DEVELOPMENT OF A SYMBOLIC COMPUTER ALGEBRA TOOLBOX FOR 2D FOURIER TRANSFORMS IN POLAR COORDINATES

Edem Dovlo

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfilment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

in Mechanical Engineering
Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
University of Ottawa
Ottawa, Canada

April 2011

Abstract

The Fourier transform is one of the most useful tools in science and engineering and can be expanded to multi-dimensions and curvilinear coordinates. Multidimensional Fourier transforms are widely used in image processing, tomographic reconstructions and in fact any application that requires a multidimensional convolution. By examining a function in the frequency domain, additional information and insights may be obtained.

In this thesis, the development of a symbolic computer algebra toolbox to compute two dimensional Fourier transforms in polar coordinates is discussed. Among the many operations implemented in this toolbox are different types of convolutions and procedures that allow for managing the toolbox effectively. The implementation of the two dimensional Fourier transform in polar coordinates within the toolbox is shown to be a combination of two significantly simpler transforms. The toolbox is also tested throughout the thesis to verify its capabilities.

Acknowledgements

I am truly grateful for my supervisor, Dr. Natalie Baddour who has been very supportive throughout my Master's degree. Thank you for all your encouragement and guidance in helping me comprehend the subject matter and giving me the room to learn and be innovative. You always did ask great questions and provided helpful advice. I enjoyed the laughs we shared at our weekly meetings. Thank you for the financial assistance as well. It would have been much more difficult to complete the program without it.

I would also like to thank my parents, brother, family and friends who cheered me on and offered to help in whichever way they could. Thank you for taking an interest in what I was researching and wanting to know how I was really doing. You are all so great and I appreciate all the support.

Last but not least, thanks to the guys in the CBY office I had the opportunity to share space with. It has been fun!

TABLE OF CONTENTS

Table of Figures

Table of tables

Nomenclature

r, θ	radial and angular variables in space domain (polar coordinates)
ρ, ψ	radial and angular variables in frequency domain (polar coordinates)
$\widehat{F}_n(\rho)$	*n*th order Hankel transform
$J_n(z)$	*n*th order Bessel function
$\mathbb{F}_S$	forward Fourier series *transform*
$\mathbb{F}_S^{-1}$	inverse Fourier series *transform*
f_n	complex Fourier series coefficients
A_n, B_n	real Fourier series coefficients
$f\left(\vec{r}\right)$	original function to be transformed
$F\left(\vec{\omega}\right)$	2D Fourier transform
δ_{nm}	Kronecker delta function
$\delta(r)$	Dirac-delta function
$S_n^k\left(u, r, r_0\right)$	shift-type operator
$*$	one dimensional convolution
$**$	two dimensional convolution
$*_\theta$	angular/circular convolution
$*_r$	radial convolution

Chapter 1 : Introduction

1.1 Background

It is well known that the Fourier transform has proven invaluable in a wide range of disciplines such as engineering, mathematics, physics and chemistry. As will be discussed in greater detail later, the Fourier transform is applied in several areas such as communications, optics, astronomy, geology, image processing and signal processing.

The Fourier transform possesses a standard toolset comprising results for Dirac-delta functions, complex exponentials, scaling, translation, multiplication and convolution. The basic transforms of Dirac-delta functions and complex exponentials form critical foundations for the derivation of the shift, multiplication and convolution results. Extending the Fourier transform to multiple dimensions is straight-forward but it is the toolset of operational properties that are useful and serve as standard rules that

simplify the calculation of more complicated transforms. These rules for the Fourier transform are well known in single and multiple dimensions [1].

In 2D, developing the Fourier transform in polar coordinates [2] rather than the traditional Cartesian coordinates is not only known but often preferred when the function to be transformed is naturally describable in polar coordinates. This has been applied in photoacoustics [3] and writing the discrete form of these ideas by developing numerical algorithms for such calculations [4] from the continuous domain has been attempted. Recently, however a detailed development of the Fourier operational tool-set of Dirac-delta, exponential, spatial shift, multiplication and convolution for the 2D Fourier transform in polar coordinates has been done [5]. The treatment of the shift, multiplication and convolution theorems is rather interesting as they can also be adapted for the special cases of circularly symmetric functions that have no angular dependence.

Until then, the Fourier transform of the Dirac-delta function were known in polar and spherical polar coordinates but the transform rules for shift, multiplication and (particularly) convolution were incomplete at best. For instance, it was known that a two dimensional Fourier transform of radially symmetric functions is a Hankel transform of zeroth order but a multiplication/convolution rule for the Hankel transform, that has proven so useful in Cartesian coordinates, had not been found for polar coordinates.

The multiplication/convolution rule for the curvilinear version of the transform shows that the Hankel transform does obey a multiplication/convolution rule once the proper interpretation of convolution is applied. The proper definition of convolution along with its correct interpretation in curvilinear coordinates allows for the standard multiplication/convolution rule to be applicable once again.

Prior work in symbolic computation has been in the two areas; the actual algorithms in the computer algebra software and how they can be made faster, and the application of symbolic algebra to problems. This work endeavors to close that knowledge gap between the algorithms and the applications by creating the necessary tools (building on the algorithms) to apply in the problems.

1.2 Motivation

The two dimensional Fourier transform of functions that are best described in polar coordinates can be written in polar coordinates as a combination of Hankel transforms and Fourier series. The function need not be circularly symmetric for this to be true and applicable. Fourier transforms are very useful in Cartesian coordinates and for them to be just as useful in polar coordinates, a polar version of the Fourier operational toolset is required for the standard operations of scale, shift, multiplication, convolution etc. Being able to successfully compute the Fourier transform (2D in polar coordinates) and its associated toolset with computer software to efficiently obtain closed form results will allow for the solution of several problems.

1.2.1 Introduction

The two dimensional Fourier transform in polar coordinates and Fourier toolset of operational properties that have been developed analytically now need to be implemented in software in a way that makes them accessible for the purpose of modeling and analysis. A detailed account of the development of the governing equations and toolset shown in Chapter 3 can be found in [5]. The concepts and methodologies used in the CAS implementation are the contributions discussed in this thesis.

This thesis focuses on the development of a symbolic computer algebra toolbox to compute two dimensional Fourier transforms in polar coordinates. The transforms implemented in the toolbox include the forward *n*th order Hankel transform and its inverse, the forward and inverse one dimensional Fourier series transform and finally, the two dimensional Fourier transform in polar coordinates. There are other operations implemented alongside these to help manage the toolbox.

A modular approach is used here along with the idea of *lookup tables* to help avoid the issue of indeterminate results when attempting to directly evaluate the transform. This concept helps prevent unnecessary computation of already known transforms thereby saving on memory space and processing time.

1.3 Overview of the thesis

This thesis stems from an application in photoacoustic tomography. While performing analysis on this subject, the need arose to solve convoluted expressions exactly and analytically using integral transforms. Symbolic computer algebra permits analytical and exact calculations, however the lack of existing tools to easily implement 2D Fourier transforms in polar coordinates made simulations cumbersome. Therefore, it was decided to create a symbolic computer algebra toolbox to simplify these simulations and convolutions. The computer algebra system used in this research is Maple.

Symbolic Computer Algebra produces exact results unlike numeric arithmetic that normally generates approximate results or often suffers loss of accuracy or inability to converge. It is often said that symbolic computations are slow but the creation of faster algorithms and toolboxes for specific applications helps to address these shortcomings. Numerical methods may be better known but the fact still remains that some problems are best solved using symbolic means.

The toolbox developed herein is called the *SCAToolbox*. It is a package in which an archive is created to store the operations and procedures used in the package as well as integral transforms. There are many operations executed in this toolbox including different types of convolutions such as the one and two dimensional Cartesian convolutions, two dimensional polar convolution, and procedures that help with the effective administration of the toolbox. The two dimensional Fourier transform in polar coordinates implemented within the toolbox is shown to be a combination of the *n*th order Hankel transform, the one dimensional Fourier series and its inverse.
Figure 1 shows the code structure of the *SCAToolbox*. It gives a picture of the breakdown of the work done.

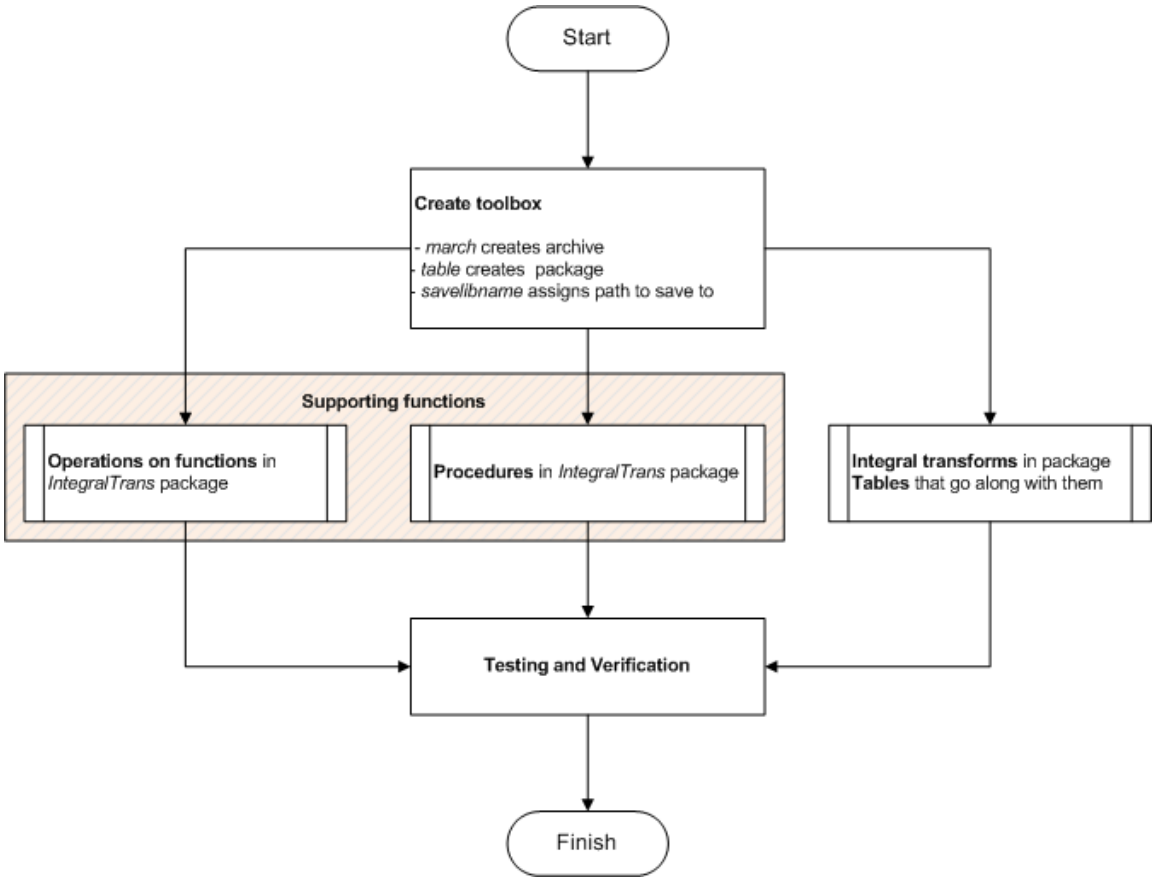


Figure 1: Overall outline of the toolbox: Maple code structure

1.3.1 Outline of the thesis

The second chapter of this thesis looks into past literature and work giving the history of Computer Algebra Systems (CAS) and how they have been applied, in particular with Fourier transforms.

The sections are broken up in groupings to help the reader better understand the logic and format of this work. This correlates with the code in Maple to allow for lucidity and readability.

Chapter 3 introduces the various equations governing this body of work. Here, the theory behind all the analysis and programming is discussed in detail to ensure that a

good mathematical background is set. This outlines what is implemented with CAS in the thesis.

The CAS software utilized in this work is discussed in the fourth chapter and since one of the main goals of this work is building a toolbox, this chapter addresses this. In this chapter, the constituents of a toolbox are discussed and a step-by-step description of the toolbox-building process as it pertains to the Maple software is given. This is by no means limited to the Maple software and as such can be generalized to other CAS software with syntactic modifications. This chapter also discusses the design criteria for ensuring the proper functioning of the toolbox as well as the architecture design of the toolbox summarizing the contents of the SCAToolbox.

Chapter 5 analyzes the supporting functions that are necessary for managing the procedures in the sub-package, *IntegralTrans,* within SCAToolbox. The supporting functions discussed consist of two types; one type being operations that can be performed on functions and the other being procedures for manipulating tables and procedures in this package. The different functions that can be utilized by the toolbox and the *operations* performed on them are discussed. These operations can be used outside the toolbox without loading the entire toolbox first, thereby saving on processing time and memory space. The toolbox also contains *procedures* necessary for the construction of and manipulations in the *IntegralTrans* subpackage. These procedures are inherent to the toolbox and cannot be accessed outside the toolbox, i.e. to be able to use them one must load the toolbox first. A summary of the supporting functions is given in Table 2 at the end of the chapter.

In the chapter that follows, the *integral transforms* and their corresponding *tables* that make up the core subpackage of the toolbox are analyzed. This is the main part of the package and as such is discussed in great detail. The idea of *mapping* is discussed and the tables help implement and use that idea. In Maple, the integral transforms are implemented as procedures which call on *lookup* tables where need be. The integral transforms discussed here include the forward and inverse Hankel transform of *n*th order, the forward and inverse one dimensional Fourier series transform and the forward and

inverse two dimensional Fourier transform in polar coordinates (implemented directly and by the use of the two other integral transforms listed above). A modular approach is employed in the Fourier transform. Chapter 6 also consists of a section that tests and verifies the implementation of these integral transforms.

Chapter 7 then analyzes rules that make up the Fourier operational toolset such as rules for Dirac-delta functions, complex exponentials, scaling, shifting and convolution-multiplication that are implemented in the SCAToolbox. Several examples are utilized in the testing and verification of these rules and are briefly discussed to show the effective execution and measure its accuracy.

The conclusion of this thesis is provided in the eighth chapter as well as some recommendations for future work. Finally, some appendices are included that provide more information that could not be included in the main body of the thesis. Appendix A contains some Maple concepts and commands used in this body of work. The intricacies of constructing a toolbox are made available in Appendix B. Links to the Maple code and sections of the code itself are provided in Appendix C.

Chapter 2 : Literature Review

2.1 History of CAS

Using computers for symbolic computation dates back to over a century and a half ago in 1834 when Charles Babbage conceived the *Analytical Engine*, a general-purpose machine that has features of the programmed digital computer inherent to its design. In 1842, Luigi Federico Menabrea (who later became prime minister of Italy) published the first description of the functional organization and mathematical operation of this engine as well as the first published computer programs. This paper was translated with added annotations providing further insight into Babbage's proposed engine by Augusta Ada King, Countess of Lovelace and daughter of Lord Byron a year later [6].

The 1960s was a period when Computer Algebra Systems (CAS) began to gain popularity evolving from theoretical physics and research in artificial intelligence. In 1963, Martin Veltman (a theoretical physicist who was later awarded the Nobel Prize) designed a program for symbolic manipulation of mathematical equations called *Schoonschip,* which means "clean ship" in Dutch, now considered the first computer

algebra system. In 1964, Carl Engelman created MATHLAB while doing research in artificial intelligence. In 1966, the first two conferences on the subject of computer algebra were held: ACM Symposium of Symbolic and Algebraic Manipulation in Washington, D.C. and the IFIP working conference on Symbol Manipulation Languages and Techniques in Pisa [7].

Reduce (started in 1960s by Anthony Hearn), muMATH (the Soft Warehouse, late 1970s and early eighties), Derive (the Soft Warehouse, 1988) and Macsyma (Joel Moses, 1979, Mathlab group, 1983, Richard Fateman, 1989) [8] were the first popular computer algebra systems. In recent times, the most popular computer algebra systems commonly used by research mathematicians, scientists and engineers include Mathematica and Maple. In this thesis, the CAS software used is Maple.

Symbolic computation allows a wider range of expression (for mathematical formulae and their various transformation rules) while computer algebra admits greater algorithmic precision as it constructs algorithms for computing algebraic quantities in various arithmetic domains, possibly involving indeterminates. Putting these two ideas together is important to help define algebraic domains for wider classes of symbolic expressions [9].

Maple is an interactive system for algebraic computation started at Waterloo in 1980. Compactness and portability are the most distinctive features Maple looks to achieve. The basic computer requirements per user are a few hundred kilobytes compared to the few thousand kilobytes usually needed for Macsyma or Reduce. It is designed to be portable to a variety of different operating systems and utilizes a library of functions to incorporate an extensive set of mathematical knowledge [10].

2.2 Review of CAS applications

From what will be discussed in this section, the emergence of a trend is noticeable. It is clear from the literature that the early years of using CAS for symbolic computations (prior to and including 1990) saw them mostly used in research (making in-roads into engineering- computer, software and electrical) and in areas in physics.

Thomas Beth attempted to construct fast arithmetic to compute a large class of problems associated with applications in communication systems, particularly in digital signal processing, coding and cryptography. In his paper, he showed in detail how to achieve his goal of using symbolic algebra (a suitable fit for these application) and claims that the techniques could be applied in the construction of devices for real-time application in communication systems [11]. In digital signal processing, there often arises the need to compute a convolution in group algebra. For instance obtaining fast multiplication schemes for large numbers [12] and in digital filtering for Finite Impulse Response (FIR) systems [13], [14]. Here, convolutions in data algebra become less-complex component-wise products in spectral algebra as will be seen later in this work as well. Changing domains is made possible by the existence of algorithms that are fast enough to compute the discrete Fourier transform and its inverse. Coding in this case refers to error control coding for which most of the algorithms, though inherently numerical, are seeing increasing importance for finite fields due to a wide range of applications of algebraic coding theory - therefore requiring the development and comprehension of symbolic and algebraic calculations [15].

Scott and Fee [16], researchers who are part of the Maple Symbolic Computation Group, also take a look at some of the projects associated with the group to illustrate what has and can be been done with CAS software. These projects spread over a wide range of areas such as Quantum theory, Relativity, Audio Engineering and an application of Fluid and Magneto-dynamics. In the area of quantum theory, Scott and Fee mention a new method for solving quantum mechanical problems by perturbation theory. This is in regards to work that has been presented by Scott, Moore, Monagan, Fee and Vrscay that shows the applicability and usefulness of symbolic computation (in this case, Maple) in providing tools for solving quantum mechanical problems by perturbation theory. This provides a sufficiently complex problem such that the methods used can be applied to similar complex problems and possesses exact solutions so that validation of any approximations is possible. Some more complex systems to which this method can be applied include atomic systems in the alkali system sequence ranging from the (light) lithium atomic system to as heavy a system as radium II. [17]

Scott and Fee also mention the extensive use of symbolic computation in solving some non-relativistic quantum mechanical problems as illustrated in Vinette and Cizek's work [18]. In their work, they emphasize how useful symbolic computation is as a new tool for obtaining numerical solutions along with symbolically solving mathematical problems. They used Maple to manipulate expressions symbolically in rational arithmetic expressions with unevaluated elements and to perform complex and tedious operations quickly using simple commands [18]. The paper also delves into the possibility of interfacing symbolic and numeric systems like Maple and FORTRAN, which is rather convenient, allowing for the use of already existing FORTRAN routines when the need arises for the final computation after the intermediate steps have been symbolically calculated with Maple.

Scott and Fee consider general and special relativity. With regards to the general theory, reference is made to Portugal and Sautu`s work which describes the Riemann package in Maple as constructed for tensor component calculations in general relativity and some tensor abstract manipulations [19]. In special relativity, Scott and Fee consider many-particle systems. Scott, More and Monagan use a symbolic computation system to analyze and manipulate various physical quantities often seen in relativistic many-particle dynamics [20].

For a completely different application, the problem of computing the complex acoustic impedance of a baffled infinite strip radiator is considered in the discipline of audio engineering. This problem had led to conflicting results in the literature. It was deduced, upon undertaking some modelling, that the problem required solving double and triple integrals for which a complete analytical solution was unavailable prior to Lipshitz, Scott and Salvy's work [21]. Being able to implement the evaluation of residues, series and indefinite integrals in Maple to a great extent allowed these researchers to resolve the previously stated issue i.e. computing the complex acoustic impedance of a baffled infinite strip radiator.

Another application of CAS is in the realm of Fluid and Magneto-dynamics, specifically for Asbestos fibre analysis. Asbestos is a known carcinogen which can

lodge in the lungs when inhaled and is suspected to result in respiratory and vascular disease after many years [22]. The connection between the properties of asbestos fibres and these diseases is being investigated by scientists who thus far have depended on manual counting using an optical or electron microscope. Riis investigated scattered-light measurements from asbestos particulates which were aligned in a magnetic field to simplify and organize the process of measuring in the lab [22]. This new method was used to detect asbestos fibres in water samples using the paramagnetic and structural properties of the fibres and was said to be simpler, more elegant, cheaper and quicker than the method of using the electron microscope. Maple was used in the theoretical modelling and experimental verification aspects of the work. Information that matched experimental data quite remarkably regarding fibre dimensions could then be extracted from the data rather easily.

In the nineties, the use of CAS software spread to many more areas including (though not limited to) applied mathematics, reliability theory, applied mechanics and related areas. The use of Computer Algebra Systems also branched out into areas like process control [23–27], fluid dynamics and biological or physiochemical systems. The latter years of this decade saw CAS software, besides being used for symbolic computations, being used numerically or in a combination of symbolic and numeric computations.

Computer algebra systems are popularly used in applied mechanics and related areas [28–30]. They provide closed-form solutions, help make direct semi-analytical, semi-numerical or symbolic-numerical methods attainable [29], [31] and are used in perturbation techniques as will be discussed later. Symbolic-numerical methods consist of numerical values as well as symbols in the consequent results which are diversely useful for example in the subject of optimization [29]. They aid in the direct conversion of theoretical formulae found in books to formulae seen in computer languages therefore minimizing related errors.

Ioakimidis applied CAS software (Mathematica) to the classical torsion problem as laid out in [32–34]. In his paper [35], he analyzed the cases of an elliptic, triangular

and rectangular cross-section in the specimen using the Ritz method. The first two cases resulted in closed-form solutions whereas the last case gave rise to symbolic-numerical results. He utilized the integral to be minimized directly, making the computer perform all computations [35].

Xinhe and Wendong [36] presented the application of symbolic computation to two dimensional domain theory of discrete event systems (DES). They used symbolic computation software to solve dioid equations and to model and analyze DES. Using dioid algebra, the class of DES that can be modelled as timed event graphs could be described by linear equations. However, this algebraic approach though leads to calculation difficulties for system modelling and analysis due to specific dioid properties and high complexity of DES, making computer aided tools necessary.

Usually, analyzing reliability models and lifetime data demands the use of an algorithmic programming language or a reliability package. Hartless and Leemis attempt to combine the flexibility of a programming language with the ease of utilizing a package, bridging the gap as it were. Computational algebra languages, interactive frameworks possessing powerful symbolic and graphical capabilities, can be easily and quickly implemented. For certain applications, this makes them superior to conventional algorithmic languages (like C and FORTRAN). They provide flexible low-level programming constructs; the flexibility of which is often lacking in specified application software packages [37]. In their paper, Hartless and Leemis used Mathematica, a computational algebra language, to analyze three different reliability problems including system reliability bounds, lifetime data analysis and model.

Like many areas in mathematics, linear systems analysis poses problems that require computational implementation rather than solving by hand, under certain governing factors such as accuracy, time and cost of obtaining such solutions. Symbolic computational packages make performing these computations rather attractive. Some merits to using CAS software in the area of linear systems are described in [38], including the ability to store variables in an *exact* form and as such avoid a loss of accuracy when making calculations, the ability to leave variables *unassigned* (without

any numerical value) enabling polynomial operations to be defined in an arbitrary indeterminate, existence of several built-in procedures spanning general and some specialized mathematical areas, and existence of unique high-level programming languages for writing desired procedures. Putting procedures together in a resourceful way to be used in linear systems is therefore not too strenuous a task and this is precisely what Jones, Karampetakis and Pugh do in their paper, creating a package of procedures in Maple (as will be done in this paper) for solving numerous and common problems in linear systems.

Ralston and Maron discuss the use of CAS for demonstrating and applying process control principles [39]. Proportional-Integral-Derivative (PID) control for single loop control and Bode plot analysis for choosing tuning constants and stability range are but a couple of familiar issues to work with in process control. In their paper, Ralston and Maron employ the symbolic as well as numeric and graphical capabilities of the Maple software to create procedures to help in the analysis and design of control systems; generating open loop and closed loop plant responses to set point changes for a single loop with a PID controller in continuous or discrete environments, along with amplitude ratio and phase plots with suitable scales and grid lines.

Ten years after Reduce, a CAS program was used to analyze the stability of difference schemes [40]. Ganzha and Vorozhtsov applied computer algebra systems for the symbolic-numeric stability analysis of difference schemes and schemes of the finite-volume method to approximate the two-dimensional Euler equations for compressible fluid flows specifically on curvilinear grids (widely used in computational fluid dynamics). In solving the above problem, a comprehensive comparison was made between Reduce and Mathematica with respect to their applicability. Using Mathematica, they put forward a new method which allows for significant reduction (a factor of 20) in computer storage necessary at the symbolic stages compared to previous algorithms [41]. The use of both symbolic and numeric means to solve a problem highlights the fact that it is not necessarily about which is better but rather a matter of appropriateness and convenience.

Over the years, sewage pollution in waterways has been a problem of public concern. Though much work has been done for methods of curbing the problem, some sewage still manages to reach waterways causing environmental problems. Knowing how long sewage is likely to remain hazardous after discharge is thus crucial and one way of addressing this is to define mathematical models of the complex bacterial population dynamics, incorporated with empirically determined parameters [42–44], which track the time-evolution of faecal coliform indicator bacteria concentration. Jemmer, Kratz and Read's work proposed using a symbolic algebra computer program to analyze a realistic and simple model to track the sequence of events resulting from a coliform discharge into waterways [45]. In their paper, they illustrated the usefulness of the model, which could be easily adapted to deal with several physical and environmental circumstances, for accurately predicting the system.

Many physiochemical or biological systems may be modelled in terms of a reactive mixture in a flow field. Predicting the role and reactions of different species in a physically dynamic chemically reactive system requires developing a set of partial differential equations to describe the selected set of chemical reactions within a predetermined flow field. The partial differential equations, under certain conditions, can be changed into solvable ordinary differential equations making a continuous problem with infinitely many degrees of freedom become a discrete problem with finitely many unknowns that is easily manageable for a computer. Jemmer [46] applied this approach to the mathematics of mass transfer with chemical reaction and obtained some analytical expressions for simple two-phase problems, in chromatography and bacterial population dynamics as well as a physical interpretation of model parameters. In his paper, Jemmer explored numerical applications of the above approach in large scale chemical process kinetics and plasma chemistry modelling.

In the last decade however, much progress has been made applying CAS not only to conventional areas but to more complex problems such as high-level data analysis, control systems analysis and design (computer aided), complex mathematical, physics and chemistry applications, communication and circuit systems, and many other

engineering systems. Much emphasis has been placed on the importance of a more holistic way of solving problems- symbolically and numerically; not in competition but in complementing each other.

Multimedia applications are a growing market requiring development of complex integrated circuits specific to their application with significant data-path portions. But most high-level synthesis tools and methods namely, most arithmetic-level optimizations, are unable to synthesize data paths automatically to make full use of complex arithmetic library blocks. Symbolic algebra can therefore be employed to construct arithmetic-level decomposition procedures [47]. In their paper, Peymandoust and Micheli present a tool that uses complex arithmetic components for improving and mapping data flow descriptions into data paths. Traditionally, mathematical manipulation with computers and calculators centers on fixed-length integers and fixed-precision floating-point arithmetic termed numeric computation compared to symbolic computation that supports exact rational and arbitrary precision floating-point arithmetic as well as algebraic manipulation of expressions having symbols. A multivariate polynomial representing a data path is that which Peymandoust and Micheli analyzed symbolically in their design, decomposing it into polynomials that represent the building blocks available in the target library. This decomposition is referred to in symbolic computer algebra as simplification modulo set of polynomials (called *simplify* in Maple).

There is an increasing need for symbolic manipulations for various mathematical projects in several areas that range from the simple cases that require computing the Laplace transform or its inverse or obtaining the transfer matrix for a system with known parameters, to the demanding cases which may include finding solutions to issues arising from a variety of control problems like those related to decoupling, model matching, stabilization, tracking and regulation or results that will shed more light on some system structural properties. These necessitate using a computer for such time-consuming and tedious mathematical computations. Symbolic computations and computer algebra help develop software and hardware systems for symbolic mathematical computations and effective symbolic algorithms for evaluating mathematically formulated problems.

As has already been noted, CAS and numerical software are not designed and constructed to solve the same problems but can be seen as complementary tools rather than adversary ones. The idea of using toolboxes created in different CAS environments is also not a new idea as can be seen with CALI (a Reduce package consisting of algorithms for commutative algebra computations [48]), Control System Professional (a Mathematica package containing control algorithms [49], [50]), NCAlgebra (a Mathematica package for non-commutative algebra [51]), to name a few. CAS programs are now being applied successfully to various areas of engineering including robotics [52–62], non-linear dynamics, computational fluid dynamics, aerodynamics, extensively in control systems [25], [52], [54], [62–76].

Karampetakis and Vardulakis's paper [77] is one that illustrates the role that symbolic computations play in control analysis and design, as it highlights a collection of papers on symbolic computations. The paper also discusses some advantages of CAS in addition to allowing exact arithmetic and others, already discussed above. The advantages include: interactivity of software, visualization of results (two- or three-dimensional graphics), availability for different types of computer platforms, allowing for exploration of different scenarios, capability of working with commutative and non-commutative algebra problems for which numerical environments are unsuitable, and research capabilities (such as testing conjectures, obtaining solution steps for mathematical algorithms thereby avoiding errors from hand-calculations, modifying and improving algorithms for solving problems, obtaining closed form solutions which help provide more insight into the problem, creating large mathematical tables, looking into new algorithms and methods, helping focus on ideas instead of calculations). CAS's still do have disadvantages like difficulty in defining the required solution domain, difficulty obtaining exact results when a closed form solution does not exist, potential impracticality of symbolic matrix analysis due to large number of symbols, difficulty handling large-scale problems because of excessive computational resource requirements, difficulty connecting with other applications and the presence of particularities only learned by experience.

Lutovac and Tosic [78] also discuss the symbolic analysis and design of control systems. Their paper discusses the role and importance of symbolic computation in control engineering and signal processing, while trying to avoid losing insight into the system being analyzed due to the great quantity of numerical data produced by contemporary computer tools. They use Mathematica to symbolically resolve some real-life application examples (with regards to linear systems, non-linear systems, algorithm development, modelling and simulation) and present an original approach to algorithm development, system design and symbolic processing to help effectively overcome some of the problems encountered with the traditional approach. Here, the authors realize the possible inefficiency of the classic computer tools, essentially based on numeric computation, to meet the new generation efficient system needs, since the design space is practically unbounded. They suggest using symbolic techniques to complement the rather obsolete traditional extensive numeric algorithms [78]. Interactive programs (like CAS) are very useful in helping to steadily investigate possible algorithm design. This then allows for effective design in converting algorithms into working systems (i.e. implementing the algorithms). CAS and symbolic processing can thereby help get insight into the inner workings of a complex system.

CAS can be applied to a mathematical, control problem to efficiently compute optimal control variational symmetries. Gouveia, Torres and Rocha do this in their paper [79] and then use the symmetries to acquire families of Noether's first integrals (possibly in the presence of non-conservative external forces). In their paper, they attain eight independent first integrals for the sub-Riemannian nilpotent problem as an application which was made possible by the introduction of invariance symmetries up to a gauge term and the presence of non-conservative external forces to an old package [80]. These modifications aided in seeing an improvement in running times. In order to mathematically describe variational symmetries, which keep an optimal control problem invariant, two transformations performed in succession may be replaced by one transformation of the same family. There exists an identity transformation and there

exists an inverse transformation for each transformation. Gouveia, Torres and Rocha create a Maple package to calculate variational symmetries and respective Noether's first integrals automatically in variations and optimal control calculus.

For its potential applications to secure communications [81–83], chaotic dynamic systems synchronization has gained recent interest. However, performing its computational analysis poses a bit of a problem. Galvez and Iglesias attempt using a computer algebra system (specifically Mathematica) in performing symbolic/ numeric analysis of the chaotic synchronization. Due to the high level of sensitivity of chaotic systems to small perturbations on initial conditions rendering close orbits of the system rapidly uncorrelated, the possibility of synchronization is not so intuitive. Its strong sensitivity to initial conditions also makes numeric methods rather susceptible to error and can therefore not be used to completely and accurately describe the nonlinear nature of dynamical systems [84]. CAS is therefore of great value here.

Symbolic computer algebra is also used in the description and verification of arithmetic circuits, a vital part of computing and signal processing systems nowadays. Watanabe, Homma and Aoki [85] use high-level mathematical objects based on weighted number systems and arithmetic formulae to describe arithmetic circuits which they verify by applying polynomial reduction techniques. The main idea is to define arithmetic circuits in a hierarchical way using integer equations. To help demonstrate the merits of their approach, experimental verification of some arithmetic circuits like multiply-accumulator and FIR filter are undertaken. The result of their approach indicates a sure possibility of verifying practical arithmetic circuits where conventional methods that are inherently built on bit-level data operations failed.

The application of CAS to control design and analysis is of great interest with symbolic computation being used for modelling, simulation and synthesis of systems. Some key benefits and applications of symbolic computation methods with regards to symbolic system simulation are discussed in [86]. Tošić and Lutovac again highlight some of the reasons CAS programs have become of great importance today; providing a powerful high-level programming language to define complex problems, the ability to

produce two- and three-dimensional graphics, arbitrary precision and interval arithmetic and ability to numerically evaluate and symbolically handle classical orthogonal polynomials and special functions in mathematical physics. The authors suggest that since computers have now gained recognition as symbol processors with numbers being just one of a range of symbols and hence a program can be seen as instructions for symbol manipulation, they can be utilized for a much wider range of tasks, changing the way programming is viewed.

Sevastianov, Kulyabov and Kokotchikova express one of the goals of computer algebra systems (software programs that facilitate symbolic mathematics) as helping researchers manipulate with formulae in physics, mechanics, and algebra analysis problems. Many a research process has been involuntarily delayed due to improper handling of mathematical operations, making formula manipulation a critical issue [87]. The CAS program used here is Cadabra, a problem-oriented system made for specific tasks i.e. problems seen in field theory.

There has also been the use of CAS software (nowadays) in computational physics and chemistry as is done in Gebremariam, Bogner and Duguet's work. In their paper, they use a Mathematica package to symbolically execute the application of the density matrix expansion to the Hartree-Fock energy from a chiral effective field theory (EFT) three-nucleon interaction at N_2LO. The package provides a quasi-local approximation to the non-local Hartree-Fock energy by producing a Skyrme-like density functional.

Using the density matrix expansion method is one approach in dealing with ground-state and excited-state properties of medium to heavy mass nuclei. It is based on the addition of microscopic long-range physics. Chiral EFT has seen developments that have allowed for microscopically-based energy density functional. Gebremariam, Bogner and Duguet develop the Hartree-Fock method from a chiral EFT three-nucleon interaction at N_2LO for this purpose. The energy density functional that results has highly non-local nature giving rise to the need to apply the density matrix expansion to obtain a Skyrme-like quasi-local energy density functional [88].

Deriving a Skyrme-like energy density functional from the exact Hartree-Fock energy using density matrix expansion (especially for the involvement of the three-nucleon interactions) requires remarkable algebra, making manual calculations quite impractical. The derivation possesses features, including involving similar, repetitive algebraic steps. The derivation mostly does not involve numerical computation and the part that appears to involve numerical computation (like multidimensional integrals) can be evaluated with analytical expansions and symbolic integration, both agreeable to symbolic computation. The authors also discuss and illustrate steps on how this approach can be extended to other three-nucleon interactions.

