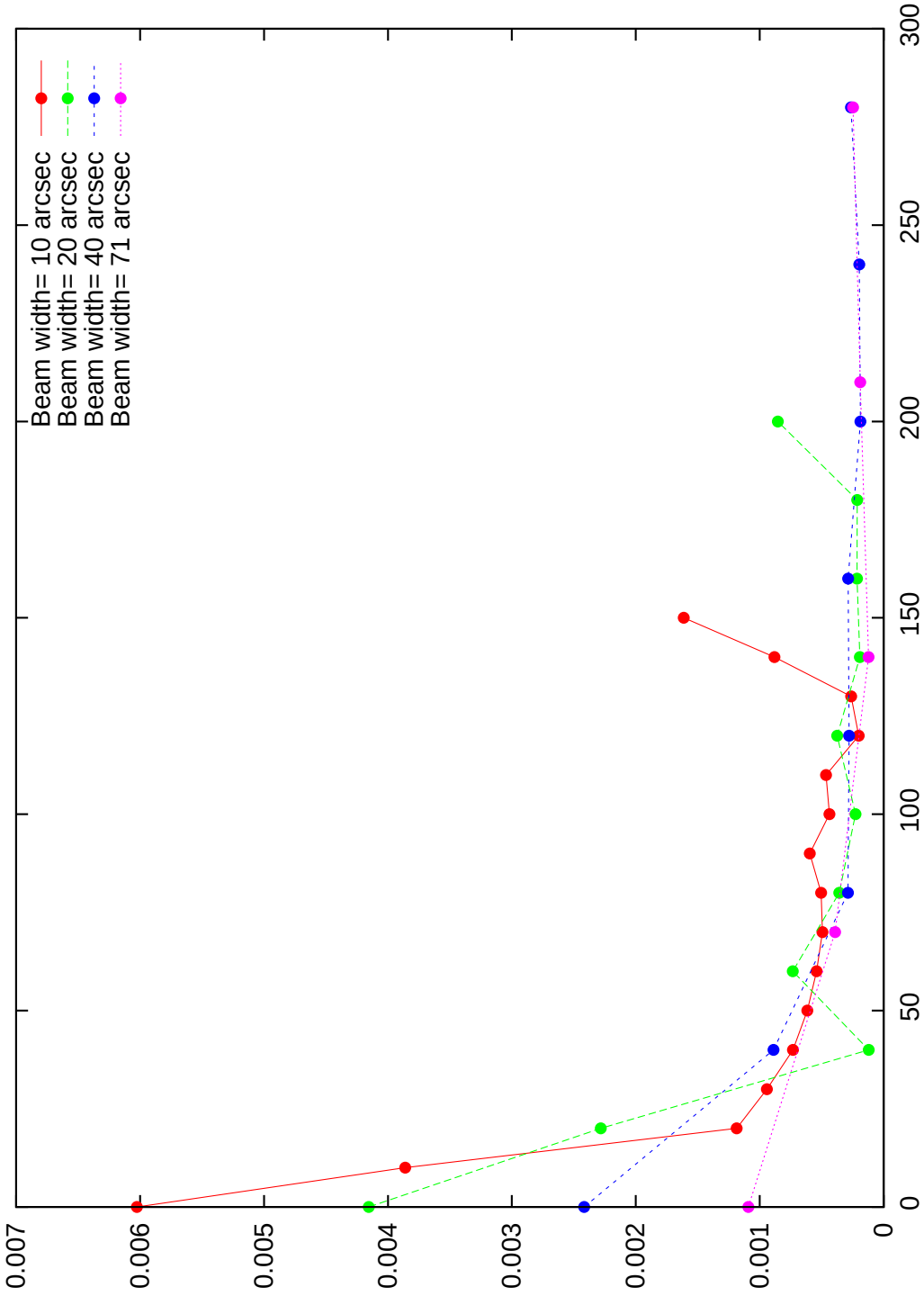

Figure 11.2: The velocity rotation curves obtained at various resolutions.