2.3 CAS and Integral transforms

Applying Computer Algebra Systems to integral transforms began gaining prominence about two and a half decades ago. The early implementations had to do mainly with proposing efficient techniques to better algorithm engineering.

As previously discussed, the use of symbolic algebra for fast arithmetic to compute a large class of problems associated with applications in communication systems, particularly in digital signal processing, coding and cryptography, was presented by Thomas Beth in the mid-eighties [89]. Symbolic algebra is used to change domains in order to compute convolutions in group algebra, which become less-complex component-wise products in spectral algebra, obtained by algorithms fast enough to evaluate the discrete Fourier transform and its inverse. Beth describes techniques applied in the algorithm engineering work during the construction of devices for real-time application in digital signal processing, coding and cryptography.

In 1988, the two dimensional symbolic substitution was a novel technique based on optical technology used to construct parallel computers. Algorithms using this technique were developed for arithmetic/logic operations and complex scientific computations such as matrix algebra and the Fast Fourier Transform [90]. The systems' performance was shown to be potentially superior to that of existing electronic array processors.

Nonlinear functional expansions used to express analytical functionals are analogous to Fourier series or integral expansions of response functions of linear systems. This characteristic of the non-commutative algebra is significant to this approach and is shown to be available in many symbolic programming languages. Nonlinear ordinary differential equations can be solved with these expansions using Laplace-Borel transforms. Can and Unal [91] demonstrated in their work that the transform of the response of the nonlinear system is a Cauchy product of its transfer function and the transform of the input function of the system, together with memory effects. This transfer-function approach is applied to nonlinear electronic circuits.

The nineties saw the implementation of faster algorithms for these transforms but also progress into integrating CAS platforms with other platforms to improve functionality. Symbolic and numerical computer algebra were seen and used complementarily (as opposed to independently) in computations.

In 1991, Rodriguez used tensor product formulations of fast Fourier transform (FFT) algorithms in the automated VLSI implementation of the discrete Fourier transform (DFT). Tensor product algebra is a branch of finite-dimensional multi-linear algebra [92]. He also discussed a transformation technique between a symbolic computation environment and a behavioral synthesis environment for the transferring of functional primitives.

The stability regions of Jameson's schemes as applied to the two dimensional advection-diffusion equations were efficiently obtained for the first time in Ganzha and Vorozhtsov's work with the aid of a symbolic-numerical method that performed Fourier stability analyses of different initial-value problems for hyperbolic or parabolic partial differential equations [93]. The symbolic aspect of this work used the CAS software REDUCE for the symbolic computation of the result and to generate the FORTRAN function to compute its value.

The mid-nineties also saw the use of a recursive digital-filter method to evaluate the chirp response of a thickness-resonant, single-plate transducer. The Fourier transform

was considered to be rather limited and in order to avoid those limitations and the limitations of the frequency-domain circuit, a Laplace transform was used instead.

The Computer Algebra Library for Parallel Symbolic Computation package, referred to by creators as CALYPSO was implemented in a computer algebra system in 1997. The package consisted of parallel algorithms for multiple-precision arithmetic using a message-passing model of computation. Machine-dependent parallel algorithms (like multiplication algorithms) were used to develop different parallel Karatsuba type and FFT-based schemes, including the integer 3-primes and floating-point FFT algorithms. A layered structure was also seen here for portability and easy extensibility of the code.

The recognition of CAS as useful systems that are here to stay allowed for the creation of packages for performing specific tasks in the late-nineties. Truncated Fourier series in approximated frequencies were used, where their corresponding trajectories were described by intersections of hypersurfaces defined by pieces of multivariate power series in phase variables of the system. These characterized the families of solutions of the normal form method for building analytic approximations in a neighborhood of the stationary point to the Henon-Heiles system. The NORT package, containing several operators written in Standard LISP, was created to treat truncated multivariate power series in arbitrary dimensions. The coefficients of the normal form and corresponding transformation were obtained from this package [94].

In more recent years there have been more applications of CAS software (along with other tools) to integral transforms. Specifically, in 2000, a *Mathematica* program was developed to find all factorizations of matrices, like the *polyphase matrices,* by automating the Euclidean algorithm for Laurent polynomials. This came about while analyzing the *lifting scheme* for finding the discrete wavelet transform that depends on the factorization of such matrices [95]. Wavelets are sets of basic functions used in the analysis of sharply-varying or non-periodic signals and images.

Felippa [96] also discussed the improvement of the mass and geometric stiffness matrices of a Bernoulli-Euler plane beam by interweaving classical techniques like

Fourier analysis and weighted orthogonal polynomials with newer tools i.e. finite element templates and computer algebra systems. Symbolic computation was used to implement Fourier analysis for seeking out template signatures of mass and geometric stiffness that deliver matrices with desirable properties, when used in combination with the well-known Hermitian beam material stiffness.

In 2002, a group of researchers delved into the issue of acquiring new formulations by using symbolic computations with Grobner bases for developing an algorithm for automatically transforming systems of partial differential equations (PDEs) that are more accessible to numerical solution [97]. Modelling in science and engineering, across various fields as material sciences, physics, chemistry, biology, economics, aeronautics, and oceanography, often utilizes systems of PDEs which can be challenging to solve numerically. The presented approach made available a powerful set of tools to handle large systems of PDEs.

Lucet's work in 2009 [98] speaks of the huge progress made in computational analysis, where numerical computation of fundamental transforms arising from convex analysis are modeled using symbolic, numeric, and hybrid symbolic-numeric algorithms. The most efficient numerical algorithms useful for applications in image processing (computing the distance transform, the generalized distance transform, and mathematical morphology operators), partial differential equations (solving Hamilton–Jacobi equations and using differential equations numerical schemes to compute the convex envelope), max-plus algebra (computing the equivalent of the fast Fourier transform), multifractal analysis, etc. [98] are discussed. These applications are seen in fields like computer vision, robot navigation, thermodynamics, electrical networks, medical imaging, and network communication, among others.

Fourier transforms play an important role in digital filter design which when implemented with numeric-based tools, typically generate a great deal of numeric data, which can easily cause the user to lose insight into the phenomenon under investigation. Computer algebra systems have been shown to successfully overcome some problems encountered in the traditional numeric-only approach. An original approach to algorithm

development and digital filter design using CAS was implemented to develop an algorithm for an infinite impulse response (IIR) filter design (theoretically impossible to do using the traditional approach) [99].

Computer Algebra Systems are now making strides in physical modeling and simulation. Maplesoft, the trade name for the Canadian software company Maple Inc., has developed a modeling and simulation tool called *MapleSim* that generates model equations, runs simulations, and performs analyses. Being built on a foundation of symbolic computation technology allows it to efficiently handle all of the complex mathematics involved in the development of engineering models, including multi-domain systems and plant models for control applications. There are several components from areas of science and engineering such as electrical, mechanical, and thermal engineering fields, in the MapleSim library that can be connected together to model a system. Also included in MapleSim are traditional signal flow components meaning that causal modeling methods and acausal techniques can be combined without the need to specify signal flow direction between all components. MapleSim, while producing high-fidelity, high-performance models, reduces model development time from months to days.

The popularity of the application of Fourier transforms to solve problems in many areas of science and engineering is therefore understandable. The simplest form of this Fourier transform is the one dimensional case which involves a single variable and can be analyzed continuously or numerically. In the continuous form, a solution is often obtained by employing a table-driven approach. On the other hand, the numerical form of the 1D Fourier transform uses the Discrete Fourier transform (DFT), which is often used to numerically approximate the continuous form of the transform [100]. The development of the Fast Fourier Transform (FFT) prompted the widespread adoption of the DFT as a useful computation tool [101].

Similarly, the two dimensional Fourier transform is a logical progression from the one dimensional case. The 2D Cartesian case is simply two one dimensional cases (one in each Cartesian direction) and can therefore be expected to follow similar concepts as the

1D Fourier transform. This implies that the ideas of the one dimensional case still apply i.e. the complimentary use of the continuous and numerical forms of the transform.

However, there are occasions when a system is best expressed in polar coordinates, prompting the need for a Fourier transform in polar coordinates. In line with the developments of Fourier transforms for the one and two dimensional Cartesian coordinate cases, the polar coordinates version of the 2D Fourier transform can be expected to have a continuous as well as a numerical implementation. In this work, we seek to obtain exact, closed-form results and as such utilize the continuous form. Symbolic Computer Algebra is used to compute algebraic, closed-form versions of the two-dimensional Fourier transform in polar coordinates.

Computer algebra systems (CAS) can provide closed-form solutions, help make direct semi-analytical, semi-numerical or symbolic-numerical methods attainable [29], [31] and are useful in many applications. Symbolic-numerical methods consist of numerical values as well as symbolic expressions in the consequent results, which are also widely useful [29]. CAS software aid in the direct conversion of theoretical formulae found in books to formulae seen in computer languages. This helps in minimizing errors related to numerical implementations.

Many application areas pose problems that require closed form, algebraic approaches. Symbolic computational packages make performing these computations rather attractive. Some merits to using CAS software include the ability to store variables in an *exact* form and as such avoid a loss of accuracy when making calculations, ability to leave variables *unassigned* (without any numerical value) enabling polynomial operations to be defined via several built-in procedures spanning general and specialized mathematical areas, and existence of unique high-level programming languages for writing desired procedures [38].

In the last decade, much progress has been made in applying CAS to conventional areas and also to more complex problems. Much emphasis has been placed on the importance of a more holistic way of solving problems- symbolically and numerically - not in competition but in complementing each other.

There exists a number of CAS software such as MACSYMA, Reduce and Mathematica but the one used for this work is Maple, an interactive system developed in Waterloo, Canada for algebraic computation. The idea of a toolbox as used in this work is also referred to as a 'library' of 'package'. It comprises a collection of tools to perform specific functions.

Maple possesses Hankel and Fourier transforms which are not used in this work. In the case of the former, the definition used is uncommon and inconsistent with the needed definition. The only Fourier transforms offered in Maple are the one dimensional Fourier transform (Cartesian coordinates), the Fourier sine and cosine transforms and the inverse one dimensional Fourier transform. In both cases (Hankel and Fourier transforms), it is deduced that results are obtained from internally programming some basic functions into the procedures; the methodology of this being unknown. A function outside these basic set is therefore unevaluable and the query is returned without being evaluated.

There is a need to implement multidimensional Fourier transforms due to their usefulness. In this work particularly, the two dimensional Fourier transform is sought for problems that are better defined in or naturally lend themselves to curvilinear/cylindrical coordinates. The manual evaluation of the above is complex and lengthy. A different approach is thus employed to achieve this goal. This approach also implements concepts that offer closed-form results for a wider range of functions. Unlike the built-in Maple transforms, an attempt is made at directly solving for an expression falling outside the programmed set before returning the unevaluated query if it fails.

It is important to realize that since most of the Maple code is proprietary, the implementation of this toolbox had to be developed from scratch, making this project rather novel. Results are obtained and verified by querying the toolbox via it various procedures and contrasting with known results.

Chapter 3 :
Mathematical Theory of Two Dimensional Fourier transforms in Polar Coordinates

3.1 Governing Equations

The governing equations for the two dimensional Fourier Transform in polar coordinates are presented in this section. The main equations discussed include the equation for the one dimensional Hankel Transform of a function, that of the Fourier series of a function, and finally the actual two dimensional Fourier Transform in polar coordinates. The latter equation, which is useful for functions (even those that do not possess circular symmetry) that are best described in polar coordinates, is a combination of the first two[5].

A summary of these integral transforms and the development of the Fourier operational toolset of Dirac-delta, exponential, spatial shift, multiplication and

convolution are provided in this chapter and summarized at the end in Table 1. The mathematical material presented in this chapter is not the work of the candidate but of the supervisor. For a detailed account of the development, please refer to [5].

3.1.1 Hankel transform

The *n*th order Hankel transform is defined by the integral [102]

$$\widehat{F}_n(\rho) = \mathbb{H}_n\{f(r)\} = \int_0^\infty f(r) J_n(\rho r) r dr \,, \tag{3.1}$$

where $J_n(z)$ is the *n*th order Bessel function with the overhat indicating a Hankel transform as shown in equation (3.1). Here, *n* may be an arbitrary real or complex number. The Hankel transform is self-reciprocating and the inversion formula is given by

$$f(r) = \int_0^\infty F_n(\rho) J_n(\rho r) \rho d\rho \,. \tag{3.2}$$

The Hankel transform exists only if the Dirichlet condition is satisfied, i.e. $\int_0^\infty \left| r^{1/2} f(r) \right| dr$ exists and is particularly useful for problems involving cylindrical symmetry.

3.1.2 Fourier series

Many transforms, including Fourier transforms, are usually thought of in terms of operators and paired functions but Fourier *series* are typically introduced as an equality $f(\theta) = \sum_{n=-\infty}^{\infty} f_n e^{in\theta}$ and not an operator. In this work, the concept of considering a Fourier series in the same way as a Fourier transform, i.e. a transform/operator is proposed. The symbol $\mathbb{F}_S\{f(\cdot)\}$ is used to represent this transform. The forward Fourier Series *Transform* is defined as the operation of finding the Fourier series coefficients:

$$f_n = \mathbb{F}_S\{f(\theta)\} = \frac{1}{2\pi} \int_0^{2\pi} f(\theta) e^{-in\theta} d\theta \tag{3.3}$$

and the inverse transform as the one where the original function is returned from the coefficients by constructing the Fourier series itself:

$$f(\theta) = \mathbb{F}_S^{-1}\{f_n\} = \sum_{n=-\infty}^{\infty} f_n\, e^{in\theta}\,. \tag{3.4}$$

Hence f_n and $f(\theta)$ are a Fourier (series) pair $f(\theta) \;\Leftrightarrow\; f_n$.

Fourier series coefficients, rather than being complex can also be real and are given as

$$\begin{aligned} A_n &= \frac{1}{\pi}\int_0^{2\pi} f(\theta)\cos(n\theta)\,d\theta, \quad n \geq 0 \\ B_n &= \frac{1}{\pi}\int_0^{2\pi} f(\theta)\sin(n\theta)\,d\theta, \quad n \geq 1 \end{aligned} \tag{3.5}$$

where A_n are the Fourier cosine coefficients, B_n are the Fourier sine coefficients, and $f(\theta)$ is a sum in terms of the Fourier cosine and sine coefficients:

$$f(\theta) = \frac{a_0}{2} + \sum_{n=1}^{\infty} A_n \cos(n\theta) + B_n \sin(n\theta) \tag{3.6}$$

'Fourier Pairs' is a term used when working with Fourier transforms, meaning a pair of functions where one is the Fourier transform of the other. For full 2D Fourier transforms, we denote a Fourier pair as $f(x,y) \;\Leftrightarrow\; F(\omega_x, \omega_y)$ which means that $F(\omega_x, \omega_y) = \mathbb{F}_{2D}\{f(x,y)\}$. Similarly, the notation $f(r,\theta) \;\Leftrightarrow\; F(\rho,\psi)$ implies that $F(\rho,\psi) = \mathbb{F}_{2D}\{f(r,\theta)\}$. These functions are 'paired' by means of a Fourier transform, and this notation is captured with the $\Leftrightarrow$ arrow. In terms of the Fourier series expansion of the function and its transform, equation $F(\rho,\psi) = \mathbb{F}_{2D}\left(f(r,\theta)\right)$ becomes

$$\sum_{n=-\infty}^{\infty} f_n(r)\, e^{in\theta} \quad\Leftrightarrow\quad \sum_{n=-\infty}^{\infty} F_n(\rho)\, e^{in\psi} \tag{3.7}$$

Since both sides of equation (3.7) contain an infinite series, where essentially θ is replaced with ψ and vice versa, it is actually far more convenient to denote the Fourier pair of (3.7) as

$$f_n(r) \quad\Leftrightarrow\quad F_n(\rho) \tag{3.8}$$

where the series is implied but dropped for shorthand and more importantly that the relationship between the Fourier pair of (3.8) is not that of a Fourier transform as a naive interpretation of (3.8) would imply. This is discussed in a later section (3.1.3.1).

3.1.3 Forward Fourier transform

The 2D Fourier transform of a function $f(x,y)$ in Cartesian Coordinates is defined as

$$F(\vec{\omega}) = F(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y)\, e^{-i(\omega_x x + \omega_y y)} dx\, dy\,. \tag{3.9}$$

The inverse Fourier transform is given by

$$f(\vec{r}) = f(x,y) = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\omega_x, \omega_y) e^{i\vec{\omega}\cdot\vec{r}}\, d\omega_x\, d\omega_y\,, \tag{3.10}$$

where the shorthand notation of $\vec{\omega} = (\omega_x, \omega_y), \vec{r} = (x,y)$ has been used.

Polar coordinates can be introduced as $x = r\cos\theta, \quad y = r\sin\theta$ and similarly in the spatial frequency domain as $\omega_x = \rho\cos\psi \quad \omega_y = \rho\sin\psi$ otherwise written as, $r^2 = x^2 + y^2$, $\theta = \arctan(y/x)$ and $\rho^2 = \omega_x^2 + \omega_y^2$, $\psi = \arctan(\omega_y/\omega_x)$.

It then follows that the two-dimensional Fourier transform can be written as

$$F(\rho,\psi) = \int_{0}^{\infty} \int_{-\pi}^{\pi} f(r,\theta)\, e^{-ir\rho\cos(\psi-\theta)} r\, dr\, d\theta\,. \tag{3.11}$$

Thus, in terms of polar coordinates, the Fourier transform operation transforms the spatial position radius and angle (r,θ) to the frequency radius and angle (ρ,ψ).

Using $\vec{r}$ to represent (r,θ) in physical polar coordinates and $\vec{\omega}$ to denote the frequency vector (ρ,ψ) in frequency polar coordinates, the following expansions are valid [2]

$$e^{i\vec{\omega}\vec{r}} = \sum_{n=-\infty}^{\infty} i^n J_n(\rho r)\, e^{in\theta} e^{-in\psi} \tag{3.12}$$

$$e^{-i\vec{\omega}\vec{r}} = \sum_{n=-\infty}^{\infty} i^{-n} J_n(\rho r)\, e^{-in\theta} e^{in\psi} \tag{3.13}$$

These expansions along with the Fourier series expansion, $f(\vec{r}) = \sum_{n=-\infty}^{\infty} f_n(r)\, e^{in\theta}$ and

$$\int_0^{2\pi} e^{in\theta} d\theta = 2\pi\, \delta_{n0} = \begin{cases} 2\pi & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

where δ_{nm} denotes the Kronecker delta function, can be applied to definition of the forward 2D Fourier transform in equation (3.9) to give

$$\begin{aligned} F(\vec{\omega}) &= \int_{-\infty}^{\infty} f(\vec{r}) e^{-i\vec{\omega}\vec{r}}\, d\vec{r} \\ &= \int_0^{\infty} \int_0^{2\pi} \sum_{m=-\infty}^{\infty} f_m(r) e^{im\theta} \sum_{n=-\infty}^{\infty} i^{-n} J_n(\rho r)\, e^{-in\theta} e^{in\psi}\, d\theta\, r dr \\ &= \sum_{n=-\infty}^{\infty} 2\pi\, i^{-n}\, e^{in\psi} \int_0^{\infty} f_n(r) J_n(\rho r)\, r dr. \end{aligned} \tag{3.15}$$

3.1.3.1 Connection between the 2D Fourier transform and Hankel transform

By comparing the expression for the Fourier series expansion of the Fourier transform, $F(\vec{\omega}) = \sum_{n=-\infty}^{\infty} F_n(\rho)\, e^{in\psi}$ and equation (3.15), we obtain the expression for $F_n(\rho)$, the Fourier coefficients of the Fourier domain expansion. This can also be interpreted in terms of a Hankel transform as in equation (3.1) as:

$$F_n(\rho) = 2\pi\, i^{-n} \int_0^{\infty} f_n(r) J_n(\rho r)\, r dr = 2\pi\, i^{-n} \mathbb{H}_n\{f_n(r)\}. \tag{3.16}$$

3.1.3.2 Special cases

A couple of cases of Fourier transforms are discussed here. These cases include the radially symmetric functions and functions that are not radially symmetric.

Radially Symmetric functions

A radially symmetric function, f simply suggests that it can be written as a function of r only. It can therefore be removed from the integration over the angular coordinate so that equation (3.11) becomes

$$F(\rho,\psi)=\int_0^{\infty} rf(r)dr\int_{-\pi}^{\pi} e^{-ir\rho\cos(\psi-\theta)}d\theta\,. \tag{3.17}$$

Using the integral definition of the zero order Bessel function [5], equation (3.17) can be rewritten as

$$F(\rho)=\mathbb{F}_{2D}\{f(r)\}=2\pi\int_0^{\infty} f(r)J_0(\rho r)rdr=2\pi\,\mathbb{H}_0\{f(r)\} \tag{3.18}$$

making the special case of the two-dimensional Fourier transform of a radially symmetric function the same as the zeroth-order Hankel transform of that function.

Note: $f(r)$ and $F(\rho)$ are functions with radial symmetry in a 2D regime, and $\mathbb{F}_{2D}\{\bullet\}$ is an operator in a 2D regime, while $\mathbb{H}_0\{\bullet\}$ is an operator in a 1D regime.

Non-radially Symmetric functions

A non-radially symmetric function, $f(r,\theta)$ indicates a function of both r and θ, and thus the preceding result (equation (3.17)) can be generalized. Since $f(r,\theta)$ depends on the angle θ, it can be expanded into a Fourier series suitable for functions that are separable in r and θ

$$f(\vec{r})=f(r,\theta)=\sum_{n=-\infty}^{\infty} f_n(r)\,e^{in\theta} \tag{3.19}$$

where $f_n(r)=\dfrac{1}{2\pi}\displaystyle\int_0^{2\pi} f(r,\theta)\,e^{-in\theta}d\theta$. Similarly, the 2D Fourier transform $F(\rho,\psi)$ can also be expanded into its own Fourier series such that

$$F(\vec{\omega})=F(\rho,\psi)=\sum_{n=-\infty}^{\infty} F_n(\rho)\,e^{in\psi} \tag{3.20}$$

and $F_n(\rho)=\dfrac{1}{2\pi}\displaystyle\int_0^{2\pi} F(\rho,\psi)e^{-in\psi}d\psi$.

It is extremely important to note that $F_n(\rho)$ is NOT the Fourier transform of $f_n(r)$. In fact, the relationship between $f_n(r)$ and $F_n(\rho)$ is clear from equation (3.16).

3.1.4 Inverse transform

The corresponding 2D inverse Fourier transform is written as

$$f(\vec{r}) = \frac{1}{(2\pi)^2} \int_0^{\infty} \int_0^{2\pi} F(\vec{\omega}) e^{i\vec{\omega}\vec{r}} \, d\psi \, \rho d\rho \, . \tag{3.21}$$

Using equation (3.20) along with the expansion (3.12) yields

$$f(\vec{r}) = \sum_{n=-\infty}^{\infty} \frac{1}{2\pi} i^n \, e^{in\theta} \int_0^{\infty} F_n(\rho) J_n(\rho r) \, \rho d\rho \tag{3.22}$$

so that

$$f_n(r) = \frac{i^n}{2\pi} \int_0^{\infty} F_n(\rho) J_n(\rho r) \, \rho d\rho = \frac{i^n}{2\pi} \, \mathbb{H}_n \{ F_n(\rho) \} \, . \tag{3.23}$$

It is clear here that the *n*th term in the original functions Fourier series transforms (Hankel) into the *n*th term of the Fourier series of the Fourier transform function but all the terms are not equivalently transformed as it is an *n*th order Hankel transform for the *n*th term. Also the general 2D Fourier transform for radially symmetric functions is equivalent to the zeroth order Hankel transform (as shown in section discussing Radially Symmetric functions above). Hence, the mapping from $f_n(r)$ to $F_n(\rho)$ is *not* a 2D Fourier transform but rather an *n*th order Hankel transform.

Key concept

From the derivations made in sections 3.1.1, 3.1.2 and 3.1.3, some vital conclusions can be drawn. Of great importance is the concept that taking the 2D Fourier transform of a function (as seen from equations (3.15), (3.16) and (3.20)) is equivalent to 1) first finding its Fourier series expansion in the angular variable and 2) then finding the *n*th order Hankel transform (of the spatial radial variable to the spatial frequency radial variable) of the *n*th coefficient in the Fourier series and appropriately scaling the result and 3) finally obtaining the inverse Fourier series. Since each of these operations involves integration

over one variable only with the others being considered parameters with reference to the integration, the order in which these operations are performed is interchangeable.

3.2 Rules

The rules that depict the operational toolset of the Fourier transform are briefly discussed in this section. These rules are given for Dirac-delta functions, complex exponentials, shift, multiplication and convolution.

3.2.1 The Dirac-delta function and its transform

The definition of the unit-mass Dirac-delta function in 2D polar coordinates is

$$f\left(\vec{r}\right)=\delta(\vec{r}-\vec{r}_0)=\frac{1}{r}\delta\left(r-r_0\right)\delta\left(\theta-\theta_0\right). \tag{3.24}$$

As previously discussed, in order to find the Fourier transform, the Fourier series expansion is required, followed by a Hankel transform and then the inverse Fourier series of the appropriately scaled result. Thus for the Dirac-delta function, the Fourier series expansions terms are

$$f_n(r)=\frac{1}{2\pi}\int_0^{2\pi}\frac{1}{r}\delta\left(r-r_0\right)\delta\left(\theta-\theta_0\right)e^{-in\theta}d\theta=\frac{1}{2\pi r}\delta\left(r-r_0\right)e^{-in\theta_0} \tag{3.25}$$

so that equation (3.24) can be written as

$$f\left(\vec{r}\right)=\delta(\vec{r}-\vec{r}_0)=\frac{1}{r}\delta\left(r-r_0\right)\delta\left(\theta-\theta_0\right)=\sum_{n=-\infty}^{\infty}\frac{1}{2\pi r}\delta\left(r-r_0\right)e^{-in\theta_0}e^{in\theta} \tag{3.26}$$

Then the full transform is given by

$$\begin{aligned}F\left(\vec{\omega}\right)&=\sum_{n=-\infty}^{\infty}F_n(\rho)ei^{jn\psi}=\sum_{n=-\infty}^{\infty}2\pi\, i^{-n}\, e^{in\psi}\int_0^{\infty}f_n(r)J_n(\rho r)\,rdr\\&=\sum_{n=-\infty}^{\infty}2\pi\, i^{-n}\, e^{in\psi}\int_0^{\infty}\frac{\delta\left(r-r_0\right)}{2\pi r}e^{-in\theta_0}J_n(\rho r)\,rdr\\&=\sum_{n=-\infty}^{\infty}i^{-n}J_n(\rho r_0)e^{-in\theta_0}\, e^{in\psi}=e^{-i\vec{\omega}\cdot\vec{r}_0}\end{aligned} \tag{3.27}$$

where equation (3.27) is the 2D linear exponential phase function.

The Fourier transform of the Dirac delta function is the exponential function, as would be expected from the results in Cartesian coordinates.

Hence the Fourier pair for the Dirac-delta function is given by

$$f(\vec{r}) = \delta(\vec{r} - \vec{r}_0) = \frac{1}{r}\delta(r - r_0)\delta(\theta - \theta_0) \Leftrightarrow F(\vec{\omega}) = \sum_{n=-\infty}^{\infty} i^{-n} J_n(\rho r_0) e^{-in\theta_0} e^{in\psi} = e^{-i\vec{\omega}\cdot\vec{r}_0} \tag{3.28}$$

This Fourier pair is included in Table 1.

3.2.1.1 Dirac-delta function at the origin

If $\vec{r}_0$ is at the origin, then the Dirac-delta function in 2D is given by

$$\delta(\vec{r}) = \frac{1}{2\pi r}\delta(r), \tag{3.29}$$

where $\delta(\vec{r})$ represents the Dirac-delta function in the appropriate multi-dimensional coordinate system of potentially varying form which is different from $\delta(r)$ that denotes the most familiar standard 1D scalar version of the Dirac-delta function.

The correct Fourier transform for the Dirac-delta function at the origin is 1 and can be obtained from either equation (3.18) or equation (3.27). Due to the radially symmetric nature of the function, the series consists of only the zeroth order terms i.e. $f_n(r) = f(r)\delta_{n0}$ and $F_n(\rho) = F(\rho)\delta_{n0}$. The Fourier pair is thus given by

$$\frac{1}{2\pi r}\delta(r) \quad \Leftrightarrow \quad 1. \tag{3.30}$$

3.2.1.2 Ring-delta function

The *Ring* delta function is a function that is used often and is given by

$$f(r) = \frac{1}{2\pi r}\delta(r - r_0). \tag{3.31}$$

This function is non-zero only on the ring of radius r_0. The 2D Fourier transform of the Ring-delta is most easily found from equation (3.18) and is given by

$$F(\rho) = 2\pi\int_0^{\infty}\left\{\frac{1}{2\pi r}\delta(r - r_0)\right\}J_0(\rho r)rdr = J_0(\rho r_0). \tag{3.32}$$

Similar to the previous case, the series consists of only the zeroth order terms since the function is radially symmetric. So the full 2D Fourier transform pair is given by

$$\frac{1}{2\pi r}\delta(r-r_0) \quad \Leftrightarrow \quad J_0(\rho r_0). \tag{3.33}$$

3.2.2 The Complex exponential and its transform

From equation (3.12), the 2D complex exponential function can be written in polar coordinates as

$$f(\vec{r}) = e^{i\vec{\omega}_0\cdot\vec{r}} = \sum_{n=-\infty}^{\infty} i^n J_n(\rho_0 r)\, e^{-in\psi_0}\, e^{in\theta} \tag{3.34}$$

so that the Fourier coefficients can be directly seen from the previous equation or can be found from the formula by

$$f_n(r) = \frac{1}{2\pi}\int_0^{2\pi}\sum_{m=-\infty}^{\infty} i^m J_m(\rho_0 r) e^{-im\psi_0}\, e^{im\theta} e^{-in\theta} d\theta = i^n J_n(\rho_0 r)\, e^{-in\psi_0}. \tag{3.35}$$

Following from the orthogonality of the Bessel functions [103]

$\int_0^{\infty} J_n(ux) J_n(vx)\, x dx = \frac{1}{u}\delta(u-v)$, the full 2D Fourier transform is given by

$$\begin{aligned} F(\vec{\omega}) &= \sum_{n=-\infty}^{\infty} 2\pi\, e^{-in\psi_0}\frac{1}{\rho}\delta(\rho-\rho_0)\, e^{in\psi} \\ &= (2\pi)^2\frac{1}{\rho}\delta(\rho-\rho_0)\delta(\psi-\psi_0) = (2\pi)^2\,\delta(\vec{\omega}-\vec{\omega}_0) \end{aligned} \tag{3.36}$$

where

$$\delta(\psi-\psi_0) = \sum_{n=-\infty}^{\infty}\frac{1}{2\pi} e^{-in\psi_0}\, e^{in\psi} \tag{3.37}$$

The correct Fourier pair of the complex exponential is therefore given by

$$f(\vec{r}) = e^{i\vec{\omega}_0\cdot\vec{r}} \quad \Leftrightarrow \quad F(\vec{\omega}) = (2\pi)^2\,\delta(\vec{\omega}-\vec{\omega}_0). \tag{3.38}$$

This Fourier pair is also included in Table 1.

3.2.2.1 Special case of the complex exponential

The Fourier transform of $f(\vec{r}) = 1$ can be computed by substituting $\omega_0 = 0$ into the above formulas as a special case of the complex exponential, which gives

$$F(\vec{\omega}) = (2\pi)^2\frac{1}{\rho}\delta(\rho)\delta(\psi) = (2\pi)^2\,\delta(\vec{\omega}) \tag{3.39}$$

or alternatively, in series form the Fourier transform of $f(\vec{r}) = 1$ is given by

$$F(\vec{\omega}) = \sum_{n=-\infty}^{\infty} \frac{2\pi}{\rho} \delta(\rho) e^{in\psi} . \tag{3.40}$$

The Fourier pair is therefore given by

$$1 \quad \Leftrightarrow \quad (2\pi)^2 \delta(\vec{\omega}) . \tag{3.41}$$

3.2.3 Spatial Shift

The correct expression for a Fourier series shifted in space is found by finding the inverse Fourier transform of the complex exponential-weighted transform. In other words, $f(\vec{r} - \vec{r}_0)$ is found from

$$f(\vec{r} - \vec{r}_0) = \mathbb{F}^{-1}\left\{ e^{-i\vec{\omega}\cdot\vec{r}_0} F(\vec{\omega}) \right\} . \tag{3.42}$$

It is often more efficient to build a new rule on previously-found results, which is why the shifted function is defined according to equation (3.42). The expansion for the complex exponential as well as the rules for finding the product of two expansions has already been found allowing for the relevant spatial shift result to be found in the desired form. It is not sufficient to find any expression for the spatial shift but rather the expression that is sought is one that is (i) in the form of a Fourier series and (ii) in terms of the unshifted coefficients of the original function as this builds the rule for what to do to the coefficients if a shift is desired.

The 2D Fourier transform of a shifted function, $f(\vec{r} - \vec{r}_0)$ is thereby given as

$$\mathbb{F}\left\{ f(\vec{r} - \vec{r}_0) \right\} = e^{-i\vec{\omega}\cdot\vec{r}_0} F(\vec{\omega}) = \sum_{n=-\infty}^{\infty} i^{-n} J_n(\rho r_0)\, e^{-in\theta_0} e^{in\psi} \cdot F(\vec{\omega}) . \tag{3.43}$$

Obtaining the Fourier transform of the spatially shifted function in terms of the Fourier domain coefficients $F_n(\rho)$ is given simply below:

i. Transform the unshifted function to the Fourier domain to find $F(\vec{\omega})$, meaning find its coefficients $F_n(\rho)$ as known from equation (3.16).

ii. Multiply the Fourier transform $F(\vec{\omega})$ by the complex exponential $e^{-i\vec{\omega}\cdot\vec{r}_0}$, which can also be expressed as a series as shown in equation (3.13).

The Fourier pair for the shift rule can therefore be written as

$$f\left(\vec{r}-\vec{r}_0\right) \quad \Leftrightarrow \quad e^{-i\vec{\omega}\cdot\vec{r}_0}F(\vec{\omega})\,. \tag{3.44}$$

It is possible to avoid the transformation to the Fourier domain, and remain in the spatial domain to perform all calculations [5].

3.2.3.1 Spatial Shift of Radially Symmetric functions

The 2D forward and inverse transforms of a radially symmetric function, a function of *r* only, are essentially Hankel transform of order zero. The correct expression of the shift of a radially symmetric function includes the use of the forward (equation (3.18)) and inverse Fourier transforms as well as the shifted function definition in equation (3.42).

While $f(r)$ is radially symmetric and only has the $n=0$ term in its Fourier series, a properly shifted $f(\vec{r}-\vec{r}_0)$ is *not* radially symmetric and thus needs the full complement of entries in its Fourier series even if the new "center" $\vec{r}_0$ is located on the radial axis so that $\theta_0=0$. Since the function is no longer radially symmetric, the 2D Fourier transform is therefore no longer equivalent to a zeroth order Hankel transform. The proper full shift (linear translation in r and angular rotation in θ) demonstrates that a function of *r* becomes a function of *r* and θ once shifted away from the origin. This is *not* the same as $f(r-r_0)$, which *is* a radially symmetric function.

Writing $h(\vec{r})=f\left(\vec{r}-\vec{r}_0\right)$ then, the corresponding Fourier coefficients in Fourier space are given by

$$H_k\left(\rho\right)=i^{-k}e^{-ik\theta_0}F\left(\rho\right)J_k\left(\rho r_0\right) \tag{3.45}$$

which establishes the Fourier-pair

$$f(\vec{r}-\vec{r}_0) \quad \Leftrightarrow \quad \sum_{n=-\infty}^{\infty} i^{-n}e^{-in\theta_0}F\left(\rho\right)J_n(\rho r_0)\,e^{in\psi}\,. \tag{3.46}$$

3.2.3.2 Shift of a Radially Symmetric function in terms of the Shift Operator

Without going into the Fourier domain, the coefficients of the shifted function can be found in terms of the original function (as done before in the case of the shift of general function) by using the forward transform definition, switching the order of integration and applying equation to equation (3.42) to get

$$f(\vec{r}-\vec{r}_0)=\sum_{k=-\infty}^{\infty} e^{ik(\theta-\theta_0)}\int_0^{\infty} f(u)\, S_0^k(u,r,r_0)\; u\,du \tag{3.47}$$

where the shift-type operator is given by $S_n^k(u,r,r_0)=\int_0^{\infty} J_n(\rho u)\,J_{k-n}(\rho r_0)\,J_k(\rho r)\,\rho\,d\rho$.

3.2.4 Multiplication

Here, the product of two functions $h(\vec{r})=f(\vec{r})g(\vec{r})$ is considered where

$f(\vec{r})=\sum_{n=-\infty}^{\infty} f_n(r)e^{in\theta}$ and $g(\vec{r})=\sum_{n=-\infty}^{\infty} g_n(r)e^{in\theta}$ with the coefficients $f_n(r)$ and $g_n(r)$

given by $f_n(r)=\frac{1}{2\pi}\int_0^{2\pi} f(r,\theta)\,e^{-in\theta}d\theta$ and $g_n(r)=\frac{1}{2\pi}\int_0^{2\pi} g(r,\theta)\,e^{-in\theta}d\theta$ respectively.

The Fourier transform of $h(\vec{r})$ is obtained using the Fourier series expansions of $f(\vec{r})$ and $g(\vec{r})$, along with equation (3.13) for the polar form of the 2D complex exponential:

$$\begin{aligned} H(\vec{\omega}) &= \int_{-\infty}^{\infty} f(\vec{r})g(\vec{r})\,e^{-i\vec{\omega}\cdot\vec{r}}d\vec{r} \\ &= \int_0^{\infty}\int_0^{2\pi}\sum_{n=-\infty}^{\infty} f_n(r)e^{in\theta}\sum_{m=-\infty}^{\infty} g_m(r)e^{im\theta}\sum_{k=-\infty}^{\infty} i^{-k}J_k(\rho r)\,e^{-ik\theta}e^{ik\psi}\,d\theta\,rdr. \end{aligned} \tag{3.48}$$

Performing the integration over the angular variable yields

$$H(\vec{\omega})=\sum_{k=-\infty}^{\infty} 2\pi\, i^{-k}e^{ik\psi}\int_0^{\infty}\sum_{m=-\infty}^{\infty} f_{k-m}(r)g_m(r)J_k(\rho r)\,rdr=\sum_{k=-\infty}^{\infty} H_k(\rho)e^{ik\psi} \tag{3.49}$$

where

$$H_k(\rho) = 2\pi\, i^{-k} \int_0^{\infty} \sum_{m=-\infty}^{\infty} f_{k-m}(r)\, g_m(r) J_k(\rho r)\, r dr \tag{3.50}$$

and comparing with equation (3.16), $h_k(r) = \sum_{m=-\infty}^{\infty} f_{k-m}(r) g_m(r)$ which is in fact the convolution of the Fourier series of $f(\vec{r})$ and $g(\vec{r})$.

By some manipulation of these equations along with equation (3.23), the Fourier transform of a product can be shown to indeed be a 2D convolution of their full transforms, $F(\vec{\omega}) ** G(\vec{\omega})$. For $h(\vec{r}) = f(\vec{r})\, g(\vec{r})$, the Fourier transform is given by

$$H(\vec{\omega}) = \frac{1}{(2\pi)^2} F(\vec{\omega}) ** G(\vec{\omega}) \tag{3.51}$$

and thus, the Fourier pair is

$$h(\vec{r}) = f(\vec{r}) g(\vec{r}) \quad \Leftrightarrow \quad H(\vec{\omega}) = \frac{1}{(2\pi)^2} F(\vec{\omega}) ** G(\vec{\omega}). \tag{3.52}$$

3.2.5 Full Two dimensional Convolution

The two-dimensional convolution, like the one-dimensional convolution of two functions is defined by

$$h(\vec{r}) = f(\vec{r}) ** g(\vec{r}) = \int_{-\infty}^{\infty} g(\vec{r}_0) f(\vec{r} - \vec{r}_0) d\vec{r}_0 . \tag{3.53}$$

The double-star notation of $**$ is used to emphasize that this is a 2D convolution and to distinguish it from a 1D convolution.

The Fourier coefficients $H_k(\rho)$ are the convolution of the two series given by

$$H_k(\rho) = \sum_{n=-\infty}^{\infty} G_{k-n}(\rho) F_n(\rho) = G_k * F_k \tag{3.54}$$

and it therefore follows from this equation that

$$H(\vec{\omega}) = G(\vec{\omega}) F(\vec{\omega}), \tag{3.55}$$

as would be expected from the standard results of Fourier theory. For details of the derivation, see [5].

The Fourier pair can then be summarized as

$$f(\vec{r}) ** g(\vec{r}) \quad \Leftrightarrow \quad G(\vec{\omega})F(\vec{\omega}) \tag{3.56}$$

3.2.5.1 2D Convolution of Two Radially Symmetric functions

The 2D convolution of two radially symmetric functions is defined (see equation (3.53)) with the emphasis that *the integration is overall* possible values of *radial and angular* variables. Integrating equation (3.47) with this definition over the angular variable gives a nonzero result only if $k = 0$. The correct convolution definition can therefore be interpreted as

$$h(\vec{r}) = f(\vec{r}) ** g(\vec{r}) = \int_0^\infty g(r_0)\, \Phi(r - r_0)\, r_0 dr_0 \tag{3.57}$$

where $\Phi(r - r_0) = \int_0^\infty f(u)\, S_0^0(u, r, r_0) u\, du = \int_0^\infty F(\rho) J_0(\rho r_0)\, J_0(\rho r) \rho d\rho$ and

$S_0^0(u, r, r_0) = \int_0^\infty J_0(\rho u)\, J_0(\rho r_0)\, J_0(\rho r)\, \rho d\rho$. The unshifted function g is still radially symmetric and thus is not affected by the integration over the angular variable.

It follows that $h(r) = f(r) ** g(r) = \frac{1}{2\pi} \int_0^\infty G_0(\rho) F_0(\rho) J_0(\rho r) \rho d\rho$. It is observed here that the 2D convolution of two radially symmetric functions yields another radially symmetric function and also the well-known convolution-multiplication rule, namely

$$f(\vec{r}) ** g(\vec{r}) = f(r) ** g(r) \quad \Leftrightarrow \quad F(\rho)G(\rho). \tag{3.58}$$

3.2.5.2 Convolution of a Radially Symmetric function with a Non-Radially Symmetric function

Convolving two functions where only one of the functions is radially symmetric holds particular importance for certain physical problems involving non-homogeneous partial differential equations which can be solved via a convolution of the forcing function with the system's Green's functions (often the radially symmetric one). The 2D convolution of two functions as defined in equation (3.53) when Fourier expanded and integrated over the angle variable (non-zero for $k = m$) gives

$$h(\vec{r}) = f(\vec{r}) ** g(\vec{r}) = f(r) ** g(\vec{r}) = \int_{-\infty}^{\infty} g(\vec{r}_0) f(\vec{r} - \vec{r}_0) d\vec{r}_0$$

$$= \int_0^{\infty} \int_0^{2\pi} \sum_{m=-\infty}^{\infty} g_m(r_0) e^{im\theta_0} \left\{ \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} e^{ik(\theta-\theta_0)} \int_0^{\infty} F(\rho) J_k(\rho r_0) J_k(\rho r) \rho d\rho \right\} d\theta_0 r_0 dr_0 \quad (3.59)$$

$$= \sum_{k=-\infty}^{\infty} e^{ik\theta} \int_0^{\infty} g_k(r_0) \phi_k(r - r_0)\, r_0 dr_0$$

where the definition of $\phi_k(r - r_0)$ below is necessary to have the specific version for the case $r_0 = 0$ without which $J_k(0) = 0$ when $k \neq 0$

$$\phi_k(r - r_0) = \begin{cases} \int_0^{\infty} F(\rho) J_k(\rho r_0) J_k(\rho r) \rho d\rho & r_0 \neq 0 \\ \int_0^{\infty} F(\rho) J_k(\rho r) \rho d\rho & r_0 = 0 \end{cases} \quad (3.60)$$

Here, it is obvious that the coefficients involve a simple 1D convolution, *not* a series convolution.

3.2.6 Angular (Circular) Convolution

A circular or angular convolution for two functions $f(\vec{r}) = f(r, \theta)$ and $g(\vec{r}) = g(r, \theta)$ can be defined as

$$f(\vec{r}) *_\theta g(\vec{r}) = \frac{1}{2\pi} \int_0^{2\pi} f(r, \theta_0)\, g(r, \theta - \theta_0)\, d\theta_0 . \quad (3.61)$$

where the $*_\theta$ notation denotes the angular convolution. This is *not* the 2D convolution as previously discussed, but rather a convolution over the angular variable only.

A Fourier series relationship can be developed for this angular convolution operation:

$$h(r, \theta) = f(\vec{r}) *_\theta g(\vec{r}) = \frac{1}{2\pi} \int_0^{2\pi} \left(\sum_{n=-\infty}^{\infty} f_n(r)\, e^{in\theta_0} \right) \left(\sum_{m=-\infty}^{\infty} g_m(r)\, e^{im(\theta - \theta_0)} \right) d\theta_0$$

$$= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f_n(r) g_m(r)\, e^{im\theta} \frac{1}{2\pi} \underbrace{\int_0^{2\pi} e^{jn\theta_0} e^{-im\theta_0} d\theta_0}_{2\pi\delta_{mn}} \quad (3.62)$$

$$= \sum_{n=-\infty}^{\infty} f_n(r) g_n(r)\, e^{in\theta}$$

which shows that the *n*th coefficient of the *angular* convolution of two functions is simply the product of the *n*th Fourier coefficient of each of the two functions. Mathematically, if $h(r,\theta) = f\left(\vec{r}\right) *_{\theta} g\left(\vec{r}\right)$, then $h_n(r) = f_n(r) g_n(r)$; which is also the same result as obtained with Fourier series of (1D) periodic functions.

3.2.7 Radial Convolution

Radial convolution, similar to the angular or circular convolution, is defined as

$$h(\vec{r}) = f(\vec{r}) *_r g(\vec{r}) = \int_0^{\infty} g(r_0,\theta) f(r - r_0,\theta) r_0 dr_0 \tag{3.63}$$

Such convolutions are less often seen than their 2D or angular counterparts; however they are included for completeness.

3.3 Summary and Conclusions

This chapter has considered the polar-coordinate version of the standard 2D Fourier transform (most useful for functions which are naturally described in terms of polar coordinates) and highlighted the operational toolset required for standard Fourier operations, the results of which are summarized in Table 1. Note: standard convolution/multiplication rules do not apply for 1D radial convolution.

Table 1a: Summary of Fourier transform relationships in polar coordinates

$f(\vec{r})$	$f_n(r)$	$F_n(\rho)$	$F(\vec{\omega})$
$f(r,\theta) = \sum_{n=-\infty}^{\infty} f_n(r)e^{in\theta}$	$\frac{1}{2\pi}\int_0^{2\pi} f(r,\theta)e^{-in\theta}d\theta$	$2\pi\, i^{-n}\int_0^{\infty} f_n(r)J_n(\rho r)\, rdr$	$F(\rho,\psi) = \sum_{n=-\infty}^{\infty} F_n(\rho)e^{in\psi}$
$f(r,\theta) = \sum_{n=-\infty}^{\infty} f_n(r)e^{in\theta}$	$\frac{i^n}{2\pi}\int_0^{\infty} F_n(\rho)J_n(\rho r)\rho d\rho$	$\frac{1}{2\pi}\int_0^{2\pi} F(\rho,\psi)e^{-in\psi}d\psi$	$F(\rho,\psi) = \sum_{n=-\infty}^{\infty} F_n(\rho)e^{in\psi}$
$\delta(\vec{r}-\vec{r}_0) = \frac{\delta(r-r_0)}{r}\delta(\theta-\theta_0)$	$\frac{\delta(r-r_0)}{2\pi r}e^{-in\theta_0}$	$i^{-n}J_n(\rho r_0)e^{-in\theta_0}$	$e^{-i\vec{\omega}\cdot\vec{r}_0}$
$e^{i\vec{\omega}_0\cdot\vec{r}}$	$i^n J_n(\rho_0 r)\, e^{-in\psi_0}$	$2\pi\, e^{-in\psi_0}\frac{1}{\rho}\delta(\rho-\rho_0)$	$(2\pi)^2\,\delta(\vec{\omega}-\vec{\omega}_0)$
$f(\vec{r}-\vec{r}_0)$			$e^{-i\vec{\omega}\cdot\vec{r}_0}F(\vec{\omega})$
$f(r-\vec{r}_0)$ (f symmetric)		$i^{-n}e^{-in\theta_0}F(\rho)J_n(\rho r_0)$	$e^{-i\vec{\omega}\cdot\vec{r}_0}F(\rho)$
$h(\vec{r})g(\vec{r})$	$f_n(r) = h_n(r) * g_n(r) = \sum_{m=-\infty}^{\infty} h_{n-m}(r)g_m(r)$	$2\pi\, i^{-n}\int_0^{\infty} f_n(r)J_n(\rho r)\, rdr$	$\frac{G(\vec{\omega})**H(\vec{\omega})}{(2\pi)^2}$
$h(\vec{r})**g(\vec{r})$		$F_n(\rho) = H_n(\rho) * G_n(\rho) = \sum_{m=-\infty}^{\infty} G_{n-m}(\rho)H_m(\rho)$	$G(\vec{\omega})H(\vec{\omega})$
$g(r)**h(r)$ (g,h symmetric)			$F(\rho)G(\rho)$

Table 1b: Contd. - Summary of Fourier transform relationships in polar coordinates

$f(\vec{r})$	$f_n(r)$	$F_n(\rho)$	$F(\vec{\omega})$
$g(\vec{r}) ** h(r)$ $(h \text{ symmetric})$	$f_n(r) = \int_0^\infty g_k(r_0)\phi_k(r - r_0)\, r_0 dr_0$ $\phi_k(r - r_0) =$ $\int_0^\infty H(\rho) J_k(\rho r_0) J_k(\rho r) \rho d\rho$	$2\pi\, i^{-n} \int_0^\infty f_n(r) J_n(\rho r)\, r dr$	$F(\rho, \psi) =$ $\sum_{n=-\infty}^{\infty} F_n(\rho) e^{jn\psi}$
$h(\vec{r}) *_\theta g(\vec{r})$	$f_n(r) = h_n(r)\, g_n(r)$	$2\pi\, i^{-n} \int_0^\infty f_n(r) J_n(\rho r)\, r dr$	$\sum_{n=-\infty}^{\infty} F_n(\rho) e^{jn\psi}$

Chapter 4 : Requirements Analysis

In this chapter, the requirements and conditions to be met for the success of the project are presented. Documenting these requirements allows for well-defined and detailed actionable goals that can be measured and tested. The requirements discussed here are architectural functional, non-functional and design requirements.
The target users for the toolbox include researchers and developers of applications in mostly science and engineering fields such as communication systems- digital signal processing, coding and cryptography, quantum theory, relativity, audio engineering, process control, fluid dynamics, biological/physiochemical systems, computational physics and chemistry, applied mechanics and control systems.

Critical characteristics are discussed. The level of requirements can therefore be measured and improved more systematically. This first implementation of the toolbox can therefore be seen as the starting phase of the project. This chapter also discusses the Computer Algebra System (CAS) software employed in this work and how that is used to create any toolbox. The CAS toolbox created in this thesis is called *SCAToolbox*. It is a

symbolic computer algebra toolbox that provides a comprehensive collection of interactive tools suitable for computing mainly the Fourier transform of expressions in two dimensional polar coordinates.

4.1 Specifications of SCAToolbox package

This section gives a brief overview of the building blocks of the SCAToolbox and the criteria for designing, building and judging its success. The first subsection deals with the design criteria i.e. what the aim and standards are in setting up the toolbox, that allow for establishing the success of the project by measuring the attained level of each of these criteria.

The second subsection discusses the contents of the SCAToolbox itself.

4.1.1 Design criteria

A few things are deemed critical in this work. Some of these are outlined in the design criteria listed below. These criteria depict the standard by which the toolbox is measured. Being able to achieve a high level of most, if not all, these criteria will mean a significantly successful application.

1. **Modularity:** Comprised in the resulting software are well-defined, independent components that are implemented and tested separately before amalgamation to form the complete desired software system. This leads to better maintainability and allows for the work to be easily divided where necessary. It also makes the components reusable wherever similar needs arise in other designs.
2. **Integrability:** This follows from above (logical level) and appropriate code structure; where the design of the package allows for integrating different algorithms in a simple manner.
3. **Extensibility:** The package should be structured to permit easy addition of new code and data without drastically altering the underlying architecture.
4. **Functionality:** All algorithms needed for successfully acquiring the 2D polar Fourier transform should be implemented in the most effective order to achieve

the "*simplest*" possible closed form result. It is important to note however, that there is a trade-off between simplicity of the output and performance.

5. **Performance:** The implementation of selected algorithms, including new ones, should be efficient.

4.1.2 Architecture design

The toolbox is designed to be modular. This implies that the different components that make up the toolbox are implemented and tested independently of each other before they are integrated to form a whole. This is a natural extension of the way the two dimensional Fourier transform in polar coordinates is approached logically.

In this work, as discussed in the preceding chapter (see Chapter 3, section (3.1.3.1)), the full 2D Fourier transform is obtained in three steps: finding the Fourier series expansion in the angular variable + finding the *n*th order Hankel transform of the *n*th coefficient in the Fourier series and appropriately scaling the result + obtaining the inverse Fourier series. Each step therefore consists of a procedure that is implemented and tested separately. The inverse 2D polar Fourier transform is seen to consist of the same components i.e. Forward Fourier series, Hankel transform and Inverse Fourier series.

There are other procedures that are also implemented alongside the afore-mentioned procedures that are very useful operations for the full functionality of the toolbox. Some additional procedures are also included for completeness. The breakdown of the toolbox is given later in this section.

Along with these transforms and operations are *lookup* tables that enhance the simplicity of some often well known results. The concept of the application lookup tables is drawn from the way we often solve integral transform problems by hand. There often exist tables of these transforms for some basic functions which one only needs to refer to where needed. *Lookup* tables, the details of which are provided in later chapters, do just that. They are made up of pairs of associated items whose first column consists of a list of functions and second column has their associated specified transforms. Lookup tables

thereby serve as the first line of action prior to attempting to directly evaluate the transform.

The breakdown of the SCAToolbox:

The SCAToolbox is broken into sub packages allowing it to be easy to use and maintain. The code structure is made up of four main sections: 1) creating the toolbox, 2) supporting functions, 3) integral transforms and 4) testing and verification.

The first section is discussed in detail in section 4.3 of this chapter. Creating the toolbox is system dependent i.e. design and syntax depend on the CAS software being used.

The supporting functions are those operations that do not constitute the core of the toolbox but are vital for the complete and effective functionality of the system. These supporting functions help manage other structures within the toolbox. There are two types of supporting functions in this toolbox.

The first type consists of procedures that perform operations on functions. All the procedures in this group are convolutions which are one of the major reasons why transformations are done from one space to another. The convolutions implemented in this toolbox include one dimensional and two dimensional convolutions in Cartesian coordinates, angular or circular convolution, radial convolution, two dimensional convolution in polar coordinates and series convolution. Making these easily accessible and usable, a "Bracket" convolution is implemented. This makes for less error when trying to evaluate a convolution since one only need know the command line for one procedure as opposed to six different ones. These convolutions are discussed in great detail in the section (5.1) of Chapter 5. Convolutions can be stand alone procedures and as such are not placed in the *IntegralTrans* package of the SCAToolbox. Using these operations therefore does not require loading the *IntegralTrans* package; downloading and loading the SCAToolbox is sufficient.

Procedures that help manage/manipulate other structures in the toolbox form the other type of supporting functions. These being rather intricate to the inner workings of

the integral transforms are stored in the *IntegralTrans* package of the SCAToolbox. Among these procedures are the *takeAlook* procedure, *addToTable* procedure, *Ekronecker* procedure and *EdDirac* procedure.

The *takeAlook* procedure manipulates tables by comparing the entered function with the list of functions in the first column of the *lookup* table of interest and returning the mapping result in the second column of said table. A process named *pattern matching* is used to compare the functions. The entered function has particular properties (arithmetic and variable types, etc.) that are matched to a fitting pattern in the table. Details of this procedure can be found in section (5.2.1) of Chapter 5.

With the *lookup* table comes the need to be able to expand the existing list of functions for which the desired transform is known. This is accomplished by the implementation of the *addToTable* procedure. This procedure adds the expression and its corresponding transform to the specified transform table and makes extending an already existing table quite simple.

The *Ekronecker* procedure provides the Kronecker delta function given the appropriate variables. The need for this operation arose when the built-in Kronecker function did not work in procedures written in the toolbox.

As discussed in section (7.1.2) of Chapter 7, the integral of a shifted Dirac function is halved at the end points. In this work however, the result at the endpoints is not to be halved. The *EdDirac* procedure redefines the built-in Dirac function so that the result of that integral produces the desired results that match the theoretical development on which this toolbox is based.

The integral transforms implemented in this toolbox include the forward Hankel transform of *n*th order and its inverse, the forward and inverse one dimensional Fourier series transform, and the forward and inverse two dimensional Fourier transform in polar coordinates. The next section of the code structure deals with implementing these transforms. As already mentioned, there are lookup tables associated with these transforms and implemented as independent structures. These tables are called on in the

integral transforms as the first line of action before any rules are applied or attempts are made at direct evaluation. Having in these tables as many functions as possible is very useful; making evaluation of these transforms rather straightforward.

These transform procedures also possess what is referred to in Maple as *remember tables*. The *remember* option is specified in the options field of the procedures; so that an entry is made in their respective remember tables at the end of each call of the specific procedure. Remember tables record the result for the specified arguments so that subsequent calls to the procedure with the same arguments simply retrieve the result from the remember table instead of re-calculating a result that has already been calculated. The *system* option identifies procedures for which their remember tables may be deleted at garbage collection time.

There are rules that apply to these transforms under specific conditions, for example the convolution-multiplication rule that says that a convolution in one domain is equivalent to a multiplication in the transform domain. These rules, as discussed in Chapter 3 and Chapter 7, are implemented within the transform procedures. The conditions are used to help decide when and where a rule should be applied.
There are different rules that are applicable to transforms in SCAToolbox. In the one dimensional Fourier series transform, the rule for Dirac delta functions, complex exponentials, scaling, shifting, modulation and convolution-multiplication are implemented whereas the two dimensional Fourier transform in polar coordinates has the rule for Dirac delta functions, complex exponentials, shifting (from the 1D Fourier series due to the modular nature of the code) and convolution-multiplication. When all else fails, the transforms are evaluated directly by integration.

The Hankel transform and 1D Fourier series are implemented as outlined above. The 2D Fourier transform in polar coordinates is implemented in two ways. The first way is via the two preceding transforms. Here, it draws on the modular nature of the code to implement the transform as specified in section (3.1.4) of Chapter 3. This method is part of the core foundation of this work and involves breaking the 2D polar Fourier transform

into three steps: 1) finding the *n*th Fourier series coefficient, 2) then finding the *n*th order Hankel transform of the *n*th series coefficient and scaling the result appropriately and 3) lastly obtaining the inverse Fourier series (summation). Being able to properly implement and test the two preceding transforms is therefore vital to its successful implementation.

The second way the 2D Fourier transform in polar coordinates is implemented in the toolbox is by direct evaluation. This is done so that the two approaches can be more adequately compared and contrasted. Section (3.1.3) of Chapter 3 discusses how the direct evaluation is carried out.

After all the integral transforms and their tables as well as the supporting functions have been implemented, the toolbox is tested and the results verified against known data for verification. The results for both ways of implementing the two dimensional Fourier transform in polar coordinates are not only tested against existing data but also compared with each other and conclusions are drawn.

4.2 CAS Software

AS previously discussed, Computer Algebra Systems (CAS), having evolved from theoretical physics and research in artificial intelligence are useful for solving problems in a wide range of applications from communication systems (in Digital Signal Processing, Coding and Cryptography) to applied mechanics, fluid and control analysis, and biological systems.

The CAS software used in this work is Maple, an interactive system developed in Waterloo, Canada for algebraic computation while incorporating an extensive set of mathematical knowledge in its libraries.

Maple is useful in the mathematical computations involving integers, rational numbers, sets, polynomials, equations, inequalities, derivatives, infinite integrals, etc. Maple possesses key features such as compactness that generally reduces the basic computer memory requirements to a few hundred kilobytes per user as opposed to the few thousand kilobytes needed for other CAS software such as MACSYMA and

REDUCE, and portability that enables its application on a variety of different operating systems. Maple is also inherently extensible given that the system's library functions and the user's functions are equal in status [10].

Hashing is widely used in Maple. It ensures that equivalent objects are not stored multiple times and also helps minimize execution time since it can remember important operations as desired. Only when specific functionality for a calculation is required by a user, does Maple automatically load the associated files to be interpreted into the main memory thereby saving the cost of having necessary code located in main memory.

4.3 Creating a toolbox

Toolbox often referred to as "library" or "package" can be defined as a collection or set of tools for some specific function. It can be used as a way to organize tools into a concise unit for easy access and usability. Here, the use of the term "toolbox" is to differentiate the complete unit from its parts i.e. in this body of work, the complete library is called a *toolbox*; the toolbox contains in itself other sub-packages termed *packages* which may contain *tables*, *procedures*, *operations* among others. Figure 2 below gives a general pictorial idea of this system.

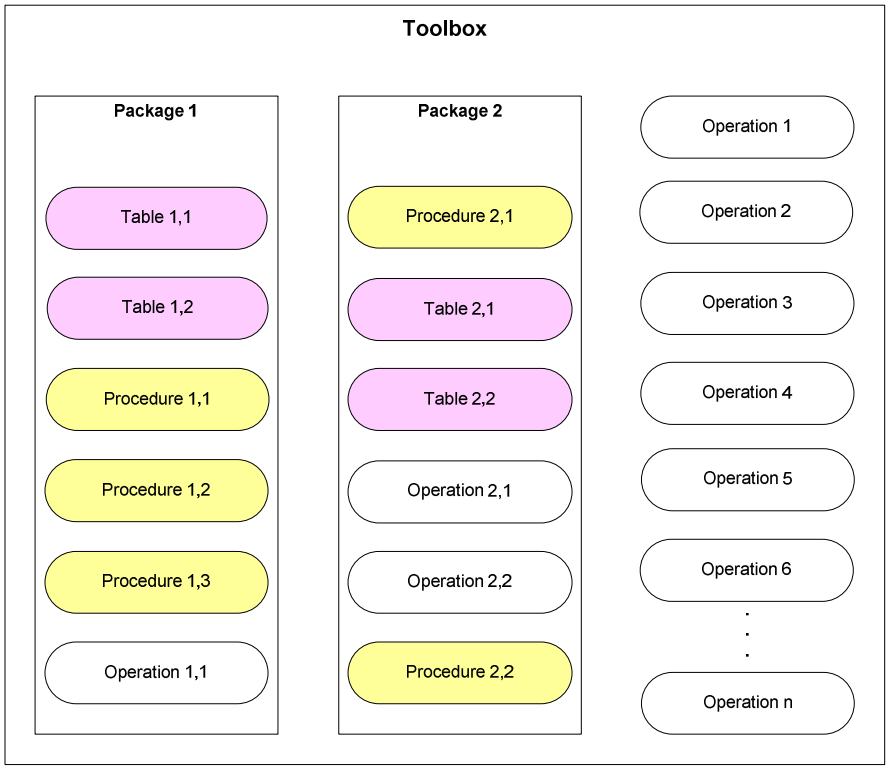


Figure 2: Pictorial representation of a general Toolbox

The toolbox created in this thesis is called *SCAToolbox*. It is a symbolic computer algebra toolbox that provides a comprehensive collection of interactive tools suitable for computing mainly the Fourier transform of expressions in two dimensional polar coordinates. SCAToolbox consists of several procedures (term used for "functions" in Maple) and operations as well as tables. One of the packages in SCAToolbox is named *IntegralTrans* package. This package contains the procedures, tables and operations necessary for computing a few well known and perhaps some not-so well known integral transforms. This package and other operations the toolbox contain will be discussed in later sections but for now how to build a generic toolbox is introduced.

As previously mentioned, the CAS software used in this work is Maple but the process of building a toolbox can be generalized to any CAS system. It is important to ensure that the software of choice or the appropriate software is properly installed on your computer.

The process of building a toolbox or uploading an existing one is outlined in detail in Appendix B of this thesis.

Chapter 5 : Supporting Functions

This chapter discusses the definitions, syntax and Maple structure for operations and procedures used in the toolbox or for manipulating other structures within the toolbox. The first section discusses Maple operations and the second section discusses Maple procedures required for the toolbox. A subsection is devoted to each separate operation and follows the same structure of mathematical definition, syntax and Maple structure. This similarity in subsection structure is done so that each subsection may be referred to independently as needed and the reader may therefore jump to whichever subsection is of interest. A summary of the supporting functions is given at the end of the chapter.

5.1 Operations on Functions

In this body of work, the manipulations considered all happen to be convolutions and as such a "Bracket" convolution is written to have a collective identifier. The different types of convolutions that make up this "Bracket" convolution include

1. One dimensional Cartesian convolution

2. Two dimensional Cartesian convolution
3. Two dimensional Polar convolution
4. Angular convolution
5. Radial convolution
6. Convolution of Infinite series

As their names suggest, these convolutions depend on different function types and so have different mathematical rules that apply to their evaluation, as will be seen in this chapter. Each of the six convolutions, along with the general bracket convolution, is discussed in turn in its own subsection.

5.1.1 "Bracket" Convolution

The "Bracket" (i.e. general) convolution allows for a uniform way of computing convolutions in this work. Having to remember a single syntax for all convolution types makes it simpler and more functional for the user. It also makes identifying convolutions in other procedures very easy since one only need recognize a particular syntax as opposed to multiple syntaxes for various convolution types. The different convolution types are distinguished one from the other by the *condition* indicated within the "bracket" convolution call.

5.1.1.1 Calling sequence

The inputs to this procedure thus include the two functions to be convolved, a list of variable(s) on which they both depend and the condition that specifies the type of convolution. The acceptable convolution types include "*1dcartesian*", "*2dcartesian*", "*2dpolar*", "*angular*", "*radial*" and "*series*". Obviously, this list is extendable to include other desired convolution types. The complete syntax for calling this operation is therefore: "*Conv(function 1, function 2, [list of variables], condition)*", where the condition is the desired type of convolution to be performed from the previously mentioned list. The output of this procedure is the result of the specified convolution type.

5.1.1.2 Implementation

This operation will call and evaluate the appropriate convolution procedure based on the condition entered. As seen from the syntax of this procedure, the first and second arguments of this procedure are the functions to be convolved, with the third being the list of variables on which they depend and the fourth being the condition. The procedure first renames the variable(s) in the list provided as the third argument as separate individual variable(s) and then uses those along with the other inputs to call the desired convolution type. This is to ensure that the appropriate type of convolution is called with the correct variables. For example, to call the two dimensional polar convolution procedure from the "Bracket" convolution, the list of variables have to be "broken up" into separate variables i.e. the call should be *"TwoDPolarConv(function 1, function 2, variable 1, variable 2)"* and NOT *"TwoDPolarConv(function 1, function 2, [variable 1, variable 2])"*.

It is important to note that since this general convolution operation is useful beyond the IntegralTrans package, it is not stored in the package but rather in the SCAToolbox and can therefore be utilized without loading the package first. This saves on processing time and memory. The flowchart of the "Bracket" convolution can be found in Figure 5.1.

5.1.1.3 Examples

```
> Conv(2, Dirac(s), [s], 1 dcartesian)
```

$$2$$

```
> Conv(a, Dirac(s), [s], 1 dcartesian)
```

$$a$$

```
> Conv(a, Dirac(x)·Dirac(y), [x, y], 2 dcartesian)
```

$$a$$

```
> Conv(Dirac(y), Dirac(x), [x, y], 2 dcartesian)
```

$$1$$

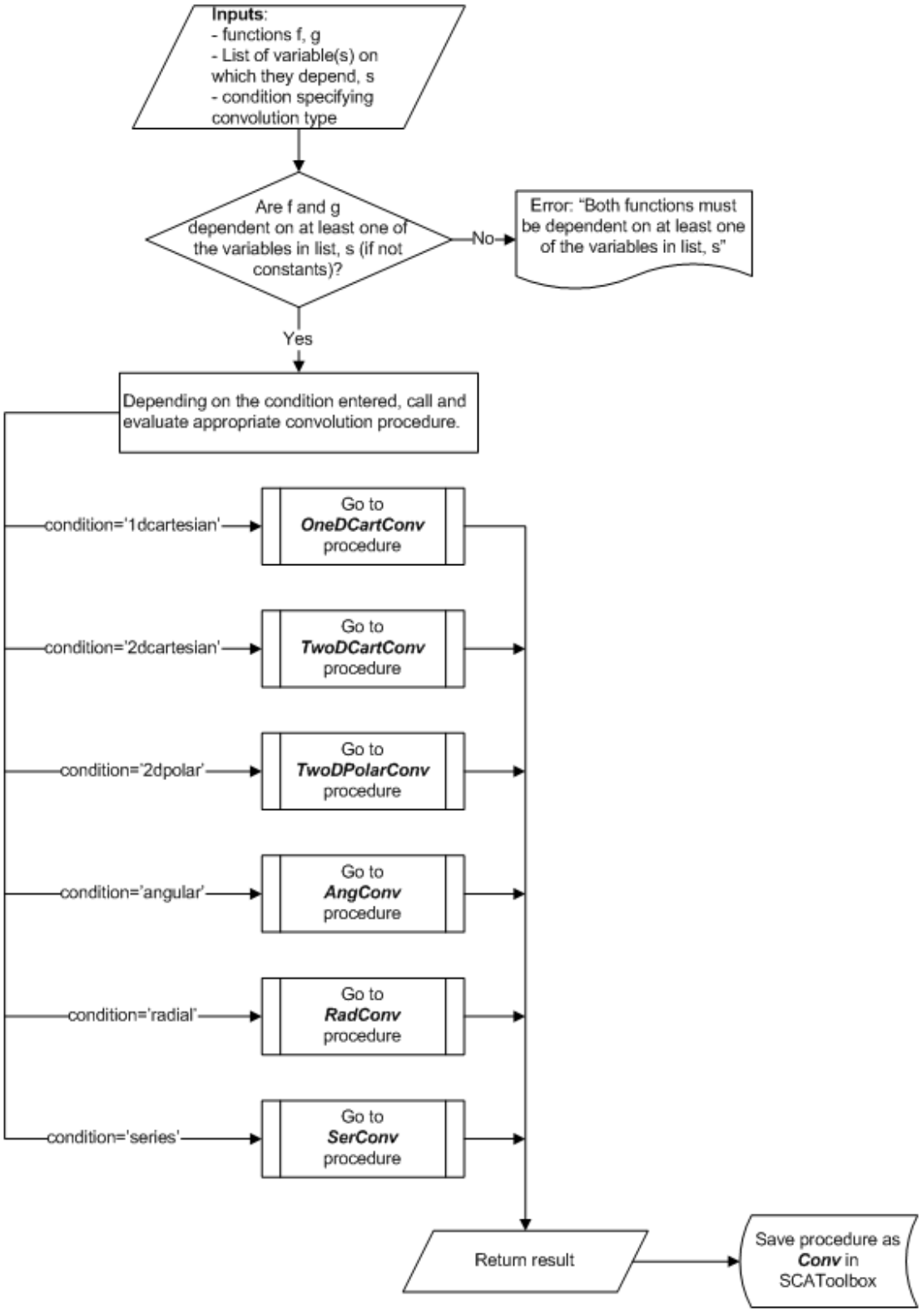


Figure 5.1: Flow chart of the "Bracket" convolution operation

5.1.2 1D Cartesian convolution of functions

5.1.2.1 Mathematical definition

The one dimensional Cartesian convolution of two functions is defined mathematically as

$$f(t) * g(t) = \int_{-\infty}^{\infty} g(t_0) f(t - t_0) dt_0 \tag{5.1}$$

where t is the variable on which the functions, f and g depend and t_0 is the dummy variable.