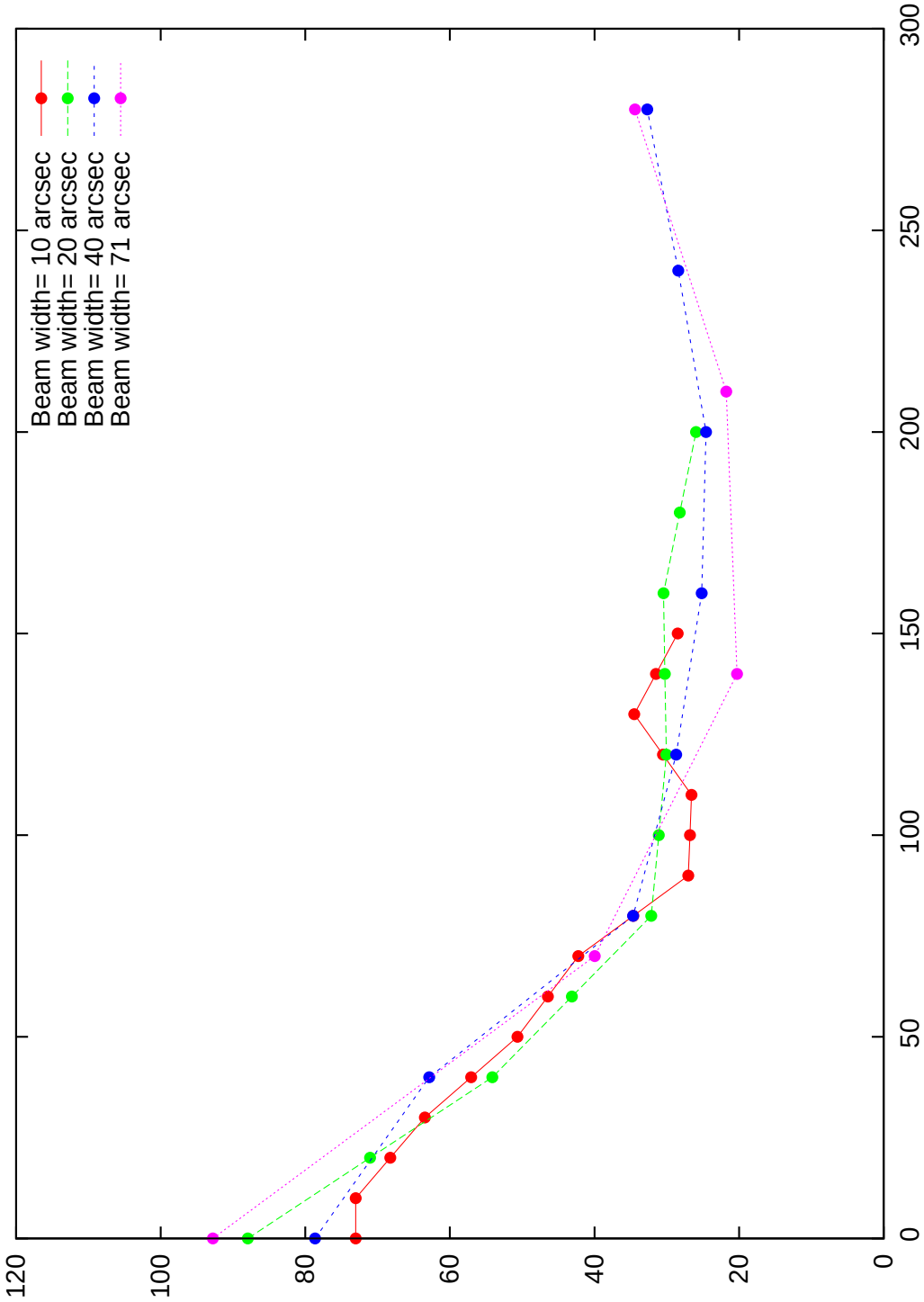

Figure 11.3: Variation of position angle with respect to radius at the available resolutions.

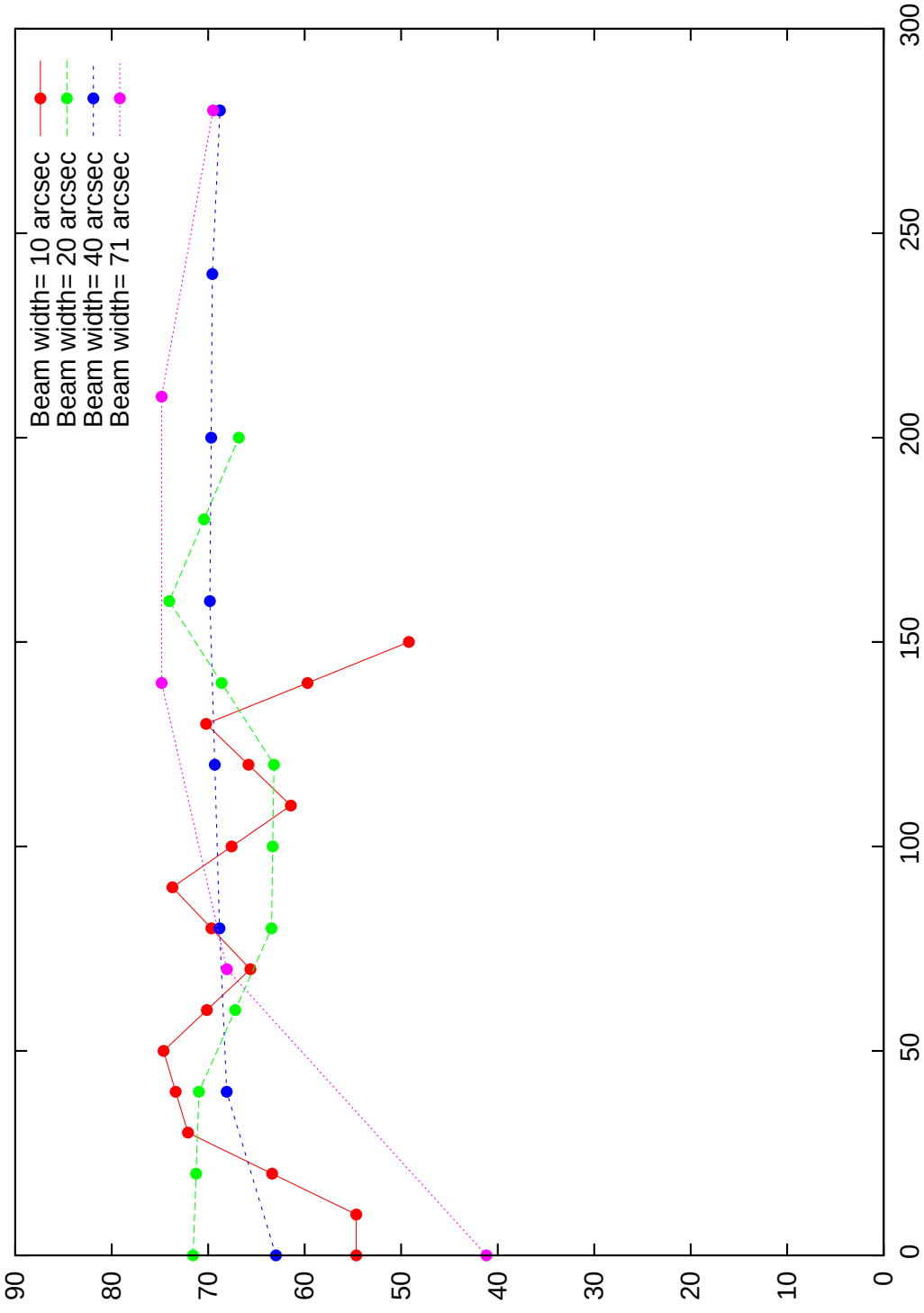

Figure 11.4: Variation of inclination with respect to radius at the available resolutions.

Bibliography

Józsa, G. I. G., Kenn, F., Oosterloo, T. A., & Klein, U. 2012, Astrophysics Source Code Library, 1208.008

Begeman, K. G. 1987, Ph.D. Thesis, Kapteyn Institute

Gentile, G., Salucci, P., Klein, U., & Granato, G. L. 2007, MNRAS, 375, 199

Begum, A., Chengalur, J. N., & Karachentsev, I. D. 2005, AAP, 433, L1

Whiting, M. T. (2012), DUCHAMP: a 3D source finder for spectral-line data. Monthly Notices of the Royal Astronomical Society, 421: 3242-32256. doi: 10.1111/j.1365-2966.2012.20548.x

Rubin, V. C., Ford, W. K. J., & . Thonnard, N. 1980, APJ, 238, 471

Jet Propulsion Laboratory. "New Recipe For Dwarf Galaxies: Start With Leftover Gas." ScienceDaily. ScienceDaily, 19 February 2009. ¡www.sciencedaily.com/releases/2009/02/090218132145.htm¿.

Rogstad, D. H., Lockhart, I. A., & Wright, M. C. H. 1974, APJ, 193, 309

Prokais, J. G., & Manolakis, D. G. 2007, Digital Signal Processing: Principles, Algorithms and Applications

Patra, N. N., Banerjee, A., Chengalur, J. N., & Begum, A. 2014, MNRAS, 445, 1424

Briggs, F. H. 1990, APJ, 352, 15