5.1.2.2 Calling sequence

The inputs to this procedure include the two functions to be convolved and the variable on which they both depend. The complete syntax for calling this operation is therefore: "*OneDCartConv(function 1, function 2, dependent variable)*".

The output of this procedure is the result of the one dimensional Cartesian convolution.

5.1.2.3 Implementation

This is an operation that performs the one dimensional Cartesian convolution of two functions. This convolution involves shifting the dependent variable by a dummy variable for one of the functions entered and multiplying it by the other function after making the latter a function of the dummy variable. The product is then integrated with respect to the dummy variable over the interval [-∞, ∞].

In order to change the dependent variable within a procedure in Maple, one must realize the subtle differences between the commands "*subs*" and "*apply*" for example. Using *"subs"* in Maple allows for replacing a variable with a substitute whereas *"apply"* makes one variable dependent on the other. This brings up the issue of an 'operator' or a 'procedure' versus a 'function'.

An 'operator' in Maple is the function name of what is referred to as a *functional operator*, a special form of procedure written using the *arrow* notation e.g. $f := x \rightarrow 3x + 5$. The hard assign, ":=" can also be used to produce this notation. A 'function' is an expression sequence of zero or more items [104] e.g. $f()$ or $f(x, y, z)$,

whereas a ‘procedure’ is the name of the procedure of a function as known in programming circles. A ‘function’ is therefore not the same as popularly considered i.e. it is not a relationship from which results can be obtained by changing the dependent variable(s) but just an expression sequence e.g. $f(x, y, z)$ cannot be assigned a function expression to evaluate for different values of x, y or z. With all this in mind, “*subs*” is used so that the dummy variable simply replaces the inputted variable instead of wrongfully making it a function of the variable. Subtleties like this often arise and must therefore be taken into consideration when using Maple, and all other CAS systems, to ensure such seemingly minor yet dangerous mistakes are avoided.

As discussed in the “Bracket” convolution case, the one dimensional Cartesian convolution operation is stored outside the IntegralTrans package within the SCAToolbox. The flowchart of the one dimensional Cartesian convolution can be found in Figure 5.2.

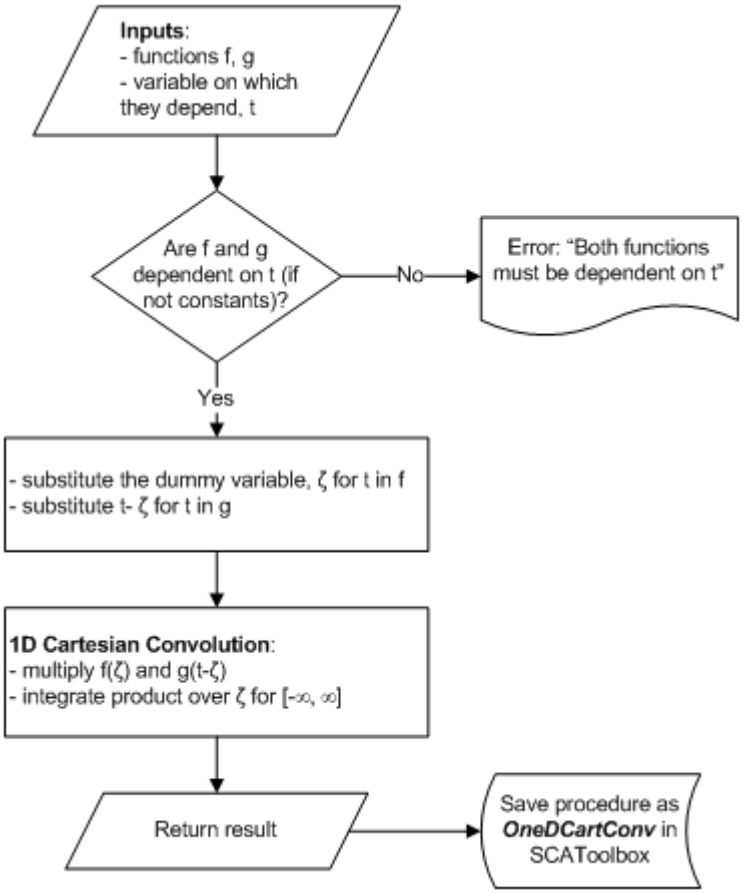


Figure 5.2: Flow chart of 1D Cartesian convolution operation

5.1.3 2D Cartesian convolution of functions

5.1.3.1 Mathematical definition

The two dimensional Cartesian convolution of two functions is mathematically defined below.

$$f(x,y)**g(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x_0,y_0)f(x-x_0,y-y_0)dx_0dy_0 \tag{5.2}$$

where x and y are the variables on which the functions, f and g depend and x_0 and y_0 are dummy variables.

5.1.3.2 Calling sequence

Inputs to this procedure include the two functions to be convolved in Cartesian coordinates and the two variables on which they both depend. The complete syntax for calling this operation is therefore: "*TwoDCartConv(function 1, function 2, variable 1, variable 2)*". The output of this procedure is the result of the two dimensional Cartesian convolution.

5.1.3.3 Implementation

The operation described in this section performs the two dimensional Cartesian convolution of two functions. In a two dimensional Cartesian convolution, dummy variables are substituted for both dependent variables that the procedure accepts as inputs in one of the functions entered. In the other function, the dependent variables are shifted by their respective dummy variables and then both functions are multiplied. The product is integrated over the interval $[-\infty, \infty]$ with respect to both dummy variables.

Similar to the one dimensional Cartesian convolution case, this operation is stored outside the IntegralTrans package within the SCAToolbox. The flowchart of the two dimensional Cartesian convolution is shown in Figure 5.3.

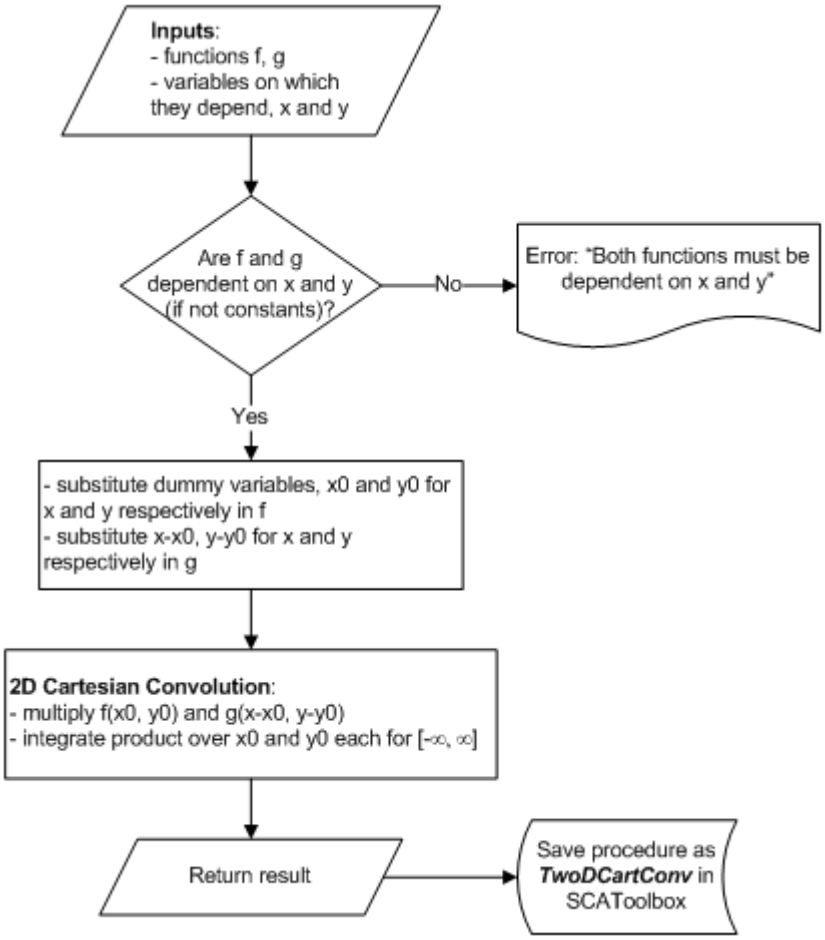


Figure 5.3: Flow chart of 2D Cartesian convolution operation

5.1.4 2D Polar convolution of functions

5.1.4.1 Mathematical definition

The two dimensional polar convolution of two functions is mathematically defined as

$$f(r,\theta)**g(r,\theta)=\int_0^{2\pi}\int_0^{\infty} g(r_0,\theta_0)f(r-r_0,\theta-\theta_0)r_0 dr_0 d\theta_0 \tag{5.3}$$

where r and θ are the variables on which the functions, f and g depend, and r_0 and θ_0 are dummy variables.

5.1.4.2 Calling sequence

Inputs to the 2D polar convolution procedure include the two functions to be convolved in polar coordinates and the radial and angular variables on which they both depend. The complete syntax for calling this operation is: "*TwoDPolarConv(function 1, function 2, radial variable, angular variable)*".

The output of this procedure is the two dimensional convolution of the two functions in polar coordinates.

5.1.4.3 Implementation

The 2D polar convolution operation described here evaluates the two dimensional convolution of two functions in polar coordinates. In the two dimensional convolution in polar coordinates, the dummy variables, r_0 and θ_0 are substituted for the dependent variables, r and θ respectively; inputted in one of the functions entered. In the other function, the dependent variables are shifted by their appropriate dummy variables. Both functions are then multiplied with the radial dummy variable. The product is integrated over the interval [0, ∞] with respect to the radial dummy variable and then over the interval [0, 2π] with respect to the angular dummy variable.

The 2D polar convolution operation is stored outside the IntegralTrans package within the SCAToolbox as in the aforementioned convolution types and for the same reasons. The flowchart of the two dimensional polar convolution is shown in Figure 5.4.

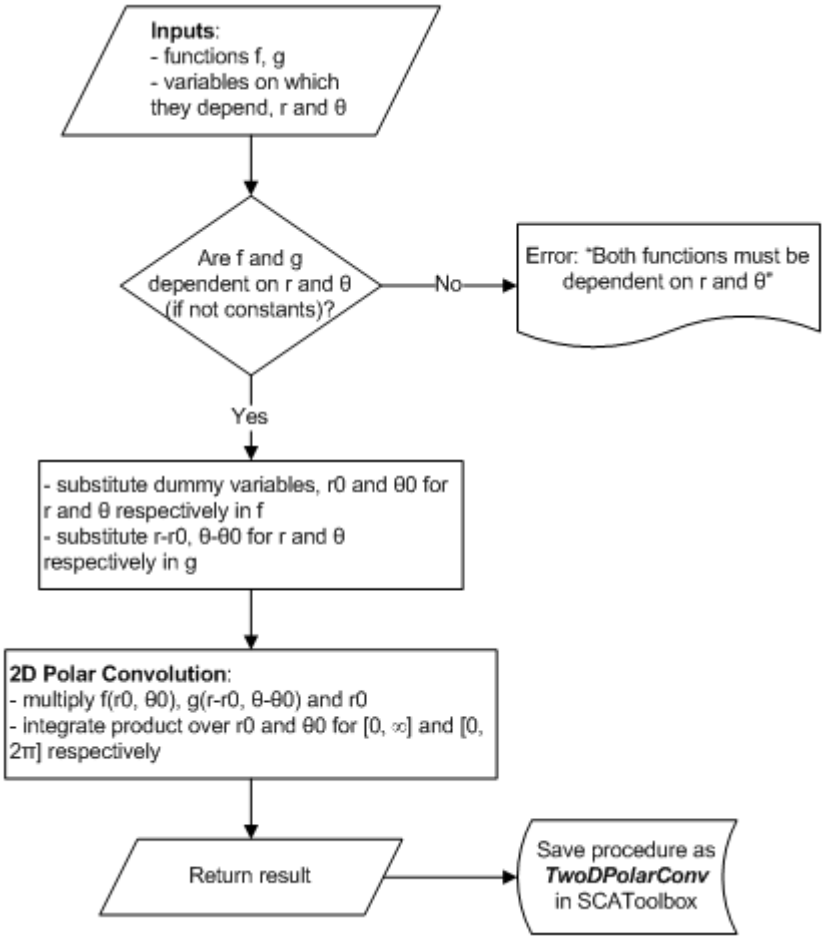


Figure 5.4: Flow chart of 2D polar convolution operation

5.1.5 Angular convolution of functions

5.1.5.1 Mathematical definition

The angular convolution of two functions is defined mathematically as

$$f(\theta) * g(\theta) = \frac{1}{2\pi} \int_0^{2\pi} g(\theta_0) f(\theta - \theta_0) d\theta_0 \tag{5.4}$$

where θ is the angle variable on which the functions, f and g depend, and θ_0 is the dummy variable.

5.1.5.2 Calling sequence

The inputs to this procedure are as follows: the two functions to be convolved and the angular variable on which they both depend. The complete syntax for calling this operation is thus: "*AngConv(function 1, function 2, angular variable)*".

The output of this procedure is the result of the angular convolution as indicated in Equation(5.4)

5.1.5.3 Implementation

The angular convolution operation computes the angular convolution of two functions. In this convolution, the dependent variable is shifted by a dummy variable for one of the functions entered and multiplied by the other function after making the latter a function of the dummy variable. The product is then integrated with respect to the dummy variable over the interval [0, 2π].

As discussed in previous convolution types, the angular convolution operation is stored outside the IntegralTrans package within the SCAToolbox since it could be useful outside the IntegralTrans package. The flowchart of the Angular convolution can be found in Figure 5.5.

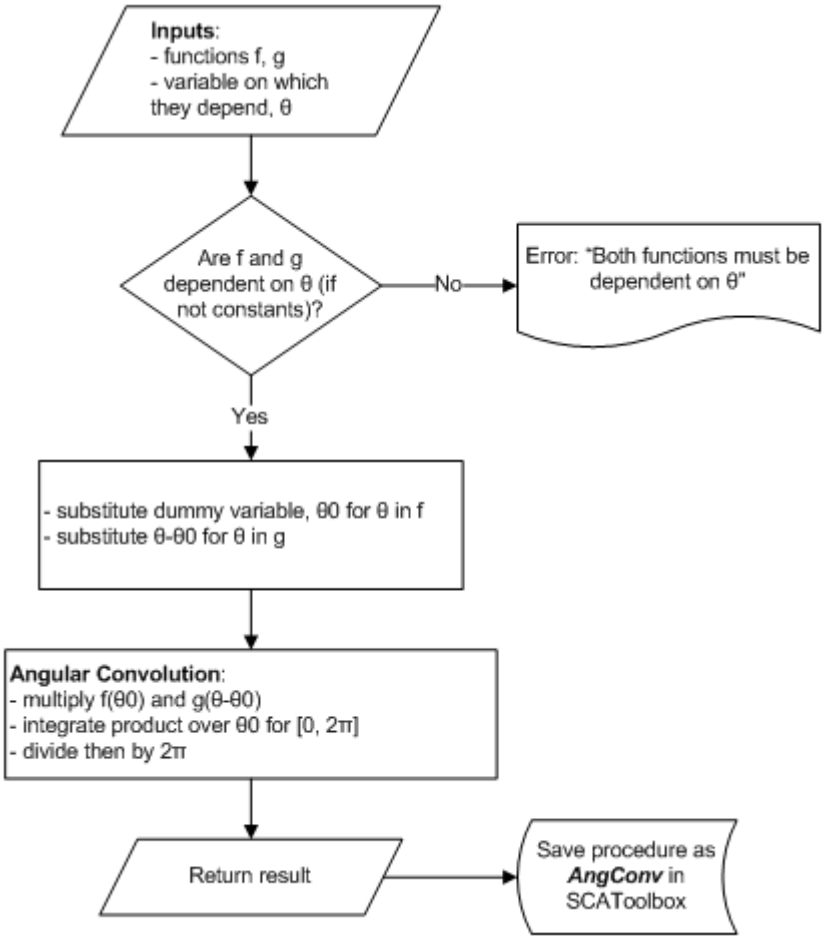


Figure 5.5: Flow chart of angular convolution operation

5.1.6 Radial convolution of functions

5.1.6.1 Mathematical definition

The radial convolution of two functions is defined mathematically as

$$f(r) * g(r) = \int_0^{\infty} g(r_0) f(r - r_0) r_0 dr_0 \tag{5.5}$$

where r is the radial variable on which the functions, f and g depend, and r_0 is the dummy variable.

5.1.6.2 Calling sequence

The inputs to the radial convolution procedure include the two functions to be convolved and the radial variable on which they both depend. The complete syntax for calling this operation is therefore: "*RadConv(function 1, function 2, radial variable)*".

The output of this procedure is the result of the radial convolution as indicated in Equation (5.5).

5.1.6.3 Implementation

This convolution operation evaluates the radial convolution of two functions. Here, the dependent variable is shifted by a dummy variable for one of the functions inputted and multiplied by the other function after making the latter a function of the dummy variable. The product is then integrated with respect to the dummy variable over the interval [0, ∞].

Similar to previously discussed convolution types, the radial convolution operation is stored outside the IntegralTrans package within the SCAToolbox. The flowchart of the radial convolution can be found in Figure 5.6.

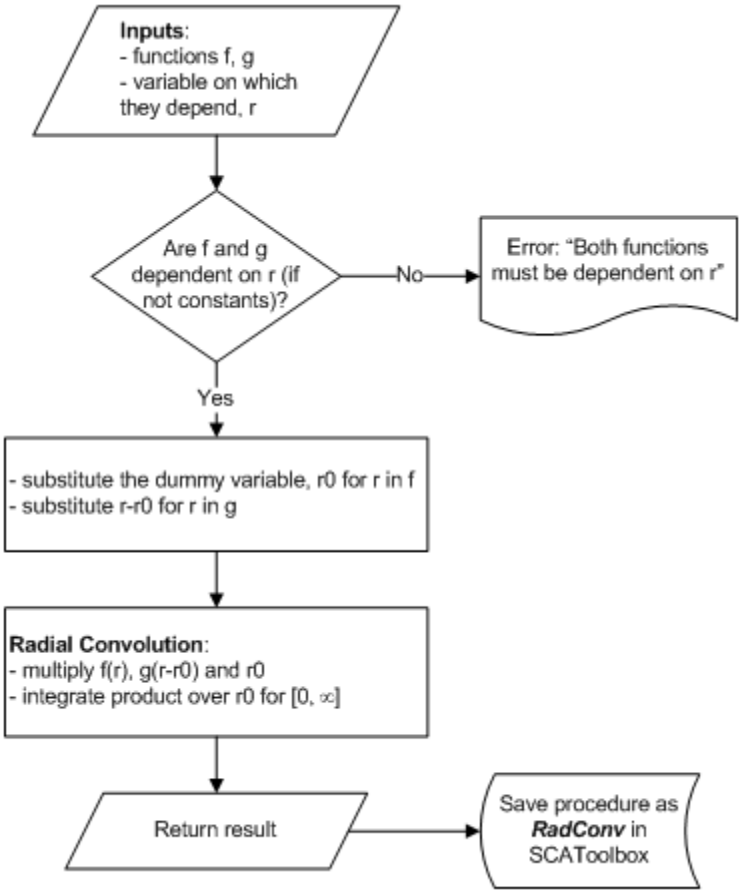


Figure 5.6: Flow chart of radial convolution operation

5.1.7 Convolution of Infinite series

5.1.7.1 Mathematical definition

The expression below shows the mathematical definition of the series convolution.

$$f_n * g_n = \sum_{k=-\infty}^{\infty} g_k f_{n-k} \tag{5.6}$$

where n is the variable (element index in series) on which the functions, f and g depend, and k is the dummy counter.

5.1.7.2 Calling sequence

The inputs to the series convolution procedure are two series functions to be convolved and the variable on which they both depend. The complete syntax for calling this operation is then: "*SerConv(series 1, series 2, dependent variable)*".

The output of this procedure is the result of the series convolution.

5.1.7.3 Implementation

The series convolution operation obtains the convolution of two infinite series. In the series convolution, the counter, k is substituted for the dependent variable, n given in one of the series functions entered. In the other series function, the dependent variable is shifted by the counter and then both series are multiplied. The product is summed over the interval [-∞, ∞] with respect to the counter.

As discussed in the other convolution types, the series convolution operation is stored outside the IntegralTrans package within the SCAToolbox. The flowchart of the convolution of infinite series can be found in Figure 5.7.

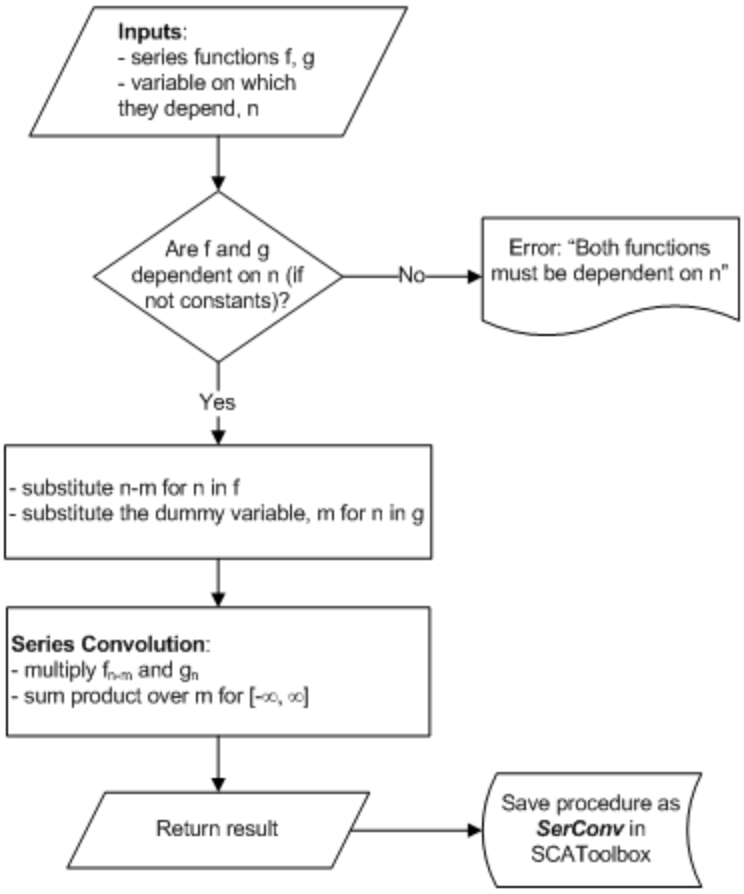


Figure 5.7: Flow chart of series convolution operation

5.2 Procedures in *IntegralTrans* package

Different procedures that are useful for managing procedures within the toolbox will be discussed in this section.

5.2.1 Procedure *takeAlook*

The first procedure, *takeAlook* is specific to manipulating tables. The inputs to this procedure are an expression and a table name. The table consists of a pair of items that are associated with each other, for example, functions and their corresponding Laplace transforms. The first column of a table would then consist of a list of transforms and the second column would consist of their associated Laplace transforms. The purpose of this procedure is thus to obtain the 'second column' result mapping of the 'first column' expression specified by the user. The expression entered by the user is compared to all

the entries in the first column of the existing table of interest and if it matches any of the patterns in the table, the corresponding expression in the second column of the table is returned, with appropriate substitutions made.

In order to match the expression with the patterns in the table in Maple, the built-in command "*tablelook*" is used. This command uses the idea of pattern matching given certain conditions to identify the appropriate pattern and make the necessary substitutions to ensure the right result is returned.

The *takeAlook* procedure, since it is rather intricate to the inner workings of the toolbox (as will be later seen) is saved in the IntegralTrans package within the SCAToolbox. As such, in order to use this procedure, one must load the package itself first. Figure 5.8 represents the flowchart for the procedure.

Examples

```
> takeAlook(A/r, HankelTable)
```

$$\frac{A}{\rho}$$

```
> takeAlook(1/(r^2+2^2), HankelTable0)
```

$$\text{BesselK}\left(0., \rho \sqrt{4}\right)$$

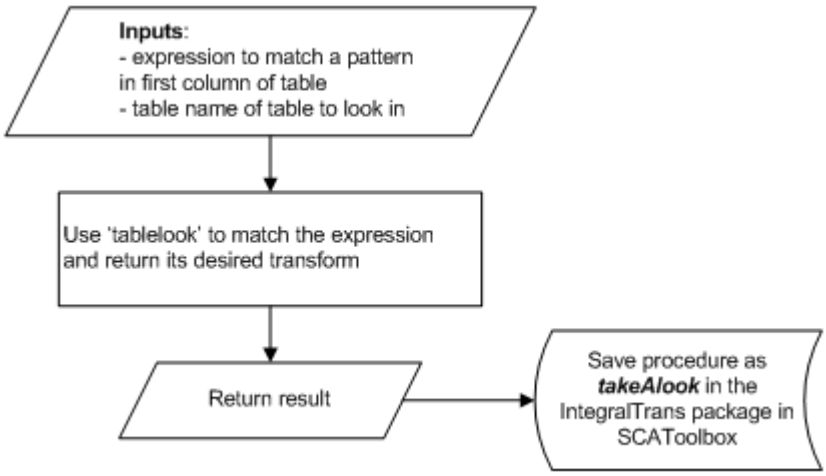


Figure 5.8: Flowchart of the *takeAlook* procedure

5.2.2 Procedure *addToTable*

The *addToTable* procedure that follows adapts the in-built Maple command "*insertpattern*" to help manage the toolbox. In Maple, the command "*insertpattern*" is used to add the expression and its corresponding transform to the specified transform table. This procedure makes it possible to add to an already existing table.
The inputs to the *addToTable* procedure include the name of the desired transform, the expression and its corresponding transform as well as the variable(s) on which they depend.

Checks must be put in place to ensure that if an unrecognized transform name is entered or the dependent variables do not match with the expression, that the procedure communicates this error to the user. This can be done by returning an error message or exiting the procedure or both. It is also a good idea to ensure there no duplicates i.e. that patterns that already exist are not reentered in the table.
The procedure is saved in the IntegralTrans package as well.

The flowchart for this procedure can be found in Figure 5.9. The flowchart agrees with the Maple code written for this procedure which does not permit indexed transform names.

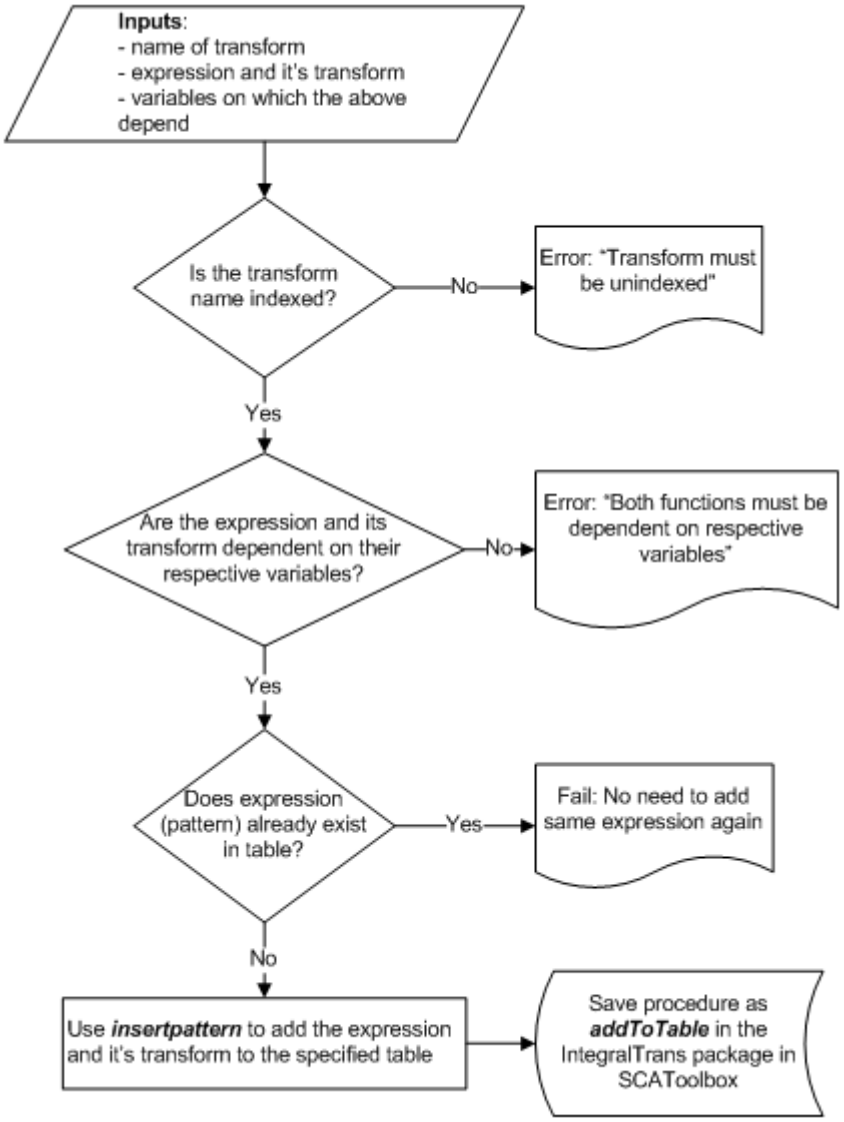


Figure 5.9: Flowchart of the *addToTable* procedure

5.3 Summary

The functions related to managing the toolbox as discussed above are summarized in Table 2. Let f and g be functions of x, ff and gg be functions of x and y, p and q be functions of r, u and v be functions of θ, and pp and qq be functions of r and θ.
Let s and w be series in n. Let *expr* be an arbitrary expression, *tablename* be name of table to consider, *procname* be name of procedure to consider, and *ans* be result that maps to *expr*.

Table 2: Summary of Supporting functions

Functions	Calling sequence	Description
"Bracket" convolution	*Conv(f, g, [x], 1dcartesian)* *Conv(ff, gg, [x, y], 2dcartesian)* *Conv(pp, qq, [r, θ], 2dpolar)* *Conv(u, v, [θ], angular)* *Conv(p, q, [r], radial)* *Conv(s, w, [n], series)*	obtain 1D convolution of *f(x)* and *g(x)* in Cartesian coordinates obtain 2D convolution of *ff(x)* and *gg(x)* in Cartesian coords. obtain 2D convolution of *pp(r, θ)* and *qq(r, θ)* in polar coords. obtain angular convolution of *u(θ)* and *v(θ)* obtain radial convolution of *p(r)* and *q(r)* obtain convolution of the infinite series, *s(n)* and *w(n)*
1D Cartesian convolution	*OneDCartConv(f, g, x)*	obtain 1D convolution of *f(x)* and *g(x)* in Cartesian coordinates
2D Cartesian convolution	*TwoDCartConv(ff, gg, x, y)*	obtain 2D convolution of *ff(x)* and *gg(x)* in Cartesian coords.
2D polar convolution	*TwoDPolarConv(pp, qq, r, θ)*	obtain 2D convolution of *pp(r, θ)* and *qq(r, θ)* in polar coords.
Angular convolution	*AngConv(u, v, θ)*	obtain angular convolution of *u(θ)* and *v(θ)*
Radial convolution	*RadConv(p, q, r)*	obtain radial convolution of *p(r)* and *q(r)*
Convolution of infinite series	*SerConv(s, w, n)*	obtain convolution of the infinite series, *s(n)* and *w(n)*
takeAlook	*takeAlook(expr, tablename)*	obtain what maps to *expr* in *tablename*
addToTable	*addToTable(procname, expr, ans)*	Add *expr*⇔ *ans* to table associated with procedure, *procname*

Chapter 6 :
Analysis of *IntegralTrans* package: Tables and Procedures

6.1 Concept

The concept used in this thesis is such that for transforms, instead of trying to evaluate them directly which may lead to indeterminate results, an effort is made to compile some tables of known transforms of basic functions. When an expression is entered for transformation, the first line of action would be to check the table of known transforms to see if the expression is one of the known ones. If it is, the result is picked up from that table, but if it is not then it is evaluated.

To be able to employ this concept successfully and efficiently, the idea of *lookup tables* is utilized. The compiled set of expressions is placed in a table which can be managed by operations described in section (5.2) of Chapter 5 to obtain the needed results. The table, as previously discussed, consists of a pair of items that are associated

with each other; the first column of a table consisting of a list of functions and the second column, their associated specified transforms.
Procedures are written to manipulate the tables and where the lookup tables are unsuccessful; the procedures would then make an attempt at evaluating the transform directly.

This concept helps prevent unnecessary computation of already known expression transforms thereby saving on memory space and processing time.

When all else fails, the output of transform procedures are written as they are entered except in the case of the Fourier Series and its inverse where the actual integral and sum respectively are returned. In Maple, the original call is returned as is by using single quotations whereas the actual sum or integral are returned by using their inert forms.

6.2 Hankel transform of *n*th order

As shown in Radially Symmetric functions section of Chapter 3, the Hankel transform of order zero is equivalent to the two dimensional Fourier transform of a radially symmetric function and naturally this order of Hankel is better known than any other. Existing tables of Hankel transform of order zero [105] are therefore not difficult to acquire and are rather sizeable whereas any other order, often seen as irrelevant, require more investigation to even obtain a few [105].

The concept of utilizing known data before attempting to evaluate transforms is used here. An extensive *lookup* table of the Hankel transform of order zero is gathered. This will consist of a mapping of as many functions as obtainable to their respective Hankel transform of order zero. The same is done for the *n*th order Hankel transform. This latter table is not as large as the former but comprises of the basics. Figure 6.1 gives the flowchart for creating these tables.

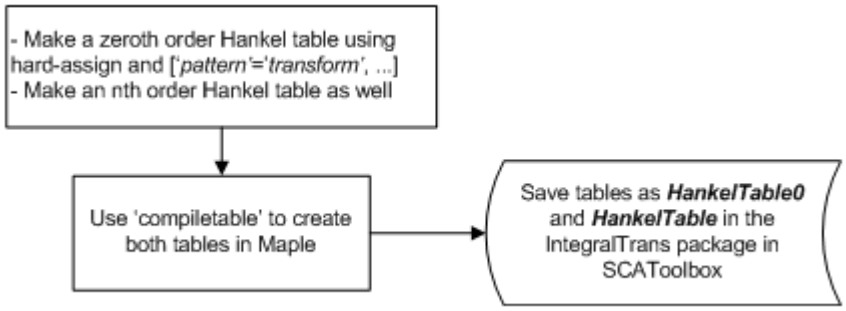


Figure 6.1: Flowchart for creating Hankel tables of zeroth and nth orders

The mapping of functions to transforms is achieved by using the hard assign operator within square brackets assigned to the table name i.e. "*table_name := [function := transform of function]*)". Creating the tables is accomplished in Maple by using the command line "*compiletable(table_name):*". There are other commands that can be used to manage a table. These commands include "*::*" to assign a type, "*patmatch*" to find out whether an expression matches a specific pattern and to obtain the values of any unidentified constants.

6.2.1 Forward Hankel transform (tables and procedure)

Mathematically, the Hankel transform of order *n* of a function, $f(r)$ is given by

$$\widehat{F}_n(\rho) = \mathbb{H}_n\left(f(r)\right) = \int_0^\infty f(r) J_n(\rho r) r dr \tag{6.1}$$

where $J_n(z)$ is the *n*th order Bessel function and the overhat indicates a Hankel transform as shown in equation (3.1) while *n* may be an arbitrary real or complex number [5].

The Hankel transform exists only if the Dirichlet condition is satisfied, that is if the following integral exists: $\int_0^\infty \left| r^{1/2} f(r) \right| dr$. The Hankel transform is particularly useful for problems involving cylindrical symmetry.

6.2.1.1 Calling sequence

The Hankel transform procedure accepts as its inputs: an expression, an equation or group of these, the variable on which these depend, the variable on which the transform depends and the order of the Hankel transform. As with all transformations of this caliber, a check must be made to ensure that the output is independent of the input variable.

6.2.1.2 Implementation

In order to obtain the Hankel transform of a group of expressions or equation, the procedure must be applied individually to each expression within the group or equation. This is executed in Maple by using the command "*map*" which parses the group of expressions or equation and applies the Hankel transform appropriately.
It is important to note that *lists* are allowed as the first argument instead of *sets* to obtain the result in the right order i.e. order of transform results corresponds with order of input.

For example, there exists three expressions, 2, $\frac{1}{r}$ and $\frac{3}{r}+1$ whose transforms are $2\frac{Dirac(s)}{s}$, $\frac{1}{s}$ and $\frac{3+Dirac(s)}{s}$ respectively. The group of expressions is entered as $\left[2,\frac{1}{r},\frac{3}{r}+1\right]$ and NOT $\left\{2,\frac{1}{r},\frac{3}{r}+1\right\}$ so that the transform can return $\left[2\frac{Dirac(s)}{s},\frac{1}{s},\frac{3+Dirac(s)}{s}\right]$ as opposed to $\left\{\frac{1}{s},2\frac{Dirac(s)}{s},\frac{3+Dirac(s)}{s}\right\}$ or $\left\{\frac{3+Dirac(s)}{s},2\frac{Dirac(s)}{s},\frac{1}{s}\right\}$, etc.

A *list* (which is ordered) is indicated in Maple by the use of square brackets "*[]*" whereas a *set* is indicated by curly brackets "*{}*".

Before doing anything else, the Hankel transform procedure needs to verify the order of the transform. Depending on the order, the appropriate table is called. If a non-zero order is entered, the procedure takes a look in the table of the *n*th order Hankel transform by comparing the expression to all the entries in the first column of the table. If it matches any of the patterns in the table, the corresponding expression in the second

column of the table is returned, after making appropriate substitutions as needed. If the transform cannot be found by matching in the lookup table, then an attempt is made to evaluate the transform directly via integration. In the case where an order of zero is entered, the procedure attempts to find a match to the expression in the table of the Hankel transform of order zero, and subsequently in the table of the *n*th order Hankel transform before trying to calculate the Hankel transform directly.

The *takeAlook* procedure described in section (5.2.1) of Chapter 5 is used to access and retrieve information from the Hankel transform tables. The Hankel transform procedure is stored in the IntegralTrans package in the SCAToolbox. The flowchart of the Hankel transform is given in Figure 6.2.

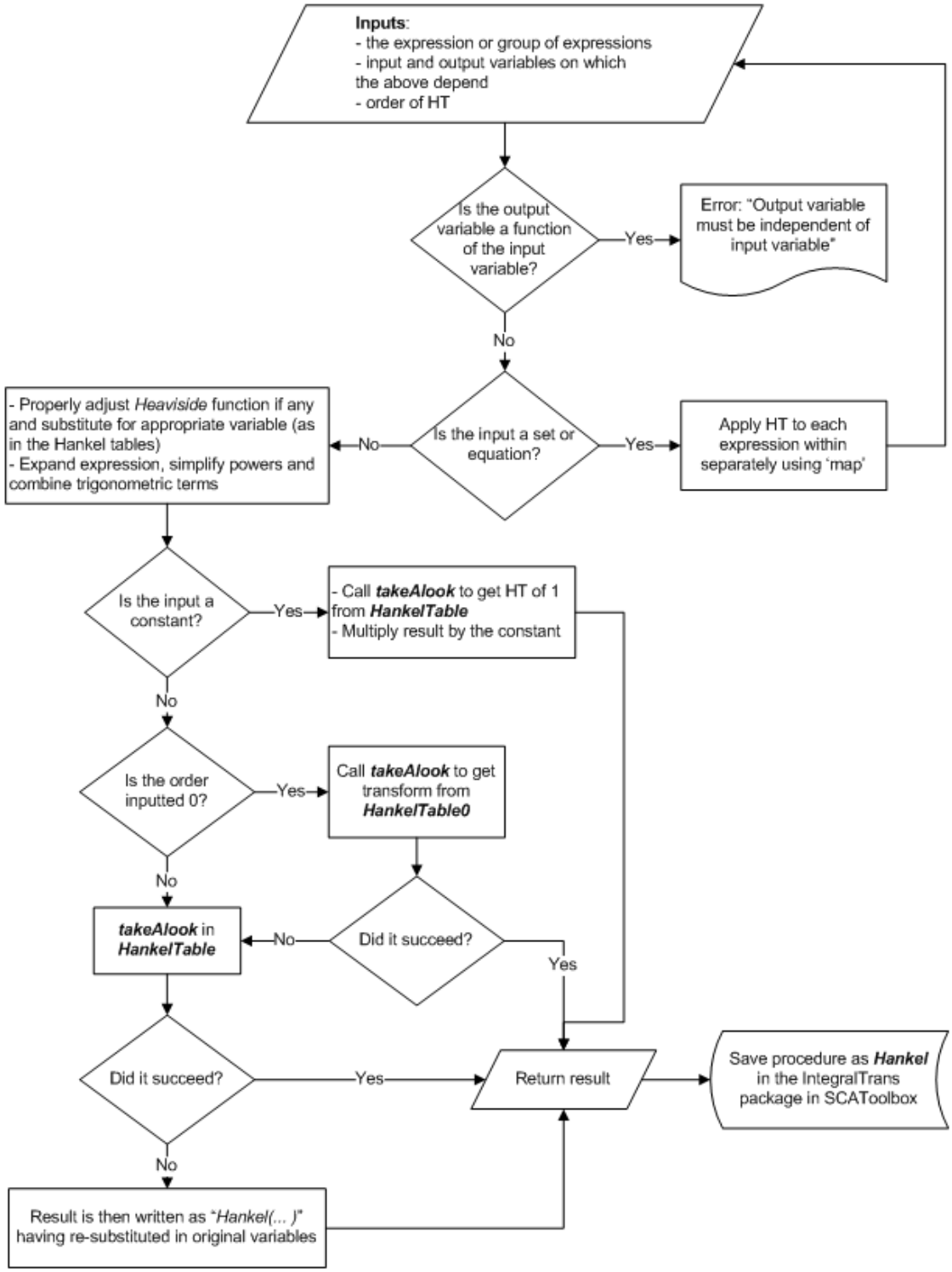


Figure 6.2: Flowchart of the Hankel Transform

6.2.2 Inverse Hankel transform (tables and procedure)

In order for an integral transform to be useful, it needs to be invertible thereby limiting the allowable values of *n.* Evaluating the Inverse Hankel transform is just as important as getting the Hankel transform. The Hankel transform is *self-reciprocating* meaning that the inverse of the Hankel transform is the Hankel transform itself [5]. Thus, the inversion formula is given by

$$f(r) = \int_0^\infty F_n(\rho) J_n(\rho r) \rho d\rho \tag{6.2}$$

where $J_n(z)$ is the *n*th order Bessel function and *n* may be an arbitrary real or complex number.

6.2.2.1 Calling sequence

The inputs needed for the Inverse Hankel transform are therefore the same as those for the forward Hankel transform- an expression, equation or group of these, variable on which these depend, variable on which the inverse transform depends and the order of the Inverse Hankel transform.

6.2.2.2 Implementation

A user would call the procedure as the inverse but internally the procedure will call the (forward) Hankel transform procedure. As described in the procedure for the forward Hankel transform, a check is made to ensure that the output variable does not depend on the input variable, and in the case of a group of expressions or equation, each expression within the group or equation is fed back individually into the procedure expression for evaluation. When the Hankel transform procedure is invoked without any success the result is the call as entered. The Inverse Hankel transform procedure is stored in the IntegralTrans package in SCAToolbox and its flowchart can be seen in Figure 6.3.

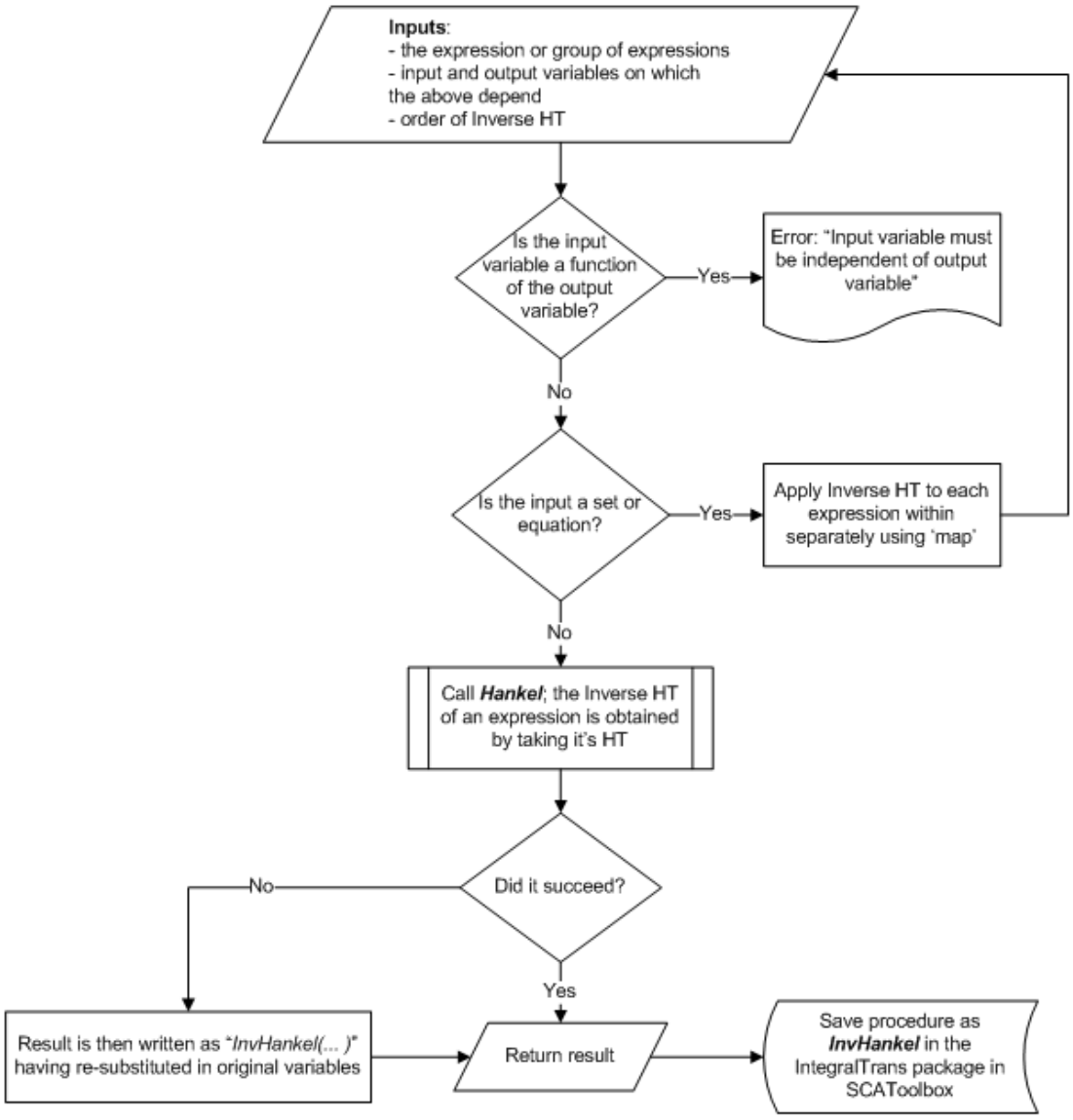


Figure 6.3: Flowchart of the Inverse Hankel Transform

6.3 Fourier series

Fourier series have often been thought of as an equality where a function is expressed as a sum i.e. $f(\theta) = \sum_{n=-\infty}^{\infty} f_n e^{jn\theta}$ under appropriate conditions on the function. However, here we consider the use of Fourier series as an operator or *transform* where the functions

$f(\theta)$ can be mapped to their Fourier series coefficients f_n. In so doing, the concept of *lookup tables* can be used for its implementation as has been done for the other transforms.

6.3.1 One dimensional Fourier series

The 1D Fourier series is considered in a similar way as a Fourier transform making use of the symbol $\mathbb{F}_S\{f(\cdot)\}$ to represent it. That is, equations (3.3) and (3.4) in Chapter 3 are re-interpreted as forward and inverse Fourier series *transforms*. Thus, the forward Fourier series transform is defined as the operation of finding the Fourier series coefficients given the function itself:

$$f_n = \mathbb{F}_S\{f(\theta)\} = \frac{1}{2\pi}\int_0^{2\pi} f(\theta)\, e^{-jn\theta} d\theta \tag{6.3}$$

Similarly, the inverse Fourier series transform returns the original function from the Fourier series coefficients and is given by

$$f(\theta) = \sum_{n=-\infty}^{\infty} f_n e^{jn\theta} \tag{6.4}$$

Hence f_n and $f(\theta)$ are a Fourier (series) pair: $f(\theta) \;\Leftrightarrow\; f_n$.

In light of this, the idea of *lookup tables* can be utilized to match up known functions and their corresponding Fourier series transforms. There are not as many functions to populate this table but this can be improved by applying the procedure to other functions and adding the results to the table as needed. Figure 6.4 gives the 1D Fourier series table's flowchart.

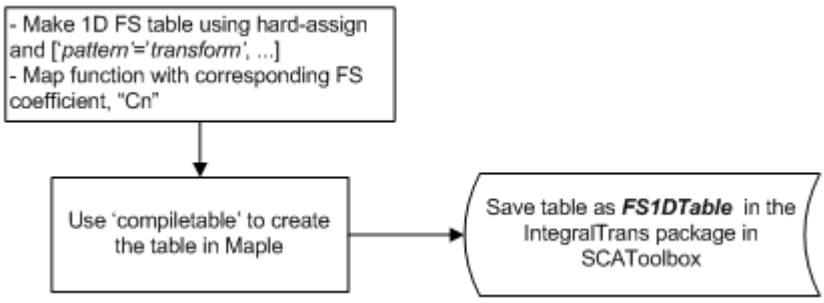


Figure 6.4: Flowchart of 1D Fourier series table

6.3.1.1 Calling sequence

The inputs admissible to the one dimensional Fourier Series transform procedure include: an expression or group of expressions, the dependent input variable, the variable on which the Fourier series of the aforementioned expression or group of expressions depend, the period of the Fourier series (written as a range) and the condition that specifies the output form.

Depending on the condition (5^{th} argument) entered by the user, the 1D Fourier series procedure returns the desired form of the result. The acceptable conditions include "*coefficientComplex*", "*coefficientReal*" and "*series*" outside of which an error message is obtained. For instance, if the user enters the "*coefficientComplex*" condition, the C_n is returned whereas if "*coefficientReal*" is entered the procedure returns the A_0, A_n, B_n and the full sum if "*series*" is entered. These coefficients or series are shown in detail in section (3.1.2) of Chapter 3.

6.3.1.2 Implementation

Fourier series can only be applied to periodic functions and as such whatever functions are entered in this procedure are assumed to be periodic. Care must therefore be taken to ensure that the expression or group of expressions entered best describes the desired function.

In this procedure, a check is made to ensure that the expression entered is dependent on the input variable and also that the fourth input, the period, is a range. Similar to the Hankel transform procedure, obtaining the Fourier series of a group of expressions means

applying the procedure to each expression within the group of expressions individually. Maple executes this by using its "*map*" command to parse the group of expressions and then appropriately applying the Fourier series.

To facilitate the calculation of the Fourier coefficients, the period of the Fourier series is evaluated as the difference between the upper and lower limits of the entered range.

There are particular cases in which the process of obtaining the Fourier series from the *lookup* table or by direct evaluation requires circumventing. Doing so allows for the application of rules that are discussed into more detail in a later chapter.
In order to identify these cases, certain queries are made to check for certain conditions for which these rules are activated. For example, the procedure checks for the existence of a convolution within the entered expression and if found, the subsequent steps are bypassed and the convolution rule (the theory of which is shown in detail in Chapter 3 and the implementation in Chapter 7) is applied.

A combination of "*op*" and "*nops*" commands is utilized to implement these checks in Maple. The "*op*" operator allows for obtaining specific operand(s) from a given expression. For example, the first operator (the multiplication) is extracted with the op and level 0 command:

```
> op(0, (x^2 + x)·exp(I·3·x))
```

```
`*`
```

The second 'operator here is the first term in the multiplication operation and is extracted with the op level 1 command:

```
> op(1, (x^2 + x)·exp(I·3·x))
```

$$x^2 + x$$

Continuing in a similar vein:

```
> op(2, (x^2 + x)·exp(I·3·x))
```

$$e^{3\,I\,x}$$

```
> op(0, op(1, (x^2 + x)·exp(I·3·x)))
```

```
`+`
```

> $op(1, op(2, (x^2 + x)\cdot\exp(I\cdot 3\cdot x)))$

$$3\ I\ x$$

The "*nops*" operator gives the number of operands in a given expression. For example,

> $nops((x^2 + x)\cdot\exp(I\cdot 3\cdot x))$

$$2$$

> $nops(f(x, y, z))$

$$3$$

> $nops([1, 2, 3, 4, 5, 6, 7])$

$$7$$

Other cases for which checks are made are the scaling, shift and modulation cases, so that appropriate rules are applied.

Outside of these special cases, the Fourier series *lookup* table is consulted. A result is obtained if a match is successful. If a successful match is not found, the procedure computes the Fourier coefficients for the exponential Fourier series, C_n as well as the real Fourier coefficients, A_0, A_n, B_n. The procedure also computes the complete Fourier series i.e. the sum of the product of the exponential Fourier coefficients, C_n and e^{inx} over n for [-∞, ∞].

The One dimensional Fourier series procedure is stored in the IntegralTrans package in SCAToolbox. The flowchart of this procedure is available in Figure 6.5.

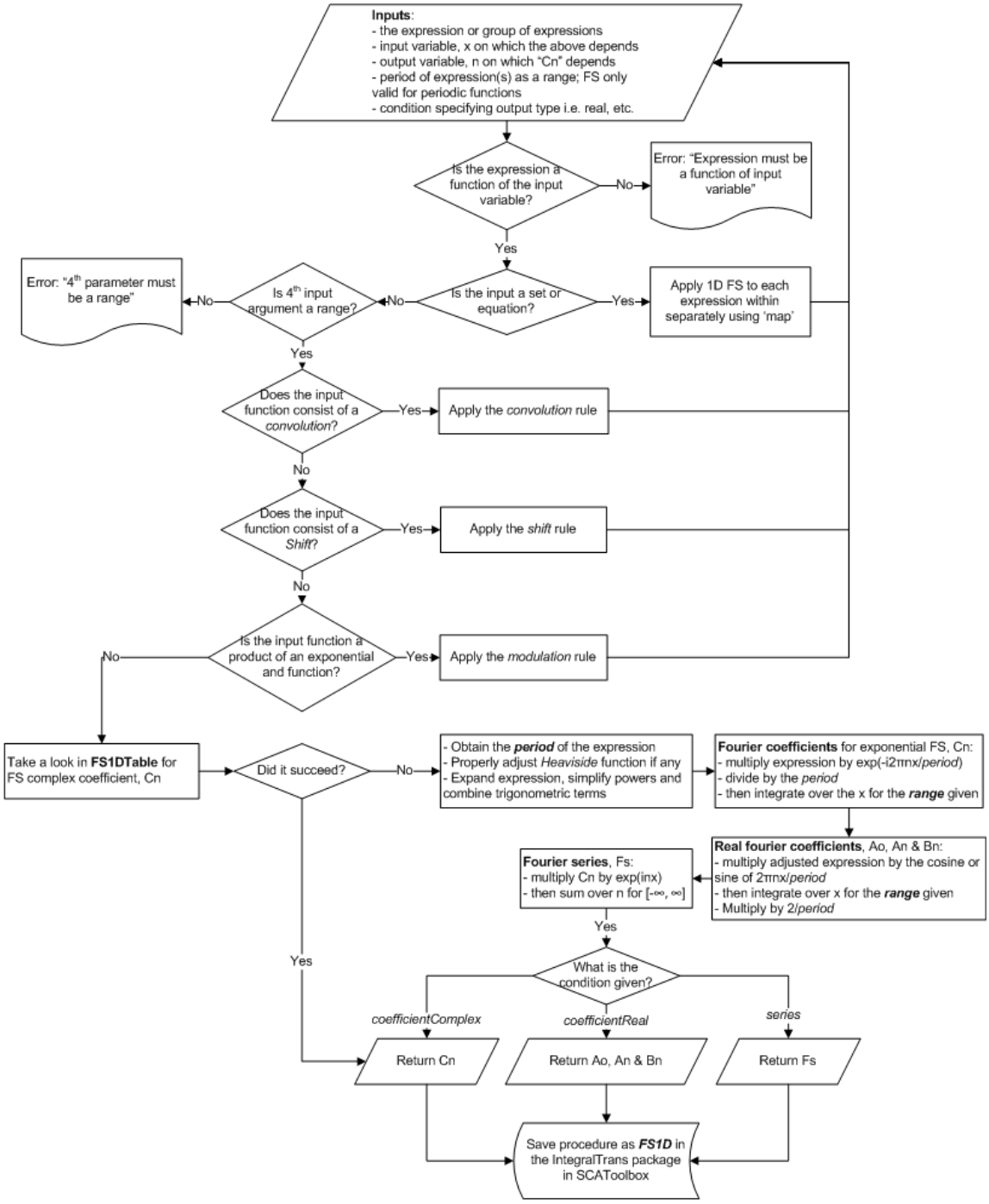


Figure 6.5: Flowchart of One dimensional Fourier series

6.3.2 The Inverse One dimensional Fourier series (table and procedure)

The inverse 1D Fourier series Transform is the one where the original function is returned from the coefficients by constructing the Fourier series itself:

$$f(\theta) = \mathbb{F}_S^{-1}\{f_n\} = \sum_{n=-\infty}^{\infty} f_n \, e^{jn\theta}. \tag{6.5}$$

Since some functions [103] can be expressed as a sum, they can be converted to Fourier series and the corresponding Fourier coefficients identified. It is therefore possible to use what is known before attempting to evaluate the Inverse Fourier series. The few known functions are put into a table by mapping the Fourier coefficient to the respective Fourier series sum. Figure 6.6 gives the table's flowchart.

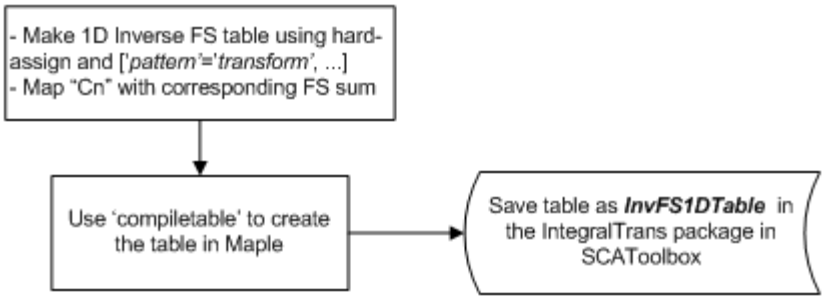


Figure 6.6: Flowchart of 1D Inverse Fourier series table

Creating the one dimensional Inverse Fourier series table is executed in Maple by using the command line "*compiletable(table_name):*". The Fourier coefficients are mapped to the Fourier series sums by using the "*hard assign*" operator within square brackets assigned to the table name such as: "*table_name := [Cn_pattern := FourierSeries_Sum]*".Other commands as previously discussed are used in managing the table.

6.3.2.1 Calling sequence

The needed inputs for the one dimensional Inverse Fourier series procedure include a Fourier coefficient or group of Fourier coefficients, variable on which these depend,

variable on which the resulting one dimensional Inverse Fourier series depends and the range that marks the period of the Fourier Series.

6.3.2.2 Implementation

First, a check is made to ensure that the Fourier coefficient entered is dependent on the input variable as in the case of the "forward" one dimensional Fourier series. Second, a check is made to see if the expression entered as a Fourier coefficient is a list; in which case the Inverse Fourier series is applied to each coefficient within the group individually. As seen before, Maple executes this by using the command, "*map*".

Similar to the forward Fourier series, a check is made for the existence of a convolution within the entered expression and if found, the subsequent steps are bypassed and the inverse convolution rule applied.

The procedure now looks into the one dimensional Inverse Fourier series table to get the needed information using the *takeAlook* procedure described in section (5.2.1 of Chapter 5. If the search is successful, the procedure returns this result but if it fails, the result will depend on whether or not the fourth parameter was inputted into the procedure. With a fourth input parameter, the result is obtained by summing the product of the Fourier coefficient and $e^{n.xi}$ over the range of n entered in the call sequence whereas without the fourth input parameter, the sum is made over an assumed range of [-∞, ∞]. The one dimensional Inverse Fourier series procedure is stored in the IntegralTrans package in the SCAToolbox and Its flowchart is given in Figure 6.7

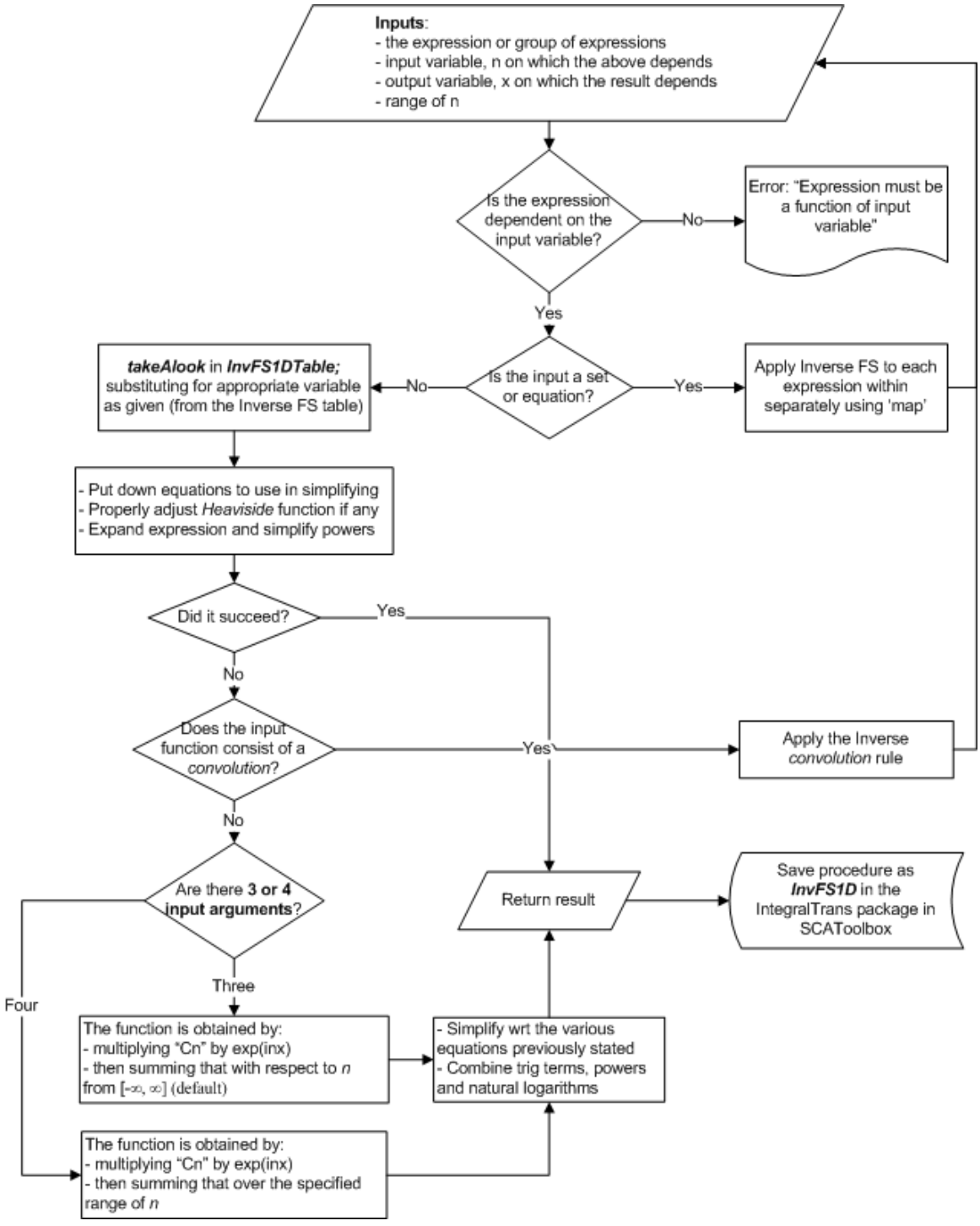


Figure 6.7: Flowchart of One dimensional Inverse Fourier series procedure

6.4 Two dimensional Fourier transform

The 2D Fourier transform of a function $f(x, y)$ (in Cartesian coordinates) is defined as

$$F\left(\vec{\omega}\right) = F(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, e^{-j(\omega_x x + \omega_y y)} dx\, dy\,. \tag{6.6}$$

The inverse Fourier transform is given by

$$f\left(\vec{r}\right) = f(x, y) = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\omega_x, \omega_y) e^{j\vec{\omega}\cdot\vec{r}}\, d\omega_x\, d\omega_y\,, \tag{6.7}$$

where the shorthand notation of $\vec{\omega} = \left(\omega_x, \omega_y\right), \vec{r} = \left(x, y\right)$ has been utilised.

As shown in section (3.1.3) of Chapter 3, the 2D Fourier transform $F\left(\vec{\omega}\right) = F\left(\rho, \psi\right)$ of a function in polar coordinates $f\left(\vec{r}\right) = f\left(r, \theta\right)$ is given by

$$F\left(\vec{\omega}\right) = \sum_{n=-\infty}^{\infty} 2\pi\, i^{-n}\, e^{in\psi} \int_0^{\infty} f_n(r) J_n(\rho r)\, r dr = \sum_{n=-\infty}^{\infty} F_n\left(\rho\right) e^{in\psi} \tag{6.8}$$

where

$$f_n(r) = \frac{1}{2\pi} \int_0^{2\pi} f(r, \theta)\, e^{-jn\theta} d\theta$$

The inverse 2D Fourier transform is given by (6.9)

$$f\left(\vec{r}\right) = \sum_{n=-\infty}^{\infty} \frac{1}{2\pi} i^n\, e^{in\theta} \int_0^{\infty} F_n(\rho) J_n(\rho r)\, \rho d\rho = \sum_{n=-\infty}^{\infty} f_n\left(\theta\right) e^{in\theta} \tag{6.9}$$

where

$$F_n(\rho) = \frac{1}{2\pi} \int_0^{2\pi} F(\rho, \psi)\, e^{-jn\psi} d\psi \tag{6.10}$$

The mapping from the Fourier series coefficients of f, $f_n(r)$ to the Fourier series coefficient of F, $F_n(\rho)$ is an *n*th order Hankel transform, and *not* a 2D Fourier transform i.e.

$$F_n(\rho) = 2\pi\, i^{-n}\mathbb{H}_n\{f_n(r)\}$$
$$f_n(r) = \frac{i^n}{2\pi}\,\mathbb{H}_n\{F_n(\rho)\}. \tag{6.11}$$

Therefore, according to the previous equations, the operation of taking the 2D polar Fourier transform of a function $f(r,\theta)$ can be broken down into 1) first finding its Fourier series expansion in the angular variable $f_n(r)$, 2) finding the *n*th order Hankel transform (of the spatial radial variable to the spatial frequency radial variable) of the *n*th coefficient in the Fourier series and appropriately scaling the result $F_n(\rho) = 2\pi\, i^{-n}\mathbb{H}_n\{f_n(r)\}$ and 3) then finding the inverse Fourier series expansion $\sum_{n=-\infty}^{\infty} F_n(\rho)\, e^{in\psi}$.

As long as this procedure calls on the Hankel transform and Fourier series, the concept of utilizing known data before attempting to evaluate transforms is employed by association. The *lookup* tables obtained for the Hankel transform and the Fourier series are thus used.

6.4.1 Direct Two dimensional Fourier transform in polar coordinates

The polar coordinates form of the 2D Fourier transform is written as

$$F(\rho,\psi) = \int_0^{\infty}\int_{-\pi}^{\pi} f(r,\theta)\, e^{-ir\rho\cos(\psi-\theta)} r\,dr\,d\theta. \tag{6.12}$$

This is done by introducing the expressions $x = r\cos\theta,\quad y = r\sin\theta$ and similarly in the spatial frequency domain, $\omega_x = \rho\cos\psi\quad \omega_y = \rho\sin\psi$ or conversely, $r^2 = x^2 + y^2$, $\theta = \arctan(y/x)$ and $\rho^2 = \omega_x^2 + \omega_y^2$, $\psi = \arctan\left(\omega_y/\omega_x\right)$.

As discussed in Chapter 3, section (3.1.3), in terms of polar coordinates, the Fourier transform operation transforms the spatial position radius and angle (r,θ) to the frequency radius and angle (ρ,ψ).

6.4.1.1 Calling sequence

The input arguments for the Direct 2D polar Fourier transform procedure include an expression, equation or group of these, radial and angular variables on which these depend, and variables in the frequency domain on which the transform depends. A check is made, as in all other transforms in this toolbox to ensure that the output variables are independent of the input variables.

6.4.1.2 Implementation

The Direct 2D polar Fourier transform of a group of expressions or equation is obtained by applying the transform to each expression within the group or equation individually using the "*map*" command.

The appropriate convolution rule is applied here if the check for convolution is successful after which the double integral for the Direct 2D Fourier transform is evaluated. It is important to note that the order of integration matters here since it determines how the result is presented to the user. It is therefore advisable to do all infinite integrals last and not before performing final integrations.

The Direct 2D Fourier transform procedure returns the call as entered in cases where it is unsuccessful in its attempt at evaluating. Like many of the other transforms in this subpackage, the direct 2D polar Fourier transform procedure is stored in the IntegralTrans package in the SCAToolbox. Figure 6.8 gives the flowchart for the direct 2D Fourier transform in polar coordinates.

Comparison between the Direct and Indirect 2D polar Fourier transform

The Direct 2D Fourier transform as defined in equation (6.12) does not always evaluate/simplify to the desired result or to the result in a useful form. The direct method sometimes returns indeterminate results as well. It has been found that evaluation of the 2D Fourier transform via Fourier series and Hankel transforms as shown in Subsection (6.4.2) results in a higher likelihood of successful evaluations.

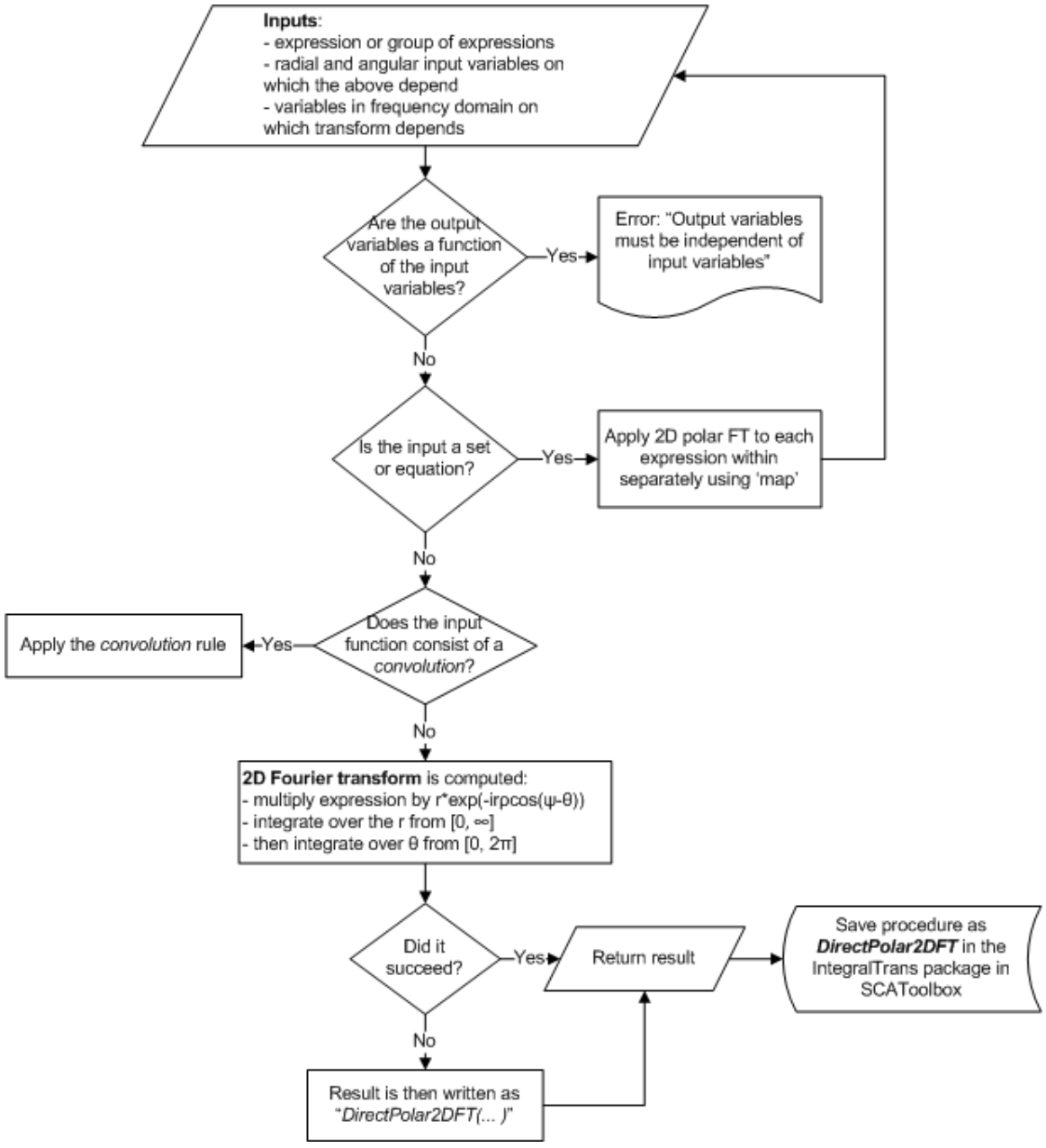


Figure 6.8: Flowchart of the Direct 2D polar Fourier transform

6.4.2 2D polar Fourier transform via Fourier series and Hankel transforms

Starting with the function $f(r,\theta)$ as an input, we evaluate its 2D Fourier transform in polar coordinates as an inverse Fourier series:

$$F(\rho,\psi) = \sum_{n=-\infty}^{\infty} F_n(\rho) e^{in\psi} \tag{6.13}$$

where the Fourier coefficient of the transform is the scaled-Hankel transform of the Fourier series coefficient of the original function:

$$F_n(\rho) = 2\pi\, i^{-n} \int_0^{\infty} f_n(r) J_n(\rho r)\, r dr = 2\pi\, i^{-n} \mathbb{H}_n\{f_n(r)\} \tag{6.14}$$

and the Fourier series coefficients of the original function are obtained as

$$f_n(r) = \frac{1}{2\pi} \int_0^{2\pi} f(r,\theta)\, e^{-jn\theta} d\theta\,. \tag{6.15}$$

The details of the derivation are discussed in Chapter 3, section (3.1.3) and show that the 2D Fourier transform in polar coordinates is equivalent to performing these three steps i.e. finding the complex Fourier series coefficients, then computing the Hankel transform of f_n to obtain the Fourier coefficient of the transform, and finally taking the inverse Fourier series of F_n.

6.4.2.1 Calling sequence

The 2D polar Fourier transform procedure accepts as its inputs an expression, equation or group of these, radial and angular variables on which these depend, and variables in the frequency domain on which the transform depends. Similarly to the other transformations in this body of work, a check is made to ensure that the output variables are independent of the input variables.

6.4.2.2 Implementation

As done with all the other transformations, in each expression within the group or equation via "*map*". The issue of identifying convolution arises again here and a similar approach of checking for it is used. The convolution rule that is appropriate here is applied if the check proves successful. This is discussed further in the Rules section of Chapter 3 and section (7.6) of Chapter 7.

After the rules have been applied where necessary, the 2D Fourier transform procedure first calls the 1D Fourier series, with respect to the angular variable to obtain

the Fourier coefficient, f_n and then evaluates the *n*th order Hankel transform of f_n, $\mathbb{H}_n\{f_n\}$. This result is then scaled by $2\cdot\pi\cdot I^{-n}$ to give F_n and finally the inverse Fourier series taken to obtain the 2D Fourier transform, $F(\rho,\psi)$.

When the 2D Fourier transform procedure unsuccessful in obtaining a result, the procedure returns the call as entered. The 2D polar Fourier transform procedure is stored in the IntegralTrans package in the SCAToolbox. Figure 6.9 gives the flowchart for the 2D Fourier transform in polar coordinates.

Figure 6.9 : Flowchart of the forward 2D Fourier transform in polar coordinates

6.4.3 Inverse Two dimensional Fourier transform

The corresponding inverse 2D Fourier transform in polar coordinates is written as

$$f(\vec{r}) = \frac{1}{(2\pi)^2} \int_0^{\infty} \int_0^{2\pi} F(\vec{\omega}) e^{j\vec{\omega}\vec{r}} \, d\psi \, \rho d\rho \tag{6.16}$$

where the shorthand notation of $\vec{\omega} = (\rho, \psi)$, $\vec{r} = (r, \theta)$ are used.

In terms of polar coordinates, the inverse Fourier transform operation transforms the frequency radius and angle (ρ, ψ) to the spatial position radius and angle (r, θ).

6.4.3.1 Calling sequence

The inverse 2D polar Fourier transform procedure accepts as its inputs an expression, equation or group of these, variables in the frequency domain on which these depend, and radial and angular variables on which the transform depends. A check is made to ensure that the output variables are independent of the input variables.

6.4.3.2 Implementation

Similar to the other transformations, in order to obtain the inverse 2D polar Fourier transform of a group of expressions or equation, the procedure is applied individually to each expression within the group or equation via "*map*". The inverse convolution rule is not applied here since it does not appear to be as useful and cost effective as the convolution-multiplication rule.

The inverse 2D polar Fourier transform procedure is broken down into steps like in its forward counterpart's case. Here, it calls the 1D Fourier series first, with respect to the angular variable in the Fourier domain to obtain the Fourier coefficients of the Fourier domain expansion, F_n and then obtains the *n*th order Hankel transform of F_n, $H_n(F_n)$. The result is then scaled by $\frac{i^n}{2\pi}$ to give f_n. The inverse 2D polar Fourier transform, $f(r, \theta)$ is finally obtained by evaluating the inverse Fourier series.

An unsuccessful run of this procedure results in the procedure returning the call as entered. The inverse 2D polar Fourier transform procedure is stored in the IntegralTrans package in the SCAToolbox. Figure 6.10 gives the flowchart for the inverse 2D Fourier transform in polar coordinates.

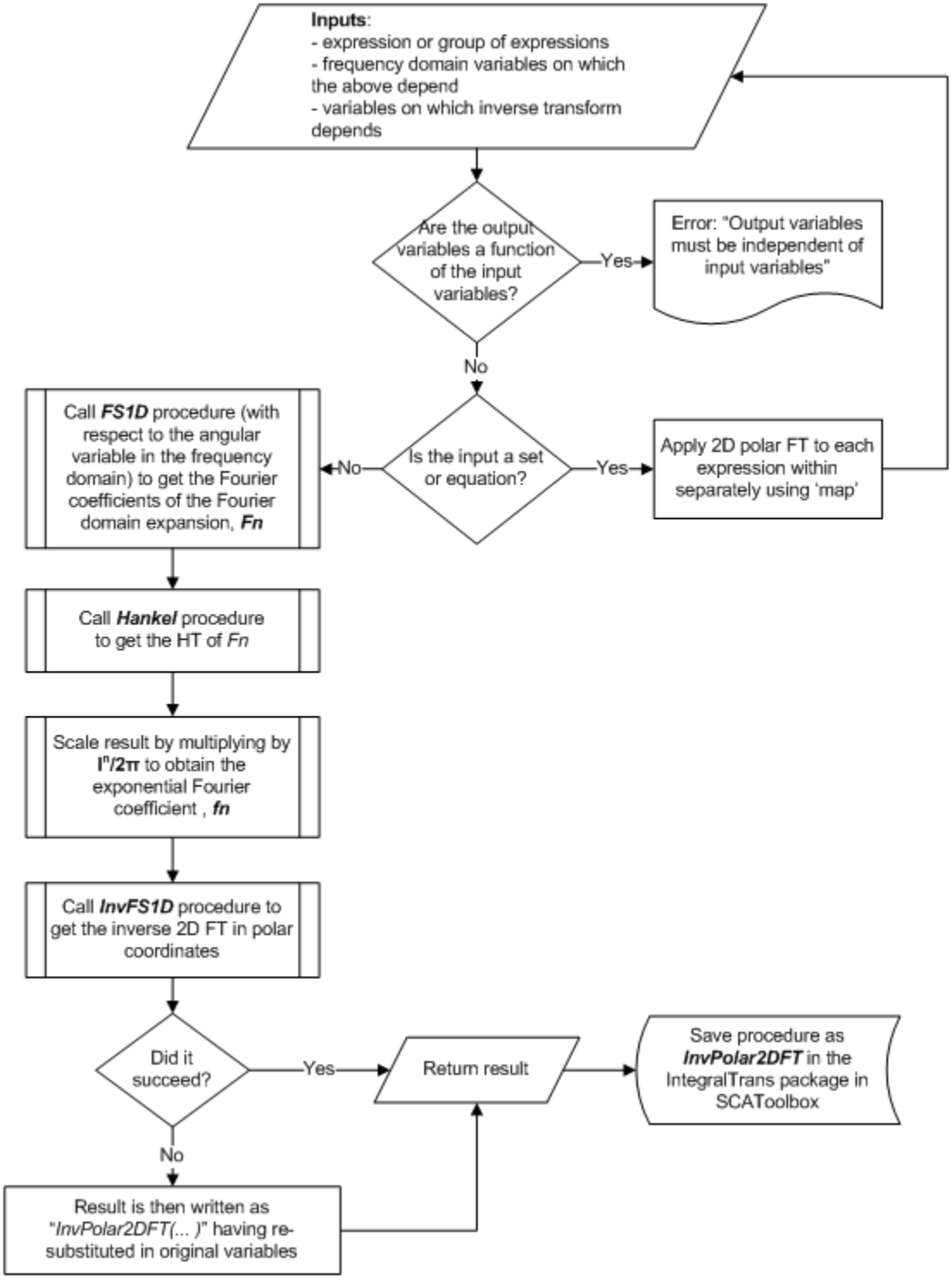


Figure 6.10: Flowchart of the inverse 2D Fourier transform in polar coordinates

6.5 Testing and Verification

In order for the *SCAToolbox* package to be tested, the path to which it is saved must be known, as discussed in section (4.3) of Chapter 4. The toolbox can be accessible from any file as long as the location of the package is known.

In Maple, the path of the package needs to be initialized to allow for the path search to be activated before it can be uploaded for use. The command line: "*libname := libname, "C:\\toolbox_path"*"" helps achieve this. The Maple command that provides access to a saved package is "*with*". When "*with(package_name)*" is called, the short form names of the commands integral to the Maple package are available for use. When using packages in Maple, there are two calling sequences that can be used to access commands within it. There is the long form (which can *always* be used to access commands in a package) and the short form (which can be used to access commands during the current Maple session). For example, the long form of calling the "*angular convolution*" command in the *SCAToolbox* package is done as follows: *SCAToolbox[AngConv](arguments)*. The short form of calling the same command after calling *with(SCAToolbox)* first is just *AngConv(arguments)*. As stated before, the "*unwith*" command is used if one wants to leave the package.

A summary of the steps (shown in Appendix B) to follow in order to be able to test a package is:

1. Open new file
2. Identify filename and desired path of toolbox
3. Initialize and upload package
4. Obtain access to short form for calling commands within package, if applicable
5. Test package with commands and inputs for which results are known or easily attainable
6. Save file if needed for future use.

The following sections give examples of how each procedure within the *SCAToolbox* package is tested.

6.5.1 Testing Supporting functions within *SCAToolbox*

In this section the procedures written to implement the supporting functions for the *SCAToolbox* package and its subpackages shown in Chapter 5 are tested by using some simple functions for which results are known. The functions are given to the procedures as inputs and checked against hand calculated results to see if they match.

6.5.1.1 Examples of convolutions

Convolutions are tested here. Each convolution type is tested with appropriate simple functions. Since the "Bracket" convolution covers all the types of convolution, it is tested by verifying that the results obtained match that of the correct type of convolution with the same function inputs.

Example 1:

First to be tested is the 1D convolution in Cartesian coordinates.

Theory: A *Dirac* function convolved with any other function returns the function i.e. $f(x) * \delta(x) = f(x)$. This is shown to be so as well from the procedure *OneDCartConv*.

> $OneDCartConv\left(s^2, \text{Dirac}(s), s\right)$

$$s^2$$

The equivalent statement using the "Bracket" convolution is

> $Conv\left(s^2, \text{Dirac}(s), [s], 1\ dcartesian\right)$

$$s^2$$

It also follows that a shifted *Dirac* function convolved with any other function gives back the function shifted by the same value i.e. $f(x) * \delta(x - x_0) = f(x - x_0)$. The *OneDCartConv* procedure helps illustrate this.

> $OneDCartConv\left(\frac{1}{x}, \text{Dirac}(x - 5), x\right)$

$$\frac{1}{x - 5}$$

> $Conv\left(\text{Dirac}(x - 5), \frac{1}{x}, [x], 1\ dcartesian\right)$

$$\frac{1}{x-5}$$

The above computation shows that the "Bracket" convolution works and that the operation does indeed commute.

Example 2:

Two *Dirac* functions are convolved together, each a function of one of the variables in 2D. This allows for testing the *TwoDCartConv* procedure.

> *TwoDCartConv*(Dirac(x), Dirac(y), x, y)

$$1$$

> *Conv*(Dirac(y), Dirac(x), [x, y], 2 *dcartesian*)

$$1$$

Theory: A shifted 2D *Dirac* function convolved with any other function gives back the function shifted by the same values i.e. $f(x,y) * \delta(x-1, y-2) = f(x-1, y-2)$. This is shown to be so as well from the procedure *TwoDCartConv*.

> *TwoDCartConv*($x \cdot y$, Dirac($x-1$) $\cdot$ Dirac($y-2$), x, y)

$$(x-1)\ (y-2)$$

Example 3:

The *TwoDPolarConv* procedure is tested here. The impulse function in polar coordinates is convolved with unity. Caution: a proper 2D convolution must be defined in order for the known properties to apply.

The polar coordinate system is obtained by

> *addcoords* ($z\_cylindrical$, [z, r, θ], [$r \cdot \cos(\theta)$, $r \cdot \sin(\theta)$, z])

The Delta Dirac function cannot be graphed in Maple and is therefore approximated for plotting purposes only as

> *DiracApprox* $(x, \varepsilon) := \frac{1}{\pi} \cdot \frac{\varepsilon}{\left(x^2 + \varepsilon^2\right)}$

$$DiracApprox := (x, \varepsilon) \rightarrow \frac{\varepsilon}{\pi \left(x^2 + \varepsilon^2\right)}$$

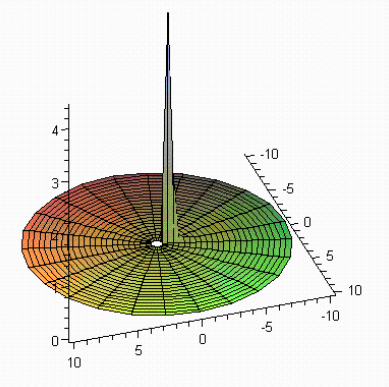
The convolution of $\frac{1}{r}\cdot\delta(r-1)\cdot\delta(\theta-\pi)$ and 1 gives

> $TwoDPolarConv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1)\cdot \mathrm{Dirac}(\theta - \mathrm{Pi}), 1, r, \theta\right)$

$$1$$

The function used in this example is shown below.

> $plot3d\left(\frac{1}{r}\cdot DiracApprox\,(r-1, 0.001)\cdot DiracApprox\,(\theta-\pi, 0.001),\right.$
$\left. r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, axes = framed\right)$



Example 4:

Trigonometric functions, specifically sine and cosine are used to test the *AngConv* procedure. The convolution is seen to be commutative.

> $AngConv\,(\sin(\theta), \cos(\theta), \theta)$

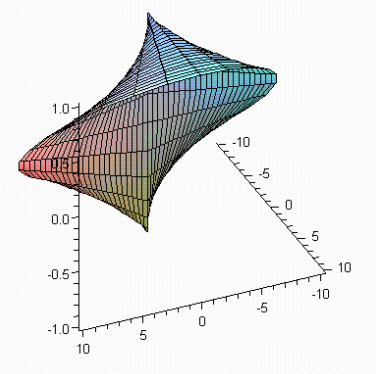
$$\frac{1}{2}\sin(\theta)$$

> $Conv\,(\sin(\theta), \cos(\theta), [\theta], angular\,)$

$$\frac{1}{2}\sin(\theta)$$

The angular convolution can further be tested by convolving the above mentioned delta function and $\sin(\theta)$ (shown below).

> $plot3d\,(\sin(\theta)\,, r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, axes = framed)$

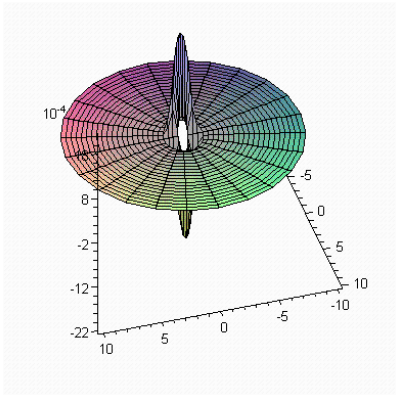


The result being:

> $Conv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1)\cdot \mathrm{Dirac}(\theta-\mathrm{Pi}), \sin(\theta), [\theta], angular\right)$

$$-\frac{1}{2}\,\frac{\mathrm{Dirac}(r-1)\,\sin(\theta)}{\pi\, r}$$

> $plot3d\left(-\frac{1}{2}\cdot\frac{\sin(\theta)\cdot DiracApprox\,(r-1, 0.001)}{r\cdot\pi}, r=0..10, \theta=0..2\cdot\pi, coords = z\_cylindrical, axes = framed\right)$



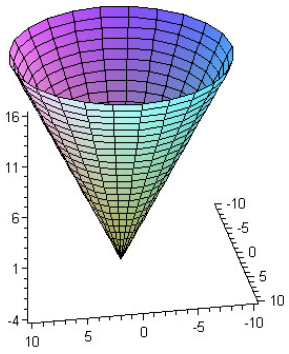
Example 5:

The *RadConv* procedure is tested by using two functions that are dependent on the radial variable.

> $RadConv\,(\mathrm{Dirac}(r-2), r, r)$

$$2\,r-4$$

> $plot3d\left(2\cdot r-4, r=0..10, \theta=0..2\cdot\pi, coords = z\_cylindrical, axes = framed\right)$



The “Bracket” convolution proves accurate again.

> $Conv(\mathrm{Dirac}(r-2), r, [r], radial)$

$$2r-4$$

Example 6:

Lastly, in order to test the *SerConv* procedure, two functions that are dependent on the series variable are used. The “Bracket” convolution approach gives identical results.

> $SerConv\left(2, \frac{1}{n}, n\right)$

$$\sum_{n=-\infty}^{\infty} \frac{2}{n}$$

> $Conv\left(2, \frac{1}{n}, [n], series\right)$

$$\sum_{n=-\infty}^{\infty} \frac{2}{n}$$

6.5.1.2 Examples for *takeAlook*

Here, the *takeAlook* procedure is tested by taking a look into the *Hankel lookup* tables. In the next section, these are verified with the *Hankel* procedure itself.

> $takeAlook\ (2, HankelTable\ )$

$$\frac{2\,\mathrm{Dirac}(\rho)}{\rho}$$

> $takeAlook\left(\frac{A}{\sqrt{r^2+2^2}}, HankelTable0\right)$

$$\frac{A\,e^{-\rho\sqrt{4}}}{\rho}$$

> takeAlook($e^{-\frac{1^2 \cdot r^2}{2}}$, HankelTable0)

$$e^{-\frac{1}{2}\rho^2}$$

6.5.2 Testing *IntegralTrans* subpackage of *SCAToolbox*

The procedures that make up the *IntegralTrans* subpackage are tested in this section. These procedures are those discussed and analyzed in sections 6.2, 6.3 and 6.4 of this chapter. As in the first part of this section, the procedures are tested by using some simple functions (entered as inputs) for which their appropriate results are known or can easily be evaluated by hand (against which procedures' results are checked).

6.5.2.1 Examples for Hankel transforms

A *list* of functions is entered for which the *Hankel transform* of order 0 of the contents is sought. The list obtained as a result has the correct order of the transforms within it. These are also used to verify results obtained from the previously discussed *takeAlook* procedure.

> $Hankel\left(\left[2, \frac{2}{r}, r^{-1.5}\right], r, s, 0\right)$

$$\left[\frac{2\,\mathrm{Dirac}(s)}{s}, \frac{2}{s}, \frac{2.092099241}{\sqrt{s}}\right]$$

> $Hankel\left(\left[\frac{1}{\sqrt{r^2+2^2}}, \frac{1}{r^2+1^2}\right], r, s, 0\right)$

$$\left[\frac{e^{-s\sqrt{4}}}{s}, \mathrm{BesselK}(0., s)\right]$$

> $Hankel\left(\left[e^{-\frac{1^2 \cdot r^2}{2}}, r^{-1} \cdot e^{-2 \cdot r}, r^{0-1} \cdot e^{-3 \cdot r}, e^{0-1 \cdot r^2} \cdot \mathrm{BesselJ}(0, 2 \cdot r)\right], r, s, 0\right)$

$$\left[e^{-\frac{1}{2}s^2}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{4+s^2}}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{9+s^2}}, \int_0^\infty e^{-r^2}\,\mathrm{BesselJ}(0, 2\,r)\,\mathrm{BesselJ}(0, s\,r)\,r\,dr\right]$$

6.5.2.2 Examples for inverse Hankel transforms

The functions entered for the *inverse Hankel transform* are also given in the form of a *list* for the same reason as in the forward transform case. It is evident from these results that the Hankel transform is indeed *self-reciprocating*.

> $InvHankel\left(\left[2, \frac{2}{s}, s^{-1.5}\right], s, r, 0\right)$

$$\left[\frac{2\,\mathrm{Dirac}(r)}{r}, \frac{2}{r}, \frac{2.092099241}{\sqrt{r}}\right]$$

> $InvHankel\left(\left[\frac{1}{\sqrt{s^2+2^2}}, \frac{1}{s^2+1^2}\right], s, r, 0\right)$

$$\left[\frac{e^{-r\sqrt{4}}}{r}, \mathrm{BesselK}(0., r)\right]$$

> $InvHankel\left(\left[e^{-\frac{1^2 \cdot s^2}{2}}, s^{-1}\cdot e^{-2\cdot s}, s^{0-1}\cdot e^{-3\cdot s}\right], s, r, 0\right)$

$$\left[e^{-\frac{1}{2}r^2}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{r^2+4}}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{9+r^2}}\right]$$

6.5.2.3 Examples for 1D Fourier series

The *FS1D* procedure evaluates the one dimensional Fourier series of a given function or group of functions. This procedure allows for four ways of calling the procedure and the examples below depict these cases. The first case is where the complex coefficients of the Fourier series of the function are sought. In the second case, the user requires the real coefficients of the Fourier series of the function whereas in the third case the user requires the series form of the Fourier series transform. The fourth and final case is when only 4 arguments are entered for which the procedure defaults to the complex coefficient case.

> $FS1D(1, x, n, 0..2\cdot\pi, coefficientComplex)$

$$\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases}$$

The real coefficients are returned as the coefficients of the cosine A_n and the coefficients of the sine functions, B_n in that order.

> $FS1D(x^2, x, 5, -\pi..\pi, coefficientReal)$

$$\frac{2}{3}\pi^2 - \frac{4}{25}, 0$$

> $FS1D(\sin(x), x, n, 0 .. \pi, series)$

$$\sum_{n=-\infty}^{\infty}\left(-\frac{2\,e^{I\,n\,x}}{\pi\,(4\,n^2-1)}\right)$$

> $FS1D\left(\frac{1}{2}\cdot \mathrm{Dirac}(r), x, n, 0..2\cdot\pi\right)$

$$\begin{cases} \frac{1}{2}\,\mathrm{Dirac}(r) & n=0 \\ 0 & otherwise \end{cases}$$

6.5.2.4 Examples for inverse 1D Fourier series

The examples given below for the *inverse Fourier series* are twofold. The first two are evaluated by direct computation within the procedure while the third is obtained from the inverse Fourier series *lookup* table.

> $InvFS1D(piecewise(n=0, 1), n, x)$

$$1$$

> $InvFS1D(piecewise(n=3, 2), n, x)$

$$2\,e^{3\,I\,x}$$

> $InvFS1D\left((-1)^n\cdot\frac{1}{2\cdot n}, n, x\right)$

$$\begin{cases} -\ln\left(2\cos\left(\frac{1}{2}\,x\right)\right) & -\pi < x \ \textbf{and}\ x < \pi \\ 0 & otherwise \end{cases}$$

6.5.2.5 Examples for the direct 2D Fourier transform in polar coordinates

The *direct 2D polar Fourier transform* is tested by the use of some well known functions as in the other transform cases. The *direct 2D polar Fourier transform* means trying to evaluate the 2D polar Fourier transform from the integral definition, as opposed to using the decomposition of Fourier series -> *n*th order Hankel transform -> inverse Fourier series, as shown in Chapter 3. The results of calculating the direct form of the 2D polar Fourier transform can be checked against those in [5]. The direct form of the 2D polar Fourier transform for the unity function and some Dirac delta functions are given here for comparison. The procedure was unable to evaluate the double integral of 1 to a closed form output for which the result is clearly known from literature or the alternative route for evaluating this transform as previously seen. The procedure therefore returns the integral without evaluating it.

> $DirectPolar2DFT(1, r, \theta, \rho, \psi)$

$$\int_0^{\infty} \int_0^{2\pi} r\, e^{-\rho\, r \cos(\psi - \theta)}\, d\theta\, dr$$

The direct 2D polar Fourier transform for the two functions below correlate well with that of the "Indirect" method as well as with the expected results in literature.

> $DirectPolar2DFT\left(\frac{1}{r} \cdot \text{Dirac}(r), r, \theta, \rho, \psi\right)$

$$2\pi$$

> $DirectPolar2DFT\left(\frac{1}{r} \cdot \text{Dirac}(r) \cdot \text{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$1$$

The direct 2D polar Fourier transform procedure is unable to evaluate the shifted Dirac function in the next example though.

> $DirectPolar2DFT\left(\frac{1}{2 \cdot \pi \cdot r} \cdot \text{Dirac}(r - 3), r, \theta, \rho, \psi\right)$

$$\frac{1}{2} \frac{\int_0^{2\pi} e^{-3\rho\,(\cos(\psi)\cos(\theta) + \sin(\psi)\sin(\theta))}\, d\theta}{\pi}$$

These few examples help solidify our findings that there is a better chance of successfully evaluating the 2D Fourier transform using Fourier series and Hankel transforms (applying their tables and later, rules) than the direct method as seen in Subsection 6.4.1.2.

6.5.2.6 Examples for 2D polar Fourier transform via Fourier series and Hankel transforms

The same idea of using known functions and corresponding/easily evaluated transforms is adapted to test the *2D Fourier transform in polar coordinates*. For proofs of these results please refer to [5].

> $Polar2DFT(1, r, \theta, \rho, \psi)$

$$\frac{2\pi\, \text{Dirac}(\rho)}{\rho}$$

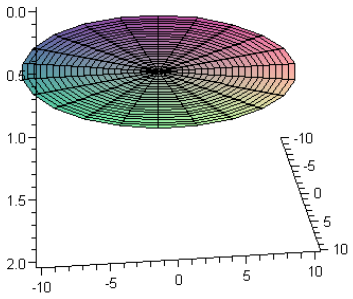


Figure 6.11: Plot of 1 in polar coordinates

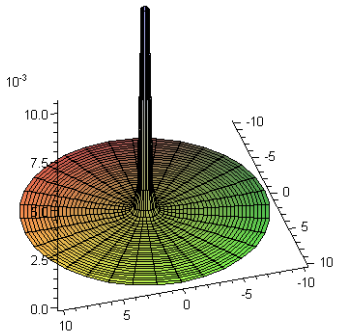


Figure 6.12: Plot of 2D polar FT of 1

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r), r, \theta, \rho, \psi\right)$

$$2\,\pi$$

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$1$$

> $Polar2DFT\left(\frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-3), r, \theta, \rho, \psi\right)$

$$\mathrm{BesselJ}(0, 3\,\rho)$$

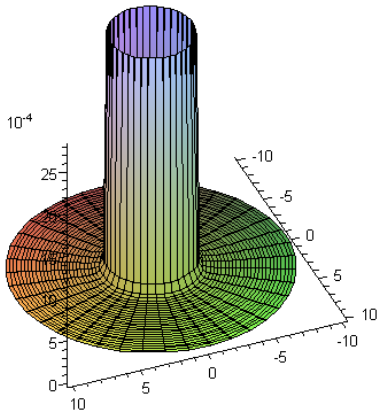


Figure 6.13: Plot of the Dirac(r-3)/2πr function

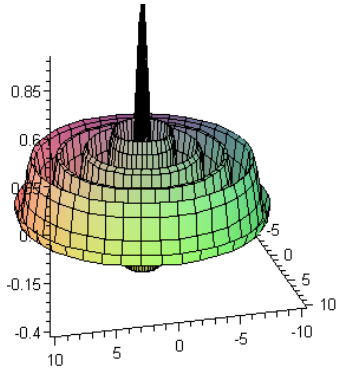


Figure 6.14: Plot of $J_0(3\rho)$

The 2D Fourier transform of 2D polar convolutions are also tested here. The two convolutions transformed include the 2D convolution of two radially symmetric functions and the 2D convolution of a radially symmetric and non-radially symmetric function. The details of the proofs are provided in section (3.2.5) of Chapter 3.

***Theory**:* The Fourier transform of a 2D convolution of two radially symmetric functions is the product of their respective Fourier transforms (solely dependent on the radial variable in the frequency domain). The two functions used in this example are $\frac{1}{r}\cdot\delta(r-1)$ and r and their transforms are shown below.

```
> Polar2DFT(1/r·Dirac(r − 1), r, θ, ρ, ψ)
```

$$2\,\pi\,\mathrm{BesselJ}(0,\rho)$$

```
> Polar2DFT(r, r, θ, ρ, ψ)
```

$$2\,\pi\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0,\rho\,r)\,dr\right)$$

The Fourier transform of the convolution of these functions gives

```
> Polar2DFT('Conv'(1/r·Dirac(r − 1), r, [r, θ], 2 dpolar), r, θ, ρ, ψ)
```

$$4\,\pi^2\,\mathrm{BesselJ}(0,\rho)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0,\rho\,r)\,dr\right)$$

Note that in order to simplify results, they are converted to piecewise functions i.e. convert expression(s) containing the Heaviside, absolute value command, etc. to the piecewise form.

***Theory**:* Convolving a radially symmetric function with a non-radially symmetric one in 2D and evaluating the Fourier transform of the result gives the well-known convolution-multiplication rule as well. The two functions used here are $\frac{1}{r}\cdot\delta(r-1)\cdot\delta(y-\pi)$ (transform below) and r (transform given above).

```
> Polar2DFT((1/r)·Dirac(r − 1)·Dirac(y − Pi), r, y, ρ, ψ)
```

$$\sum_{n=-\infty}^{\infty} e^{-\frac{3}{2} I n \pi} \, \mathrm{BesselJ}(n, \rho) \, e^{I \psi n}$$

Fourier transforming the convolution of these functions results in

```
> Polar2DFT('Conv'((1/r)·Dirac(r − 1)·Dirac(y − Pi), r, [r, y],
  '2dpolar'), r, y, ρ, ψ)
```

$$2 \left(\sum_{n=-\infty}^{\infty} e^{-\frac{3}{2} I n \pi} \, \mathrm{BesselJ}(n, \rho) \, e^{I \psi n} \right) \pi \left(\int_0^{\infty} r^2 \, \mathrm{BesselJ}(0, \rho \, r) \, dr \right)$$

6.5.2.7 Examples for inverse 2D Fourier transform in polar coordinates

Examples of the *inverse 2D Fourier transform in polar coordinates* are shown below. The *InvPolar2DFT* procedure is used to implement this inverse transform. The input functions can also be entered as lists or equations.

```
> InvPolar2DFT(1, ρ, ψ, r, θ)
```

$$\frac{1}{2} \, \frac{\mathrm{Dirac}(r)}{r \, \pi}$$

Chapter 7 : Rules

The syntax and implementation of the rules that make up the Fourier operational toolset of Dirac-delta, exponential, spatial shift, multiplication and convolution in Maple are discussed in this chapter. In addition to the idea of *lookup tables*, the concept of applying rules is employed. This concept allows for the identification of certain situations and activates the appropriate rules for them.

Each section in this chapter is dedicated to a particular rule. The first two sections discuss the rules associated with the *Dirac-delta* and *complex exponential* functions respectively. The *scaling* rule and its Maple implementation are discussed in the third section while the fourth section discusses the *shift* rule and the Maple procedures required to implement it. The *modulation* rule, which only applies to the 1D Fourier series, is discussed in the fifth section. The *convolution-multiplication* rule is the last to be tackled in this chapter. The subsections for each section follow the same structure of theory, calling sequence and method of execution and finally testing, so that each subsection may be referred to independently where necessary, allowing the reader to then jump to whichever subsection is of interest.

A summary of the rules is given in Table 2 at the end of the chapter.

7.1 Dirac-delta functions

For the general two dimensional polar impulse function, the 2D Fourier transform in polar coordinates is implemented using the idea of *lookup tables*. That Dirac-delta function and its corresponding rule is given as an entry to the "*FT2Dtable*" in the *IntegralTrans* package.

A radially symmetric delta function like the Ring-Delta function, discussed in section (3.2.1.2) of Chapter 3, however, is not implemented the same way but allowed to go through the steps for the direct evaluation of the 2D polar Fourier transform (see section (3.1.3) of Chapter 3 for details) in order to analyze the results vis-à-vis the *direct* 2D Fourier transform.

7.1.1 Theory

The Fourier pair that illustrates the general 2D polar impulse function and its transform is

$$f(\vec{r}) = \delta(\vec{r} - \vec{r}_0) = \frac{1}{r}\delta(r - r_0)\delta(\theta - \theta_0) \Leftrightarrow F(\vec{\omega}) = \sum_{n=-\infty}^{\infty} i^{-n} J_n(\rho r_0) e^{-in\theta_0}\, e^{in\psi} = e^{-i\vec{\omega}\cdot\vec{r}_0} \qquad (7.1)$$

where the 'spike' is at $\vec{r}_0$.

For the Ring-Delta function (only a function of *r*), the 2D Fourier transform in polar coordinates is given by

$$F(\rho) = 2\pi \int_0^{\infty} \left\{ \frac{1}{2\pi r}\delta(r - r_0) \right\} J_0(\rho r) r dr = J_0(\rho r_0)\,. \qquad (7.2)$$

7.1.2 Method of execution

The general 2D impulse function in polar coordinates and its transform are entered in the "*FT2Dtable*" as previously discussed. The *Polar2DFT* procedure, takes a look into the *FT2Dtable* to see if the expression to be transformed matches any of the entries in the first column of the table. For a properly written general impulse function as seen in equation (7.1), a match will be found and the associated specified transform in the second

column of the table is returned. This helps bypass the more complicated execution of the transform.

However, the Ring-Delta function would have to go through the steps in the procedure since it will not be able to obtain a result from the table since a rule is not entered for that case in the table for the 2D polar Fourier transform. There is, however, a rule for one dimensional Dirac-delta functions in the table for the Fourier series transform, *FS1Dtable*. For the Dirac-delta function centered at x_0 then, the rule that applies for the Fourier series coefficient is

$$f(x) = A\delta(x - x_0) \Leftrightarrow C_n = \frac{A}{2\pi} e^{-i \cdot n \cdot x_0} \tag{7.3}$$

where A is a constant.

For the Hankel transform of the Dirac-delta function, it is important to ensure that its definition and properties in the CAS software being used match those of the users. In Maple, for instance, the value of the integral of the Dirac-delta and any function is halved at the end limits; i.e.

$$\int_a^b Dirac(x - x_0) f(x)\, dx = \begin{cases} 0 & x_0 < a, x > b \\ f(x_0) & a < x_0 < b \\ f(x_0)/2 & x_0 = a, x_0 = b \end{cases} \tag{7.4}$$

where $a < b$ when in fact, we want $f(x_0)$ at the end limits as well.

7.1.3 Implementation

The rule for the Dirac-delta functions is tested and verified here. The implementation of this rule is tested for the general case where the impulse is at $\vec{r}_0$, as well the case when the Dirac-delta function is at the origin. The Ring-delta function is also tested.

7.1.3.1 Examples of the general Dirac-delta function in polar coordinates

The Dirac-delta function centred at $r = 1$ and $\theta = \pi$ is used in this example.

```
> Polar2DFT((1/r)·Dirac(r − 1)·Dirac(y − π), r, y, ρ, ψ)
```

$$\sum_{n=-\infty}^{\infty} I^{-n}\, \mathrm{BesselJ}(n, \rho)\, e^{-I n \pi}\, e^{I \psi n}$$

Also tested here is the Dirac-Delta function at the origin. Both of these results correlate with expected results.

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$1$$

The plots for the impulse response at the origin and its Fourier transform are shown below. As already mentioned in section (6.5.1.1) of Chapter 6, the polar coordinate system is obtained by

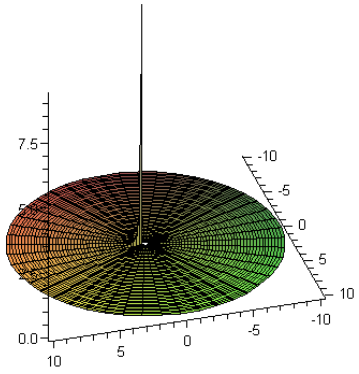
> $addcoords\left(z\_cylindrical, [z, r, \theta], [r\cdot\cos(\theta), r\cdot\sin(\theta), z]\right)$

and the Dirac-delta function approximated for plotting purposes only as

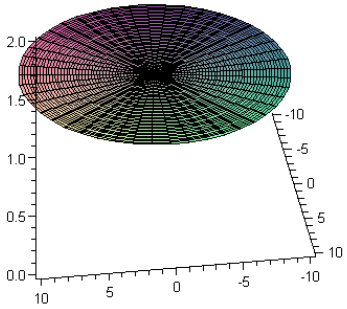
> $DiracApprox\left(x, \varepsilon\right) := \frac{1}{\pi}\cdot\frac{\varepsilon}{\left(x^2+\varepsilon^2\right)}$

$$DiracApprox := (x, \varepsilon) \rightarrow \frac{\varepsilon}{\pi\left(x^2+\varepsilon^2\right)}$$

> $plot3d\left(\frac{1}{r}\cdot DiracApprox\left(r, 0.001\right)\cdot DiracApprox\left(\theta, 0.001\right), r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 2000, axes = framed\right)$



> $plot3d\left(1, \rho = 0..10, \psi = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 2000, axes = framed\right)$



7.1.3.2 Example of the Ring-delta function

Tested here is the Ring-delta function as shown in equation (7.2) with $r_0 = 3$. The transform does indeed match with the theory.

> $Polar2DFT\left(\frac{1}{2\cdot \mathrm{Pi}\cdot r}\cdot \mathrm{Dirac}(r-3), r, \theta, \rho, \psi\right)$

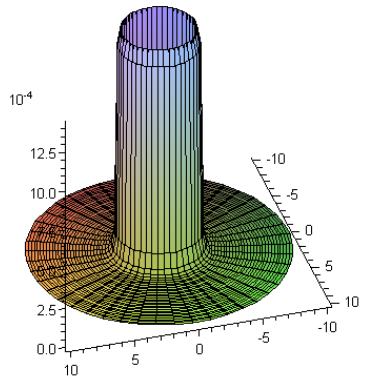
$$\sum_{n=-\infty}^{\infty} e^{-\frac{1}{2} I n \pi} \begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \mathrm{BesselJ}(n, 3\rho)\, e^{I \psi n}$$

> $convert\,(\ \mathbf{(4.7.8)}, 'piecewise', n)$

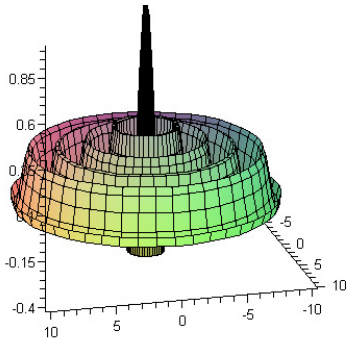
$$\mathrm{BesselJ}(0, 3\rho)$$

Plotted here are the Ring-delta function given above and its transform.

> $plot3d\left(\frac{1}{2\cdot\pi\cdot r}\cdot DiracApprox\,(r-3, 0.001), r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 2000, axes = framed\right)$



> $plot3d\left(\mathrm{BesselJ}(0, 3\cdot\rho), \rho = 0..10, \psi = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 2000, axes = framed\right)$



7.2 Complex exponential functions

The idea of *lookup tables* is again employed for the complex exponential functions. The Fourier transform rule for the complex exponentials is discussed in detail in section (3.2.2) of Chapter 3. The complex exponential function covers a wide range of expressions and is therefore useful for defining several functions. A special case of the complex exponential is 1, written as e^0 (see section (3.2.2.1) of Chapter 3).

7.2.1 Theory

The complex exponential and its transform are given in the Fourier pair,

$$\begin{aligned} f(\vec{r}) = e^{i\vec{\omega}_0\cdot\vec{r}} &= \sum_{n=-\infty}^{\infty} i^n J_n(\rho_0 r)\, e^{-in\psi_0}\, e^{in\theta} \Leftrightarrow \\ F(\vec{\omega}) = (2\pi)^2\, \delta(\vec{\omega}-\vec{\omega}_0) &= (2\pi)^2 \frac{1}{\rho}\delta(\rho-\rho_0)\,\delta(\psi-\psi_0) \end{aligned} \tag{7.5}$$

For the special case, $f(\vec{r}) = 1$, the 2D polar Fourier transform becomes

$$F(\vec{\omega}) = 2\pi \frac{1}{\rho}\delta(\rho). \tag{7.6}$$

7.2.2 Method of execution

The complex exponential function $e^{i\vec{\omega}_0\cdot\vec{r}}$ is entered in the first column of the *FT2Dtable* in the *IntegralTrans* package, while its corresponding transform is entered in the second column of the table. Similar to the case of the Dirac-delta function, the *Polar2DFT* procedure checks the expression against those in the first column of the *FT2Dtable* and if found returns its associated transform from the second column. Without this rule, the procedure tries to evaluate the transform directly via finding the Fourier series and then the Hankel transform which is very complex and thus time-consuming.

There exists a rule for a simple one dimensional complex exponential for the Fourier series transform given as

$$f(x) = Ae^{im\cdot x} \Leftrightarrow C_n = \delta_{n,m} = \begin{cases} A & n = m \\ 0 & \text{otherwise} \end{cases} \tag{7.7}$$

where A is a constant and $\delta_{n,m}$ is known as the *Kronecker delta* function.

7.2.3 Implementation

The three expressions tested and validated here are the complex exponential, its special case of unity and a simple 1D exponential for the Fourier series transform. The complex exponential is tested first with $f(\vec{r}) = e^{i\vec{\omega}_0 \cdot \vec{r}} = \sum_{n=-\infty}^{\infty} i^n J_n(2r)\, e^{-i\pi n}\, e^{in\theta}$ with $\vec{\omega}_0 = \{\rho_0 = 2, \psi_0 = \pi\}$ and then the special case of $f(\vec{r}) = 1$ is considered. The 1D exponential is tested last.

7.2.3.1 Example of the Complex exponential function

The Fourier transform of $f(\vec{r}) = e^{i\vec{\omega}_0 \cdot \vec{r}} = \sum_{n=-\infty}^{\infty} i^n J_n(2r)\, e^{-i\pi n}\, e^{in\theta}$ is

```
> Polar2DFT(sum(I^n·BesselJ(n, 2·r)·exp(-I·n·Pi)·exp(I·n·θ), n = -∞ .. ∞), r, θ, ρ, ψ)
```

$$\frac{4\,\pi^2\,\mathrm{Dirac}(\rho - 2)\,\mathrm{Dirac}(\psi - \pi)}{\rho}$$

The result here appears to match the literature, therefore validating it.

7.2.3.2 Example of the special case of the complex exponential function

The 2D Fourier transform of 1 is

```
> Polar2DFT(1, r, θ, ρ, ψ)
```

$$\sum_{n=-\infty}^{\infty} \frac{2\,\pi\, e^{-\frac{1}{2} I n \pi} \left\{ \begin{array}{ll} 1 & n = 0 \\ 0 & otherwise \end{array} \right. \mathrm{Dirac}(\rho)\, e^{I \psi n}}{\rho}$$

```
> convert((4.7.1), 'piecewise', n)
```

$$\frac{2\,\pi\,\mathrm{Dirac}(\rho)}{\rho}$$

which correlates with expected results are discussed before.

7.2.3.3 Example of the 1D exponential function for the Fourier Series transform

The 1D Fourier series coefficient of $f(\theta) = 2e^{i3\theta}$ is

```
> FS1D(2·exp(I·3·θ), θ, n, 0..2·π)
```

$$\begin{cases} 2 & n = 3 \\ 0 & otherwise \end{cases}$$

which matches the *Kronecker delta* function obtained in equation (7.7) above.

7.3 Scaling

Maple possesses an internal scaling rule which it applies to its functions. As a result, it tends to evaluate with this rule. The scaling rule requires parsing the entered expression to obtain the scaling factor and the expression devoid of the scaling factor. Getting the scale factor of the variable is quite straightforward and can be done in Maple by using the command "*coeff*". Obtaining the original expression devoid of the scaling factor, however, is rather complicated - requiring going through of the expression to alter the coefficients appropriately. It is therefore simpler to evaluate the scaled function by direct computation to obtain the desired form of the coefficient or series. The alternative is to side-step the internal rule by forcing our own. This is done by specifying a '*Scale*' operator which when identified, allows for executing the rule as desired. This is what is used in this toolbox.

7.3.1 Theory

A scaled function $f\left(\frac{\theta}{a}\right)$ will have the Fourier series,

$$f\left(\frac{\theta}{a}\right) = \sum_n f_n e^{\frac{in\theta}{a}} = \sum_m f_{ma} e^{im\theta} \tag{7.8}$$

where a is an integer except 0 and $m = \frac{n}{a}$. It is seen here that the period of the series is altered from T to $\frac{T}{a}$.

7.3.2 Method of execution

The scaling rule is implemented as a special operator in the Fourier series procedure. The user must specifically enter the scaled function in the form of the '*Scale*' operator in order for the scaling rule to be activated. The acceptable form of the entered expression to activate this rule is '*Scale*'*(original_function, dependent_variable ,scale_factor)*. For instance if the desired scaled function is $\frac{x^2}{4}$, then the expression to enter is

'*Scale*'(x^2 , x, $\frac{1}{2}$) over a desired range, say $x = -L..L$. The Fourier coefficient is then evaluated by reapplying the Fourier series to the original function over a scaled period i.e. "*FS1D*(x^2 , x, n, $\frac{-L}{2}..\frac{L}{2}$)".

The Fourier coefficients of the scaled function can also be obtained without specifying the '*Scale*' operator. This, in fact is how the scaling rule is verified.

7.3.3 Implementation

A few functions are tested in this section. The Fourier series transform/coefficient of the original function is obtained and compared with that of the scaled version of the function (both the direct form and the '*Scale*' operator form).

7.3.3.1 Examples of the Scaling rule

The first function to be tested is a simple parabolic function. The scaled versions are shown right after the evaluation of the Fourier coefficients of the original function. The scaling rule can be seen to produce correct results since the results obtained are the same as those in the direct scaling operation.

> $FS1D\left(x^2, x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{4}{3} & n = 0 \\ -\frac{1}{\pi^3 n^3}\left(2\left(-2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} - 2\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi\,n + \mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi^2 n^2 + 2\,\mathrm{I} - 2\,n\,\pi - \mathrm{I}\,\pi^2 n^2\right) e^{-\mathrm{I}\,\pi\,n}\right) & otherwise \end{cases}$$

> $FS1D\left(\frac{x^2}{4}, x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{1}{3} & n = 0 \\ -\frac{1}{2}\,\frac{1}{\pi^3 n^3}\left(\left(-2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} - 2\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi\,n + \mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi^2 n^2 + 2\,\mathrm{I} - 2\,n\,\pi - \mathrm{I}\,\pi^2 n^2\right) e^{-\mathrm{I}\,\pi\,n}\right) & otherwise \end{cases}$$

> $FS1D\left('Scale'\left(x^2, x, \frac{1}{2}\right), x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{1}{3} & n = 0 \\ -\frac{1}{2}\,\frac{1}{\pi^3\, n^3}\left(\left(-2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} - 2\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi\,n\right.\right. & \textit{otherwise} \\ \left.\left. +\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi^2\,n^2 + 2\,\mathrm{I} - 2\,n\,\pi - \mathrm{I}\,\pi^2\,n^2\right)\,e^{-\mathrm{I}\,\pi\,n}\right) & \end{cases}$$

The sine function is also tested here.

> $FS1D(\sin(x), x, n, 0 .. \pi, series)$

$$\sum_{n=-\infty}^{\infty}\left(-\frac{2\,e^{\mathrm{I}\,n\,x}}{\pi\,(4\,n^2-1)}\right)$$

> $FS1D\left(\sin\left(\frac{x}{2}\right), x, n, 0 .. \pi, series\right)$

$$\sum_{n=-\infty}^{\infty}\frac{2\,(-1+4\,\mathrm{I}\,n)\,e^{\mathrm{I}\,n\,x}}{\pi\,(16\,n^2-1)}$$

> $FS1D\left('Scale'\left(\sin(x), x, \frac{1}{2}\right), x, n, 0 .. \pi, series\right)$

$$\sum_{n=-\infty}^{\infty}\frac{2\,(-1+4\,\mathrm{I}\,n)\,e^{\mathrm{I}\,n\,x}}{\pi\,(16\,n^2-1)}$$

7.4 Shifting

The shift rule is applicable to two dimensional Fourier transforms in polar coordinates as well as to one dimensional Fourier series transforms. The former is discussed in great detail in section (3.2.3) of Chapter 3 and shows that the Fourier transform of a shifted function, $f(\vec{r}-\vec{r}_0)$ is obtained by multiplying the Fourier transform of the original unshifted function $F(\vec{\omega})$ by the complex exponential, $e^{-i\vec{\omega}\cdot\vec{r}_0}$.

In the case of the 1D Fourier series transform, the appropriate shift rule is the one dimensional version of the Fourier transform case where the Fourier coefficient of the shifted function becomes the complex exponential-weighted coefficient of the original unshifted function.

7.4.1 Theory

The Fourier pair that illustrates the 2D transform of the shifted function is given by

$$f(\vec{r}-\vec{r}_0) \Leftrightarrow e^{-i\vec{\omega}\vec{r}_0} F(\vec{\omega}) \tag{7.9}$$

where the shift consists of a translation by r_0 and then a rotation by θ_0. The shift rule that applies to the 1D Fourier series transform is obtained from the expression for Fourier series given as

$$f(\theta-\theta_0) = \sum_{n=-\infty}^{\infty} f_n e^{in(\theta-\theta_0)} = \sum_{n=-\infty}^{\infty} \left\{ e^{-in\theta_0} f_n \right\} e^{in\theta} . \tag{7.10}$$

The Fourier coefficients of the shifted function are therefore given as

$$f(\theta-\theta_0) \Leftrightarrow e^{-in\theta_0} f_n . \tag{7.11}$$

7.4.2 Method of execution

Since the 2D polar Fourier transform is obtained by the use of the 1D Fourier series, the implementation of the latter is tackled first. In order to apply the shift rule in the 1D Fourier series transform procedure, the entered expression must be parsed to obtain the value of the shift. As mentioned before, parsing is a rather cumbersome operation in Maple and is therefore not recommended.

The alternative approach here is to somehow indicate that a shift is desired. This is thus how the '*Shift*' operator came about. The user must specifically enter this command in order for the rule to be applied. The command line of the shift operation is written as '*Shift(expression, variable, shift_value)*'. For example, in order to shift $f(x) = 3x^2 + 1$ by 1 to obtain $f(x-1) = 3(x-1)^2 + 1$, we write: '*Shift(* $3x^2+1$, x, 1*)*'. This way when the '*Shift*' operator is identified (by matching a string) the rule can be applied to the expression. It is important to note that the shift rule has been developed, so far, in this toolbox for scalar multiples of the shifted function and would require development to include more complex forms of it.

The shift rule applied for the 2D Fourier transform in polar coordinates is implemented similarly to the 1D Fourier series case. The shift operator here though has five arguments; the first being the expression to be shifted, the second and third being the radial and angular variables respectively, and the last two being the values to respectively

shift the radial and angular variables. An example would be shifting the radial variable of the function $f(\vec{r}) = f(r,\theta) = \frac{1}{r}\delta(r)\delta(\theta)$ by 3 and the angular variable by π. The Maple command written to do so would be '*Shift2D*($\frac{1}{r}\delta(r)\delta(\theta)$, $r, \theta, 3, \pi$)'. Like the one dimensional case, the shift rule has been implemented for scalar multiples of the shifted function.

7.4.3 Implementation

Tested and verified here are some examples for the shift rule in both the 1D Fourier series transform and the 2D polar Fourier transform. In both cases the appropriate Dirac-Delta function is used; shifted directly and then shifted using the '*Shift*' operator. The results are the same verifying the accuracy of the shift rule in this toolbox.

7.4.3.1 Example of the Shift rule in 1D Fourier series transform

The Dirac-delta function centred at $x = 1$ i.e. shifted by 1 is tested in this example. The unshifted function is tested first to obtain the original Fourier coefficient. It is indeed clear that the Fourier coefficient of the shifted function is the complex exponential-weighted coefficient of the original function.

```
> FS1D(Dirac(x), x, n, -2..2, coefficientComplex)
```

$$\frac{1}{4}$$

```
> FS1D(Dirac(x - 1), x, n, -2..2, coefficientComplex)
```

$$\frac{1}{4}\, e^{-I n}$$

```
> FS1D('Shift'(Dirac(x), x, 1), x, n, -2..2, coefficientComplex)
```

$$\frac{1}{4}\, e^{-I n}$$

7.4.3.2 Example of the Shift rule in 2D polar Fourier transform

In this example the 2D Dirac-Delta function in polar coordinates centred at $r = 3$ and $\theta = \pi$ is tested and seen to correlate with expected results keeping in mind that the complex exponential $e^{-i\vec{\omega}\cdot\vec{r}_0}$ is given in series as

$$e^{-i\vec{\omega}\cdot\vec{r}_0} = e^{-i\rho r_0\cos(\psi-\theta_0)} = \sum_{n=-\infty}^{\infty} i^{-n} J_n(\rho r_0)\, e^{-in\theta_0} e^{in\psi}\,. \tag{7.12}$$

```
> Polar2DFT(2/r·Dirac(r)·Dirac(θ), r, θ, ρ, ψ)
                                  2
> Polar2DFT(2/r·Dirac(r − 3)·Dirac(x − π), r, x, ρ, ψ)
                    ∞          -3/2 I n π
                   Σ    2 e              BesselJ(n, 3 ρ) e^(I ψ n)
                  n = - ∞
> Polar2DFT(2·'Shift2D'(1/r·Dirac(r)·Dirac(θ), r, θ, 3, Pi), r, θ, ρ, ψ)
                                 3 I ρ cos(ψ)
                              2 e
```

7.5 Modulation

When a function is multiplied by the complex exponential of the dependent variable, a rule can be applied to obtain its Fourier series coefficients. The modulation rule is that which links the function, $e^{ia\theta} f(\theta)$ with its Fourier coefficients. In this thesis, the modulation rule is explicitly defined for the one dimensional Fourier series only and *not* implemented for the two dimensional polar Fourier transform since a rule has already been implemented for the complex exponentials.

7.5.1 Theory

Using the expression for Fourier series given in equation (3.4) of Chapter 3, the Fourier expression that demonstrates the modulation rule is given by

$$e^{ia\theta} f(\theta) = \sum_{n=-\infty}^{\infty} f_n e^{ia\theta} e^{in\theta} = \sum_{n=-\infty}^{\infty} f_n e^{i(n+a)\theta} = \sum_{k=-\infty}^{\infty} f_{k-a} e^{ik\theta} \tag{7.13}$$

where a is an integer and thus the Fourier pair here is

$$e^{ia\theta} f(\theta) \Leftrightarrow f_{n-a}\,. \tag{7.14}$$

7.5.2 Method of execution

As previously stated, this rule is associated with Fourier series and as such the modulation rule is implemented within the 1D Fourier series transform procedure. The procedure checks the entered expression to see if there is a complex exponential

multiplying a function dependent on the input variable. Neither the complex exponential nor the original function is independent of the input variable.

Once the expression is confirmed to contain all the components necessary for the application of the modulation rule, the Fourier coefficients is obtained by applying the Fourier series procedure with a shifted output variable i.e. for the expression, $e^{i2\theta} f(\theta)$ the Fourier coefficient is evaluated using the command line:
'*FS1D(* $f(\theta), \theta, n-2$, *range, coefficientComplex)*'.

Similar to the Shift rule, the modulation has only been implemented for the product of the exponential and the original function without any added components such as scalar multiplication, etc. It can however be generalized to include all scenarios.

7.5.3 Implementation

The $e^{iax} f(x)$ expression is tested in this subsection where $a = 5$ and $f(x) = \delta(x-1)$. This permits the verification of the modulation rule.

7.5.3.1 Examples of the Modulation rule in 1D Fourier series transform

The Fourier coefficient of the function $f(x) = \delta(x-1)$ (which is known from subsection 7.4.3.1) is first evaluated after which the complex exponential-weighted version is found. It is obvious from the results that the Fourier coefficient of the product of the complex exponential and the function is the shifted Fourier coefficient of the original function.

> *FS1D*(Dirac$(x-1), x, n, -2..2,$ *coefficientComplex*)

$$\frac{1}{4} e^{-I n}$$

> *FS1D*(Dirac$(x-1)\cdot\exp(I\cdot 5\cdot x), x, n, -2..2,$ *coefficientComplex*)

$$\frac{1}{4} e^{-I (n-5)}$$

The modulation rule is also applied to the product of two exponential functions. To verify the result, the product is simplified into a single exponential and the Fourier coefficient obtained by direct evaluation.

> *FS1D*(exp$(I\cdot 2\cdot x)$, $x, n, -2..2,$ *coefficientComplex*)

$$\begin{cases} 1 & n = 2 \\ 0 & otherwise \end{cases}$$

> $FS1D(\exp(I\cdot 2\cdot x)\cdot\exp(I\cdot 3\cdot x), x, n, -2..2, coefficientComplex)$

$$\begin{cases} 1 & n = 5 \\ 0 & otherwise \end{cases}$$

> $FS1D(\exp(I\cdot(2+3)\cdot x), x, n, -2..2, coefficientComplex)$

$$\begin{cases} 1 & n = 5 \\ 0 & otherwise \end{cases}$$

7.6 Convolution-Multiplication

Convolution is a complex operation that is preferably avoided. This is often why integral transforms are used convert such complicated operations to simple ones in the associated transform space. In Fourier transforms, the full 2D convolution simply becomes multiplication as proven in section (3.2.5) of Chapter 3.

The reverse of this rule is also true i.e. multiplication transforms into convolution, and is therefore not implemented as a rule in this toolbox as convolution is an operation to be avoided. The converse of the rule is also not implemented since it is not useful in simplifying the transform of a product since it is just as computation-intensive and time-consuming as the direct transform of the product of two functions.

Implemented and tested in this toolbox are two special cases: 1) the Fourier transform of the 2D convolution of two radially symmetric functions and 2) the Fourier transform of the 2D convolution of a radially symmetric function and a non-radially symmetric function.

7.6.1 Theory

The full 2D convolution is Fourier transformed (see details in section (3.2.5) of Chapter 3) to give the Fourier pair,

$$f(\vec{r}) ** g(\vec{r}) = \int_{-\infty}^{\infty} g(\vec{r}_0) f(\vec{r} - \vec{r}_0) d\vec{r}_0 \;\Leftrightarrow\; G(\vec{\omega}) F(\vec{\omega}) . \tag{7.15}$$

For two radially symmetric functions that depend solely on the radial space variable, the Fourier transform of the 2D convolution is shown to also depend solely on the radial frequency variable in the Fourier pair given in equation (7.16).

$$f(\vec{r}) ** g(\vec{r}) = f(r) ** g(r) \;\Leftrightarrow\; F(\rho)G(\rho). \tag{7.16}$$

The transform of the 2D convolution of a radially symmetric function $f(r)$ and a non-radially symmetric one $g(\vec{r})$ is represented by the same Fourier pair given in equation (7.15). The Fourier coefficients of the transform of the convolution is however, given by

$$H_k(\rho) = G_k(\rho) F(\rho). \tag{7.17}$$

where $h(\vec{r}) = f(r) ** g(\vec{r})$.

7.6.2 Method of execution

Convolution (in its various forms) is an evaluable procedure in *SCAToolbox* and will be executed when entered as an expression to be Fourier transformed. To avoid the direct evaluation of the convolution therefore, it must be specifically entered as a string i.e. write it as '*Conv*' instead of *Conv*. The single quotes '' define a string and will therefore prevent its execution. Once this is done, the arguments of the convolution can be picked out and used in the convolution-multiplication rule appropriately.

The Maple commands used in sorting through the arguments include '*op*' and '*nops*'; the former making extraction of particular operands possible while the latter indicates the number of operands in an expression. The transform of a full two dimensional convolution is therefore obtained by evaluating the Fourier transform of the first operand of the convolution expression and multiplying it with that of the second operand.

There also exists a convolution-multiplication rule with regards to the one dimensional Fourier series for the angular convolution of two functions. For two functions, $f(\theta)$ and $g(\theta)$ the angular (circular) convolution is given as

$$f(\vec{r}) *_\theta g(\vec{r}) = \frac{1}{2\pi} \int_0^{2\pi} f(r,\theta_0)\, g(r,\theta-\theta_0)\, d\theta_0. \tag{7.18}$$

The Fourier relationship as discussed in section (2.7) of chapter 3 is defined to be

$$f(\vec{r}) *_{\theta} g(\vec{r}) = \sum_{n=-\infty}^{\infty} f_n(r) g_n(r)\, e^{jn\theta} \tag{7.19}$$

showing that the *n*th Fourier coefficient of the angular convolution of two functions is simply the product of the *n*th Fourier coefficient of each of the two functions. This gives the Fourier pair,

$$f(\vec{r}) *_{\theta} g(\vec{r}) \;\Leftrightarrow\; f_n g_n . \tag{7.20}$$

7.6.3 Implementation

The convolution-multiplication rule is tested with respect to the Fourier transform and Fourier series separately. In the case of the Fourier transform, this rule is tested for the full 2D convolution of two radially symmetric functions and then for the full 2D convolution of a radially symmetric function and a non-radially symmetric function. It is also tested for two functions that do not have radial symmetry.

The Fourier coefficients of the angular convolution of two functions for which their Fourier coefficients are known is evaluated in the example given in subsection (7.6.3.2).

7.6.3.1 Examples of the Convolution-Multiplication rule in 2D Fourier transform in polar coordinates

The Dirac-delta ring of radius $r_0 = 1$ and the Dirac-delta function at $r_0 = 0$, both of which are radially symmetric, are used in this example. The 2D polar Fourier transform of the respective functions are evaluated first before the transform of their convolution (directly and via the rule). We see here that the transform is simply a multiplication and only depends on the radial variable in the Fourier domain as expected.

```
> Polar2DFT(1/r · Dirac(r − 1), r, θ, ρ, ψ)
```

$$\sum_{n=-\infty}^{\infty} 2\,\pi\, e^{-\frac{1}{2} I n \pi} \begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \text{BesselJ}(n, \rho)\, e^{I \psi n}$$

```
> convert( (4.7.26), 'piecewise', n)
```

$$2\,\pi\,\text{BesselJ}(0, \rho)$$

```
> Polar2DFT(1/r · Dirac(r), r, θ, ρ, ψ)
```

$$2\,\pi$$

> $$Polar2DFT\left(Conv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r),\frac{1}{r}\cdot \mathrm{Dirac}(r-1),[r,\theta],2\,dpolar\right),r,\theta,\rho,\psi\right)$$

$$\sum_{n=-\infty}^{\infty} 4\,\pi^2\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}$$

> $$convert\,(\,\mathbf{(4.7.28)},'piecewise',n)$$

$$4\,\pi^2\,\mathrm{BesselJ}(0,\rho)$$

> $$Polar2DFT\left('Conv'\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1),\frac{1}{r}\cdot \mathrm{Dirac}(r),[r,\theta],2\,dpolar\right),r,\theta,\rho,\psi\right)$$

$$2\left(\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}\right)\pi$$

> $$convert\,(\,\mathbf{(4.7.30)},'piecewise',n)$$

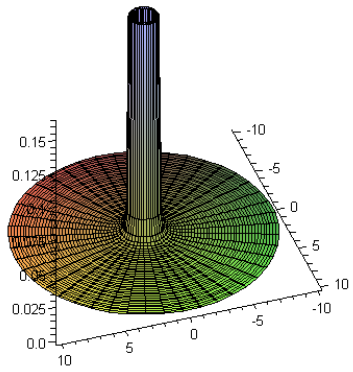
$$4\,\pi^2\,\mathrm{BesselJ}(0,\rho)$$

The plot of the convolution of the Ring-delta and Dirac-delta functions used in this example is given as

> $$Conv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r),\frac{1}{r}\cdot \mathrm{Dirac}(r-1),[r,\theta],2\,dpolar\right)$$

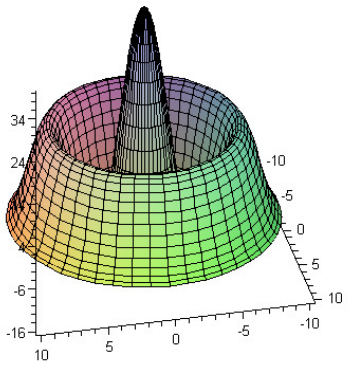
$$2\,\pi\,\mathrm{Dirac}(r-1)$$

> $$plot3d\left(2\cdot \mathrm{Pi}\cdot DiracApprox\,(r-1,0.001),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical,numpoints=2000,axes=framed\right)$$



with the Fourier transform of the convolution giving

> $$plot3d\left(4\cdot\pi^2\cdot \mathrm{BesselJ}(0,\rho),\rho=0..10,\psi=0..2\cdot\pi,coords=z\_cylindrical,numpoints=2000,axes=framed\right)$$



Another example of two radially symmetric functions is the Dirac-delta ring of radius $r_0 = 1$ used above and the function, $f(r) = r$.

> $Polar2DFT(r, r, \theta, \rho, \psi)$

$$\sum_{n=-\infty}^{\infty} 2\,\pi\, e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \left(\int_0^{\infty} r^2\, \mathrm{BesselJ}(n, \rho\, r)\, dr \right) e^{I \psi n}$$

> $convert(\ \mathbf{(4.7.24)}, 'piecewise', n)$

$$2\,\pi \left(\int_0^{\infty} r^2\, \mathrm{BesselJ}(0, \rho\, r)\, dr \right)$$

> $Polar2DFT\left('Conv'\left(\frac{1}{r} \cdot \mathrm{Dirac}(r-1), r, [r, \theta], 2\ dpolar \right), r, \theta, \rho, \psi \right)$

$$\left(\sum_{n=-\infty}^{\infty} 2\,\pi\, e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \mathrm{BesselJ}(n, \rho)\, e^{I \psi n} \right) \left(\sum_{n=-\infty}^{\infty} 2\,\pi\, e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \left(\int_0^{\infty} r^2\, \mathrm{BesselJ}(n, \rho\, r)\, dr \right) e^{I \psi n} \right)$$

> $convert(\ \mathbf{(4.7.22)}, 'piecewise', n)$

$$4\,\pi^2\, \mathrm{BesselJ}(0, \rho) \left(\int_0^{\infty} r^2\, \mathrm{BesselJ}(0, \rho\, r)\, dr \right)$$

In this next example, the non-radially symmetric function used is the 2D impulse function in polar coordinates centered at $r_0 = 1$ and $\theta_0 = \pi$. The same radially symmetric function $f(r) = r$ is used here as in the preceding example. It is clear here that the convolution becomes a simple multiplication.

> $Polar2DFT\left(\frac{1}{r} \cdot \mathrm{Dirac}(r-1) \cdot \mathrm{Dirac}(\theta - \pi), r, \theta, \rho, \psi \right)$

$$\sum_{n=-\infty}^{\infty} I^{-n}\, \mathrm{BesselJ}(n, \rho)\, e^{-I \pi n}\, e^{I \psi n}$$

> $Polar2DFT\left('Conv'\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1)\cdot \mathrm{Dirac}(\theta-\pi), r, [r,\theta], '2dpolar'\right), r, \theta, \rho, \psi\right)$

$$\left(\sum_{n=-\infty}^{\infty} I^{-n}\,\mathrm{BesselJ}(n,\rho)\,e^{-I\pi n}\,e^{I\psi n}\right)\left(\sum_{n=-\infty}^{\infty} 2\pi\, e^{-\frac{1}{2}I n\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(n,\rho\, r)\,dr\right)e^{I\psi n}\right)$$

> $convert\,(\,\mathbf{(4.7.35)}, 'piecewise', n)$

$$2\left(\sum_{n=-\infty}^{\infty} I^{-n}\,\mathrm{BesselJ}(n,\rho)\,e^{-I\pi n}\,e^{I\psi n}\right)\pi\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0,\rho\, r)\,dr\right)$$

Finally to be tested are two non-radially symmetric functions, $\frac{1}{r}\delta(r)\delta(\theta)$ and $e^{i\vec{\omega}_0\cdot\vec{r}}$.

The 2D polar Fourier transforms of the functions are first shown separately before the transform of their convolution is evaluated. The convolution-multiplication rule holds here i.e. convolution simply becomes multiplication when transformed.

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$1$$

> $Polar2DFT\left(sum\left(I^n\cdot \mathrm{BesselJ}(n, 2\cdot r)\cdot \exp(-I\cdot n\cdot \mathrm{Pi})\cdot \exp(I\cdot n\cdot\theta), n=-\infty..\infty\right), r, \theta, \rho, \psi\right)$

$$\frac{4\pi^2\,\mathrm{Dirac}(\rho-2)\,\mathrm{Dirac}(\psi-\pi)}{\rho}$$

> $Polar2DFT\left('Conv'\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), sum\left(I^n\cdot \mathrm{BesselJ}(n, 2\cdot r)\cdot \exp(-I\cdot n\cdot\pi)\cdot \exp(I\cdot n\cdot\theta), n=-\infty..\infty\right), [r,\theta], 2\ dpolar\right), r, \theta, \rho, \psi\right)$

$$\frac{4\pi^2\,\mathrm{Dirac}(\rho-2)\,\mathrm{Dirac}(\psi-\pi)}{\rho}$$

7.6.3.2 Examples of the Convolution-Multiplication rule in 1D Fourier series

The angular convolution of two functions $f(\theta)=1$ and $g(\theta)=\delta(\theta-\pi)$ are used in this example. The Fourier coefficients of each function are first found and then the Fourier

coefficient of the angular convolution is evaluated to verify the relevant convolution-multiplication rule here.

```
> FS1D(Dirac(θ − Pi), θ, n, 0..2·Pi, coefficientComplex)
```

$$\frac{1}{2}\,\frac{e^{-I\pi n}}{\pi}$$

```
> FS1D(1, x, n, 0..2·π, coefficientComplex)
```

$$\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases}$$

```
> FS1D(2·'Conv'(1, Dirac(θ − π), [θ], angular), θ, n, 0..2·π,
      coefficientComplex)
```

$$\begin{cases} \frac{1}{\pi} & n = 0 \\ 0 & otherwise \end{cases}$$

```
> FS1D(2·Conv(Dirac(θ − Pi), 1, [θ], angular), θ, n, 0..2·π,
      coefficientComplex)
```

$$\begin{cases} \frac{1}{\pi} & n = 0 \\ 0 & otherwise \end{cases}$$

The above results validate the accuracy of the rule as implemented in this toolbox. The last example where the convolution is not avoided allows for checking of the results from the rules with that of direct evaluation of the transform of the convolution. The same result is obtained which is an indication that the rule has been properly applied.

7.7 Summary

Table 2 gives a summary of the rules that make up the Fourier operational toolset of Dirac-delta, exponential, spatial shift, multiplication and convolution. Let $f(\theta)$ and $g(\theta)$ be functions of θ, $f(\vec{r})$ and $g(\vec{r})$ be functions of both r and θ in polar coordinates, and $F(\vec{\omega})$ and $G(\vec{\omega})$ be the Fourier transform of $f(\vec{r})$ and $g(\vec{r})$. Here, $*_\theta$ denotes angular convolution and $**$ denotes the full two dimensional convolution.

Table 3: Summary of rules that make up the Fourier operational toolset

Rule for	Fourier series	Fourier transform
Dirac-delta functions	$f(\theta)=\delta(\theta-\theta_0) \Leftrightarrow f_n=\frac{1}{2\pi}e^{-in\theta_0}$	$f(\vec{r})=\frac{1}{r}\delta(r-r_0)\delta(\theta-\theta_0) \Leftrightarrow F(\vec{\omega})=e^{-i\vec{\omega}\cdot\vec{r}_0}$
Complex exponentials	$f(\theta)=e^{in_0\theta} \Leftrightarrow f_n=\delta_{n,n_0}$	$f(\vec{r})=e^{i\vec{\omega}_0\cdot\vec{r}} \Leftrightarrow F(\vec{\omega})=(2\pi)^2\delta(\vec{\omega}-\vec{\omega}_0)$
Scaling	$f\left(\frac{\theta}{a}\right) \Leftrightarrow f_{an}$ where a is an integer, $a\neq 0$	
Shifting	$f(\theta-\theta_0) \Leftrightarrow e^{-in\theta_0}f_n$	$f(\vec{r}-\vec{r}_0) \Leftrightarrow e^{-i\vec{\omega}\cdot\vec{r}_0}F(\vec{\omega})$
Modulation	$e^{ia\theta}f(\theta) \Leftrightarrow f_{n-a}$ where a is an integer	
Convolution-Multiplication	$f(\theta)*_\theta g(\theta) \Leftrightarrow f_n g_n$	$f(\vec{r})**g(\vec{r}) \Leftrightarrow G(\vec{\omega})F(\vec{\omega})$

Chapter 8 : Problems encountered

Being able to build on already existing code suggests that the definitions used in the software match those needed to build the tools for the desired purpose. This was the assumption made at the onset of the development of the toolbox, however this assumption proved incorrect. This section is designed to elaborate on some problems that were encountered while implementing the toolbox, the ways in which these problems were addressed, with the goal being to help the user better understand the inner-workings of a Computer Algebra Software (CAS) system.

Initially, the computational capacity of Maple was overestimated. It was not obvious that the transforms and series were not actually being directly evaluated from their definitions. It eventually became apparent that Maple possesses built-in functionalities that were being used to solve such problems. Maple applies a different approach than direct computation; an idea which was later emulated to solve integral transforms.

8.1 Lookup tables

By examining some of their published code, it was verified that Maple was indeed not programmed to directly compute the integral transforms and series. Thus, the approach for implementing the toolbox had to be revisited. This inspired the idea of the *lookup* table, essentially a table that would contain a list of functions with known transforms.

For the concept to be achieved, a comparison had to be made between the function entered by the user and each of the ones in the left column of the desired table to verify if the desired function was contained as an entry in the table. This meant that all the element and variable *types* specified in the table had to match those of the given input function. It was therefore important to know (to a great extent) the many different element types in Maple. Unfortunately, Maple defines certain element types slightly differently from how they are commonly known. For instance, Maple's *function* type refers to a function call and not in fact to a "mathematical function", as commonly used in most programming languages. . The type *procedure* corresponds to a traditional mathematical "function", as used by most programming languages. Hence, one must be cautious when using these element types. There are also element types that consist of a group of element types. The element type '*algebraic*' is one such type that actually refers to `+`, `*`, `^`, *complex*, *extended numeric*, *function*, *indexed*, *name*, *series* and a few others. Care must be taken when using these to ensure that the element type is not so broad as to allow more types than necessary for a particular function, thus rendering it defective and therefore inaccurate.

8.2 Integral Transforms

The Integral transforms needed in this work include the *n*th order Hankel transform and its inverse as well as the one dimensional Fourier Series *transform* and its corresponding inverse. Maple possesses a built-in Hankel transform and one dimensional Fourier transform. The former, however, is defined in an uncommonly used way, which

does not conform to the needed definition for our application. Maple defines the Hankel transform as

$$\widehat{F}_n(\rho) = \mathbb{H}_n\{f(r)\} = \int_0^\infty f(r)\sqrt{\rho r}J_n(\rho r)dr \tag{8.1}$$

when the mathematical definition used in this application is

$$\widehat{F}_n(\rho) = \mathbb{H}_n\{f(r)\} = \int_0^\infty f(r)J_n(\rho r)rdr\,. \tag{8.2}$$

An attempt was made at modifying the built-in Hankel transform by multiplying the original function $f(r)$ in equation (8.1) by $\sqrt{r/\rho}$ to match the desired definition in equation (8.2) but obtaining closed form results proved difficult once we started building on the altered function. The 1D Fourier transform built into Maple is the standard Cartesian Coordinates and is not applicable to functions given in polar coordinates.

Since these integral transforms are crucial to the toolbox, it was necessary to ensure their proper implementation. A large amount of the Maple code is proprietary and thus unavailable for analysis. These factors led to the decision to directly implement the required transforms, rather than relying on Maple's built-in functions for the integral transforms. This also permitted the implementation of intelligence and memory into the implemented code and also made it easier to verify the code for accuracy and efficiency.

It is clear from the above that there is an added level of complexity and often inaccuracies due to the Maple platform itself. This is not immediately obvious but it is important to make accommodation for such setbacks since one is building on the software of a third party. Having a better understanding of how the specific platform works would be highly desirable.

8.3 Parsing

All Fourier transforms (polar, Cartesian, single and multidimensional) posses a standard "toolset" that can be used to simplify certain classes of transforms. For

example, the multiplication/convolution rule or the shift/modulation rule. The implementation of these *rules* in this toolbox, poses some challenges. Parsing is required in order to apply these rules. The entered expression must be taken apart so that its actual content can be obtained in order to know which specific Fourier transform rule to apply.

8.3.1 Comparing expressions

The implementation of the rules requires the ability to compare part or all of expressions. A standard property (e.g. Dirac-delta, exponential, scaling, etc.) is compared to part or all of the entered expression and if found to be the same, the appropriate rule is applied. There are a number of ideas for achieving this – the concept of *lookup* tables may be used and so can a built-in Maple operation called "applyrules" or by using "*if…else*" statements.

An attempt was made at all these ideas. *Lookup* tables were not used in the end due to the element type issues previously discussed and the multiple variables used in the multidimensional expressions.

The "*applyrules*" operation works for comparing a particular expression with another and not a collection of expressions i.e. *applyrules* matches strings of characters and cannot interpret properties such as element types. This was rather tedious to determine since Maple help did not provide much detail about the procedure.

The rules were therefore implemented via "*if…else*" statements where string manipulation is used in conjunction with manual comparison made for each specific standard rule property.

When a function call is made, Maple tries to simplify the functions within the call as best as it can. An attempt at comparing the expression as entered with any property thus poses a problem since it is the simplified expression that is compared as opposed to the entered expression. The state of the entered expression is changed by Maple and is therefore unknown at the time of the comparison, resulting in a futile effort at parsing in order to match it to a standard property. What the user knows to be the starting point of

the entered expression has been altered, not to the desired final result but to an in-between expression whose properties are unknown to the user.

This is where the need to know the inner workings of the system arose. In the case of our application, the operations of importance were that of *integration* and *series* summation. Examining the publicly available code revealed that there is a sequence of execution of different types of functions given to the *integration* operator. This sequence is mostly proprietary making it difficult to know everything about how it works.

The alternative here is to enter expressions in Maple as strings. This affords the user the design prerogative to analyze and manipulate the entered expression as needed since Maple does not attempt to simplify a string. An example of this is seen when a user wants to obtain the 2D polar Fourier transform of a convolution. The convolution must be input as a string so that instead of 'Polar2DFT(Conv(...))', the appropriate function call becomes 'Polar2DFT('Conv'(...))', so that the string 'Conv' replaces the function call *Conv*. The convolution therefore remains unevaluated and thus unchanged, allowing for the standard convolution property to be matched and the convolution-multiplication rule to be applied accordingly. This also holds for operations that already exist in Maple as symbols. Implementing a rule for any of those will require avoiding the symbolic representation and applying the quote operator to the function. For instance, an integration operation is written as *'int'* instead of $\int$ or *int*.

8.3.2 String manipulations

Some of the operations available in Maple for string manipulation for parsing purposes are simple enough to comprehend but not quite as straight forward to utilize. For example, the '*op*' and '*nops*' operators are used to select specific operands and obtain the number of operands in a function, respectively. However, the problem encountered with these is the inconsistent order in which the operands are retrieved. Obviously, writing a function in a different way may result in a change in the order of the operands. But when the designer must anticipate all possible ways a user might input a function, one must be careful in how the operation is applied. It is important to cover all bases when dealing with interactive systems. For example, for an expression such as $e^{in\theta} f(\theta)$,

a user might input it as shown or reverse the order by writing $f(\theta)\cdot e^{in\theta}$ resulting in different operand positions.

To solve this problem, one must iterate through the function operands each time the operator is used to ensure that the desired data is obtained irrespective of the order in which Maple retrieves it. In other words, avoid having to rely on Maple's order of retrieval or manipulation. Even though having to look through the whole function for each function call is computationally expensive, it is necessary to guarantee the accuracy of results in this case.

8.4 Other problems

There were other trivial problems encountered when developing the toolbox. One of those problems arose when trying to apply a procedure to more than one expression and obtain the results in the same order. Using a *set* for this purpose will return the result in the order of fastest to slowest computation time. This is because a set is not an ordered collection of elements. The appropriate element type to use is a *list*. Using a list ensures that the order of the returned results corresponds with the input expressions.

Another problem encountered in this work had to do with substitutions. There is a tendency to use the Maple operator '*apply*' to perform substitutions. It is however important to note that the *apply* operator is linked to the *function* operator discussed earlier and therefore does not work the same way as a mathematical "function". It constructs a Maple *function* expression. The more appropriate operation to use here is *subs*.

Chapter 9 : Conclusions and Future work

This thesis outlined the design and development of the SCAToolbox for the evaluation of 2D Fourier transforms in polar coordinates, along with many supporting or complimentary functions and procedures. This thesis has shown the many facets of the SCAToolbox. Among the procedures implemented in the toolbox are operations on functions and procedures within the *IntegralTrans* package that make up the supporting functions and integral transforms (along with their associated *lookup* tables).

The operations on functions discussed in this work are all various types of convolutions- one and two dimensional Cartesian convolutions, two dimensional polar convolution, angular/circular convolution, radial convolution and series convolution. The IntegralTrans package contains procedures that help manage the toolbox. These include:

a procedure for retrieving the transform of an expression from the appropriate lookup table, a procedure that allows for adding to an already existing table, a functional Kronecker delta function and an adapted Dirac-delta function.

The integral transforms implemented in this toolbox are the Hankel transform of *n*th order, one dimensional Fourier series transform, the Direct two dimensional polar Fourier transform and the 2D Fourier transform in polar coordinates obtained via the Hankel and Fourier series transforms.

The proposed design criteria were in the form of characteristics that may have several levels of attainment. These are the standards by which the designed toolbox is measured and judged, as discussed here.

The first criterion considered is modularity. The different components of the SCAToolbox are indeed designed separately. All procedures such as those for the different types of convolutions, managing tools (e.g. taking a look into tables, adding to tables, etc.), and the integral transforms are well-defined and most of them can stand independently. Procedures are implemented and tested independently before being put together to work successfully.

Work has been done to maintain (and improve) the toolbox since it was first created, and due to its modular nature this has proven to be relatively simple. Although the toolbox represents the effort of a single person at the moment, it would have been simpler to divide up the work between several people due to the modular nature of the toolbox. The development of the toolbox has been such that it can be open for re-use or maintenance from anywhere. A high level of modularity can therefore be said to have been achieved.

Putting the independent components of the toolbox together was rather straightforward due to the adequate code structure, proper definition of these components and the modularity of the system. The components were also stored as a single unit (or a subset of the file) in the same location. The level of integrability into the toolbox is relatively high.

High levels of extensibility are also attained with the design of this toolbox. The code has been structured and components grouped in a way that is intuitive, meaning that new procedures or algorithms can be added in appropriate locations to match the architecture. For instance, the integral transforms are grouped together in the *IntegralTrans* subpackage. Procedures to implement other integral transforms can easily be added to this subpackage which would in fact be the logical location. Remember tables help build a database of known results for all procedures/functions. However, manually adding to an existing lookup table is also possible by the use of one of the procedures for managing the toolbox discussed above.

The functionality of all parts of the toolbox is very important. As seen from examples provided throughout this work, all procedures have produced accurate results. There has indeed been a trade-off seen between simplicity of output and performance. For instance, in order to obtain the "*simplest*" possible closed form result when finding the 2D polar Fourier transform of a few functions, a table was necessary (as opposed to directly evaluating the transform). The Fourier transform was initially created without a lookup table; the reasoning being that the proper, comprehensive implementation of the three basic functions on which it depends and their rules were enough to simplify the result as desired. Though, this seemed logical at the time, it was soon clear that it did not always provide the simplest result.

Even though the SCAToolbox does possess a high level of functionality, the current level of performance can be said to be medium. There can still be more improvements made on the overall efficiency of the system by adding more functions to the database of known function-transform pairs (i.e. implement more examples), and including new algorithms, among other things. One must be careful in ensuring that new algorithms and code are implemented in an efficient way so as not to diminish the efficiency of the toolbox as a whole. Additions made to improve system efficiency or at the very least leave it unchanged.

Having been able to develop this toolbox would allow for great strides to be made in several areas over a wide range of applications where the multidimensional

convolution is required. Being able to efficiently evaluate two dimensional Fourier transforms in polar coordinates makes this an extremely useful tool in science and engineering as they have proven useful in different areas such as ultrasound, photoacoustics and thermography (image reconstruction). The availability of the polar version of the Fourier operational toolset makes it at least as useful as its Cartesian counterpart.

The multidimensional Fourier transform is also used to transform (partial) differential equations to ordinary algebraic equations as per the rules that apply to the derivatives of functions. Partial differential equations are often used to model multidimensional systems i.e. problems that involve functions of several variables. This makes this toolbox potential quite useful in applications that deal with sound or heat propagation, electrodynamics, fluid flow and elasticity, for instance.
Fourier transforms help represent a signal very compactly making this toolbox useful when processing radio waves, audio, light waves, etc. It would therefore be relevant when a component of a complex waveform needs to be picked out for easier detection and/or removal.

The overall goal of developing and implementing a working and efficient symbolic computer algebra toolbox to evaluate the dimensional Fourier transform in polar coordinates has been successfully achieved.

References

[1] R. N. Bracewell, *The Fourier transform and its applications*. McGraw Hill, 2000.

[2] G. S. Chirikjian and A. B. Kyatkin, *Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups*, 1st ed. CRC Press, 2000.

[3] Yuan Xu, Minghua Xu, and L. V. Wang, "Exact frequency-domain reconstruction for thermoacoustic tomography. II. Cylindrical geometry," *Medical Imaging, IEEE Transactions on*, vol. 21, no. 7, pp. 829-833, 2002.

[4] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Elad, and M. Israeli, "Fast and accurate Polar Fourier transform," *APPL. COMPUT. HARMON. ANAL.*, vol. 21, p. 145--167, 2006.

[5] N. Baddour, "Operational and convolution properties of two-dimensional Fourier transforms in polar coordinates," *Journal of the Optical Society of America A*, vol. 26, no. 8, p. 1767, Jul. 2009.

[6] D. H. Hook, J. M. Norman, and M. R. Williams, *Origins of cyberspace: a library on the history of computing, networking, and telecommunications*. Norman Publishing, 2002.

[7] B. Caviness, "Computer algebra: Past and future," in *EUROCAL '85*, 1985, pp. 1-18.

[8] R. J. Fateman, "A Review of Mathematica," *J. SYMBOLIC COMP*, vol. 13, p. 545--579, 1993.

[9] S. M. Watt, "Making computer algebra more symbolic," *in Proc. Transgressive Computing 2006: A conference in honor or Jean Della Dora*, p. 43--49, 2006.

[10] B. W. Char, G. J. Fee, K. O. Geddes, G. H. Gonnet, and M. B. Monagan, "A tutorial introduction to Maple," *Journal of Symbolic Computation*, vol. 2, no. 2, pp. 179-200, Jun. 1986.

[11] T. Beth, "Algebraic and symbolic computation in digital signal processing, coding and cryptography," in *EUROCAL '85*, 1985, pp. 93-101.

[12] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*. Addison-Wesley publ. company, 1976.

[13] H. J. Nussbaumer, *Fast Fourier transform and convolution algorithms*. Springer-Verlag, 1981.

[14] S. Winograd, *Arithmetic complexity of computations*. SIAM, 1980.

[15] E. R. Berlekamp, *Algebraic coding theory*. Aegean Park Press, 1984.

[16] T. C. Scott and G. J. Fee, "Some applications of Maple symbolic computation to scientific and engineering problems," in *Proceedings of the international symposium on Symbolic and algebraic computation*, Tokyo, Japan, 1990, pp. 302-303.

[17] T. C. Scott, R. A. Moore, M. B. Monagan, G. J. Fee, and E. R. Vrscay, "Perturbative solutions of quantum mechanical problems by symbolic computation," *Journal of Computational Physics*, vol. 87, no. 2, pp. 366-395, Apr. 1990.

[18] F. Vinette and J. Cízek, "The Use of Symbolic Computation in Solving Some Non-Relativistic Quantum Mechanics Problems," in *Symbolic and Algebraic Computation, Lecture Notes in Computer Science*, vol. 358, Springer Berlin/ Heidelberg, 1989, pp. 85-95.

[19] R. Portugal and S. L. Sautú, "Applications of Maple to general relativity," *Computer Physics Communications*, vol. 105, no. 2-3, pp. 233-253, Oct. 1997.

[20] T. C. Scott, R. A. Moore, and M. B. Monagan, "Resolution of many particle electrodynamics by symbolic manipulation," *Computer Physics Communications*, vol. 52, no. 2, pp. 261-281, January.

[21] S. P. Lipshitz, T. C. Scott, and B. Salvy, "On the Acoustic Impedance of Baffled Strip Radiators," *Journal of Audio Engineering Society*, vol. 43, no. 7/8, pp. 573-580, Jul. 1995.

[22] P. Riis, "Optical Measurements of Magnetically-Aligned Asbestos Fibers," *Open Dissertations and Theses*, Sep. 1989.

[23] A. B. Ogunye, "Advanced State Space analysis using computer algebra," in *Proceedings of the American Control Conference*, Seattle, Washington, 1995, vol. 1, pp. 559-563.

[24] A. B. Ogunye, "Process Control and Symbolic Computation: An Overview with Maple V," *Maple Technical Journal*, vol. 3, no. 1, pp. 94-103, 1996.

[25] A. B. Ogunye and A. Penlidis, "State space computations using Maple V," *IEEE Control Systems Magazine*, vol. 16, pp. 70-77, 1996.

[26] N. Munro and P. Tsapekis, "Some recent results using symbolic algebra," in *Proceedings of the IEEE/IEAC Joint Symposium on Computer-Aided Control System Design*, Tucson, Arizona, 1994, pp. 109-116.

[27] T. Ohtani, M. Fukuzawa, and M. Masubuchi, "A CAD system for nonlinear control system design using Mathematica," in *Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, Tucson, Arizona, 1994, pp. 197-204.

[28] "Symbolic computations and their impact on mechanics," *Fluid Dynamics Research*, vol. 6, no. 2, pp. 105-105, Jul. 1990.

[29] A. I. Beltzer, "Engineering Analysis via Symbolic Computation --- A Breakthrough," *Applied Mechanics Reviews*, vol. 43, no. 6, pp. 119-127, Jun. 1990.

[30] T. Herbert and M. M. Yovanovich, *Symbolic computation in fluid mechanics and heat transfer*. Winter Annual Meeting of ASME, Chicago, Illinois: ASME, 1988.

[31] N. I. Ioakimidis, "Semi-numerical iterative series solution of linear algebraic equations with 'MATHEMATICA'," *Communications in Applied Numerical Methods*, vol. 8, no. 7, pp. 421-429, Jul. 1992.

[32] A. I. Lurie and A. Belyaev, *Theory of elasticity*. Springer, 2005.

[33] S. P. Timoshenko and J. N. Goodier, *Theory of Elasticity*, 3rd ed. Tokyo: McGraw-Hill Kogakusha, 1970.

[34] I. S. Sokolnikoff, *Mathematical theory of elasticity*. McGraw-Hill, 1956.

[35] N. I. Ioakimidis, "Application of Mathematica to the direct solution of torsion problems by the energy method," *Computers & Structures*, vol. 43, no. 4, pp. 803-807, May 1992.

[36] Xu Xinhe and Xiao Wendong, "Symbolic computation for discrete event systems," in *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, 1995, vol. 3, pp. 2618-2623.

[37] G. Hartless and L. Leemis, "Computational algebra applications in reliability theory," *Reliability, IEEE Transactions on*, vol. 45, no. 3, pp. 393-399.

[38] J. Jones, N. P. Karampetakis, and A. C. Pugh, “Some applications of Maple in linear systems analysis,” *IEEE Colloquium on Symbolic Computation for Control (Digest No.1996/078)*, pp. 7/1-7/5, Apr. 1996.

[39] P. A. Ralston and M. J. Maron, “Computer algebra system applications in process control,” in *Intelligent Control, 1997. Proceedings of the 1997 IEEE International Symposium on*, 1997, pp. 221-225.

[40] V. G. Ganzha and R. Liska, “Application of the reduce computer algebra system to stability analysis of difference schemes,” in *Proceedings of the third conference on Computers and mathematics*, Boston, Massachusetts, United States, 1989, pp. 119-129.

[41] V. G. Ganzha and E. V. Vorozhtsov, “Application of Computer Algebra Systems for Stability Analysis of Difference Schemes on Curvilinear Grids,” *Journal of Symbolic Computation*, vol. 28, no. 3, pp. 401-433, Sep. 1999.

[42] S. Haitao, N. Zhaodong, L. Yongjun, Z. Honglin, and Z. Tonglei, “Determination of power-time curves of bacterial growth. Study of lowest growth temperature,” *Journal of Thermal Analysis and Calorimetry*, vol. 48, no. 4, pp. 835-839, 1997.

[43] K. A. Presser, D. A. Ratkowsky, and T. Ross, “Modelling the growth rate of Escherichia coli as a function of pH and lactic acid concentration.,” *Applied and Environmental Microbiology*, vol. 63, no. 6, pp. 2355-2360, Jun. 1997.

[44] A. A. Firsov, S. N. Vostrov, A. A. Shevchenko, and G. Cornaglia, “Parameters of bacterial killing and regrowth kinetics and antimicrobial effect examined in terms of area under the concentration-time curve relationships: action of ciprofloxacin against Escherichia coli in an in vitro dynamic model.,” *Antimicrobial Agents and Chemotherapy*, vol. 41, no. 6, pp. 1281-1287, Jun. 1997.

[45] P. Jëmmer, K. Kratz, and N. Read, “Symbolic algebra in the analysis of bacterial populations,” *Mathematical and Computer Modelling*, vol. 30, no. 1-2, pp. 169-178, Jul. 1999.

[46] P. Jemmer, “Symbolic algebra in the analysis of dynamic chemical-kinetic systems,” *Mathematical and Computer Modelling*, vol. 30, no. 5-6, pp. 33-47, Sep. 1999.

[47] A. Peymandoust and G. De Micheli, “Application of symbolic computer algebra in high-level data-flow synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 9, pp. 1154-1165, 2003.

[48] H.-G. Gräbe, “CALI -- A Reduce package for commutative algebra. Version 2.2.1,” 1995.

[49] I. Bakshee, “Control System Professional: Integrated Symbolic Capability,” *http://www.wolfram.com/products/applications/control/*, 2005. [Online]. Available: http://www.wolfram.com/products/applications/control/. [Accessed: 19-Aug-2010].

[50] B. Palancz, Z. Benyo, and L. Kovacs, “Control System Professional Suite,” *IEEE Control Systems Magazine*, vol. 25, no. 4, pp. 67-75, 2005.

[51] J. W. Helton, M. Stankus, and Kronewitter, “NCAlgebra,” *http://www.math.ucsd.edu/~ncalg/*, 2010. [Online]. Available: http://www.math.ucsd.edu/~ncalg/. [Accessed: 19-Aug-2010].

[52] B. Buchberger, “Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory,” in *Multidimensional Systems Theory - Progress, Directions and Open Problems in Multidimensional Systems*, Reidel Publishing Company, 1985, pp. 184-232.

[53] J. F. Canny, *The complexity of robot motion planning*. MIT Press, 1988.

[54] H.-Y. Lee and C.-G. Liang, “Displacement analysis of the general spatial 7-link 7R mechanism,” *Mechanism and Machine Theory*, vol. 23, no. 3, pp. 219-226, 1988.

[55] S. Cetinkunt and B. Ittop, “Computer Automated Symbolic Modeling of Dynamics of Robotic Manipulators with Flexible Links,” *Robotica*, vol. 10, no. 1, pp. 19-24, 1992.

[56] J. Lin and F. L. Lewis, “A Symbolic Formulation of Dynamic Equations For a Manipulator With Rigid and Flexible Links,” *The International Journal of Robotics Research*, vol. 13, no. 5, pp. 454 -466, Oct. 1994.

[57] A. (Arjeh) Cohen and A. J. P. Heck, “Applied computer algebra□: experience from cam design,” World Scientific, 1995, p. 57.

[58] H. R. Pota and T. E. Alberts, "Multivariable transfer functions for a slewing piezoelectric laminated beam," *Journal of Dynamic Systems, Measurement, and Control*, vol. 117, pp. 352-359, 1995.

[59] S. Basu, R. Pollack, and M.-F. Roy, "Computing roadmaps of semi-algebraic sets," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, Philadelphia, Pennsylvania, United States, 1996, pp. 168-173.

[60] M. L. Husty, "An algorithm for solving the direct kinematics of general Stewart-Gough platforms," *Mechanism and Machine Theory*, vol. 31, no. 4, pp. 365-379, May 1996.

[61] J. Grabmeier, E. Kaltofen, and V. Weispfenning, *Computer algebra handbook: foundations, applications, systems*. Springer Science & Business, 2003.

[62] N. J. Higham, M. Konstantinov, and V. Mehrmann, "The sensitivity of computational control problems," *IEEE Control Systems Magazine*, vol. 24, no. 1, pp. 28-43, Feb-2004.

[63] P. Gawthrop, "Symbolic modeling in control," in *CAD for control systems*, Marcel Dekker, 1993, pp. 127-146.

[64] P. J. Gawthrop and D. J. Ballance, "Symbolic algebra and physical-model-based control," *Computing and Control Journal*, vol. 8, 1996.

[65] A. Varga and G. Looye, "Symbolic and numerical software tools for LFT-based low order uncertainty modeling," *IN PROC. CACSD'99 SYMPOSIUM, COHALA*, p. 1--6, 1999.

[66] N. Munro, *The symbolic methods in Control System analysis and design*. Institution of Electrical Engineers, 1999.

[67] O. Akhrif and G. L. Blankenship, "Computer algebra for analysis and design of Nonlinear Control Systems," presented at the American Control Conference, 1987, Minneapolis, MN, 1987, pp. 547-554.

[68] H. van Essen and B. de Jager, "Analysis and design of nonlinear control systems with the symbolic computation system Maple." 1994.

[69] B. de Jager, "The use of symbolic computation in nonlinear control: is it viable?," *IEEE Transactions on Automatic Control*, vol. 40, no. 1, pp. 84-89, Jan. 1995.

[70] K. Schlacher and A. Kugi, “Control of nonlinear descriptor systems, a computer algebra based approach,” in *Nonlinear control in the year 2000 volume 2*, vol. 259, A. Isidori, F. Lamnabhi-Lagarrigue, and W. Respondek, Eds. London: Springer London, 2001, pp. 379-395.

[71] A. B. Ogunye, “SYMCON: Symbolic Computation Controller package for use with Maple V,” in *Proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design*, Dearborn, MI, 1996, pp. 488-494.

[72] M. Boulehmi and J.-L. Calvet, “Using computer algebra for solving some optimal control problems by dynamic programming,” in *Computer aided control systems design (CACSD’97)*, Gent, Belgium, 1997, pp. 33-37.

[73] D. R. Stoutemyer, “Computer algebra for the calculus of variations, the maximum principle, and automatic control,” *International Journal of Computer & Information Sciences*, vol. 8, no. 5, pp. 419-434, Oct. 1979.

[74] M. T. Soylemez and N. Munro, “Pole assignment and symbolic algebra: a new way of thinking,” presented at the UKACC International Conference on Control ’98, Swansea, UK, 1998, vol. 2, pp. 1306-1310.

[75] M. T. Söylemez and İ. Üstoğlu, “Designing control systems using exact and symbolic manipulations of formulae,” *International Journal of Control*, vol. 79, no. 11, p. 1418, 2006.

[76] M. C. De Oliveira and J. W. Helton, “Computer algebra tailored to matrix inequalities,” *IN PROCEEDINGS OF THE 42ND IEEE CONFERENCE ON DECISION AND CONTROL*, p. 4973--4978, 2003.

[77] N. P. Karampetakis and A. I. G. Vardulakis, “Special issue on the use of computer algebra systems for computer aided control system design,” *International Journal of Control*, vol. 79, no. 11, p. 1313, 2006.

[78] M. D. Lutovac and D. V. Tošić, “Symbolic analysis and design of control systems using Mathematica,” *International Journal of Control*, vol. 79, no. 11, p. 1368, 2006.

[79] P. D. F. Gouveia, D. F. M. Torres, and E. A. M. Rocha, “Symbolic Computation of Variational Symmetries in Optimal Control,” *math/0604072*, Apr. 2006.

[80] P. D. F. Gouveia and D. F. M. Torres, "Automatic Computation of Conservation Laws in the Calculus of Variations and Optimal Control," *Comput. Methods Appl. Math.*, vol. 5, no. 4, pp. 387-409, Sep. 2005.

[81] Cuomo and Oppenheim, "Circuit implementation of synchronized chaos with applications to communications," *Physical Review Letters*, vol. 71, no. 1, pp. 65-68, Jul. 1993.

[82] K. M. Cuomo, A. V. Oppenheim, and S. H. Strogatz, "Synchronization of Lorenz-based chaotic circuits with applications to communications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal processing*, vol. 40, no. 10, pp. 626-633, Oct. 1993.

[83] A. Iglesias, "A new scheme based on Semiconductor Lasers with Phase-Conjugate Feedback for Cryptographic Communications," in *EurAsia-ICT 2002: Information and Communication Technology*, vol. 2510, H. Shafazand and A. M. Tjoa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 135-144.

[84] A. Iglesias, J. M. Gutiérrez, J. Güémez, and M. A. Matías, "Chaos suppression through changes in the system variables and numerical rounding errors," *Chaos, Solitons & Fractals*, vol. 7, no. 8, pp. 1305-1316, Aug. 1996.

[85] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi, "Application of symbolic computer algebra to arithmetic circuit verification," in *Computer Design, 2007. ICCD 2007. 25th International Conference on*, Lake Tahoe, CA, 2007, pp. 25-32.

[86] D. V. Tosic and M. D. Lutovac, "Symbolic simulation of engineering systems," in *Circuits and Systems for Communications, 2008. ECCSC 2008. 4th European Conference on*, Bucharest, Romania, 2008, pp. 59-68.

[87] L. Sevastianov, D. Kulyabov, and M. Kokotchikova, "An application of computer algebra system Cadabra to scientific problems of physics," *Physics of Particles and Nuclei Letters*, vol. 6, no. 7, pp. 530-534, Dec. 2009.

[88] B. Gebremariam, S. K. Bogner, and T. Duguet, "Symbolic computation of the Hartree-Fock energy from a chiral EFT three-nucleon interaction at N2LO," *Computer Physics Communications*, vol. 181, no. 6, pp. 1167-1177, Jun. 2010.

[89] T. Beth, "Algebraic and symbolic computation in digital signal processing, coding and cryptography," in *EUROCAL '85*, 1985, pp. 93-101.

[90] A. Louri and K. Hwang, "A bit-plane architecture for optical computing with two-dimensional symbolic substitution," *ACM SIGARCH Computer Architecture News*, vol. 16, pp. 18–27, May 1988.

[91] S. Can and A. Unal, "Transfer functions for nonlinear systems via Fourier-Borel transforms," in *Circuits and Systems, 1988., IEEE International Symposium on*, 1988, pp. 2445-2448 vol.3.

[92] D. Rodriguez, "Tensor product algebra as a tool for VLSI implementation of the discrete Fourier transform," in *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, 1991, pp. 1025-1028 vol.2.

[93] V. G. Ganzha, "Symbolic-numerical computation of the stability regions for Jameson's schemes," *Mathematics and Computers in Simulation*, vol. 42, no. 4-6, pp. 607-615.

[94] V. F. Edneral, "A symbolic approximation of periodic solutions of the Henon–Heiles system by the normal form method," *Mathematics and Computers in Simulation*, vol. 45, no. 5-6, pp. 445-463.

[95] M. Maslen, "Automation of the lifting factorisation of wavelet transforms," *Computer Physics Communications*, vol. 127, no. 2-3, pp. 309-326, May 2000.

[96] C. A. Felippa, "Customizing the mass and geometric stiffness of plane beam elements by Fourier methods," *Engineering Computations*, vol. 18, no. 1/2, pp. 286-303, Jan. 2001.

[97] Y. Gil, Z. Gutterman, S. Onn, and I. Yavneh, "Automated Transformations for PDE Systems with Application to Multigrid Solvers," *SIAM Journal on Scientific Computing*, vol. 24, no. 3, p. 886, 2003.

[98] Y. Lucet, "What Shape Is Your Conjugate? A Survey of Computational Convex Analysis and Its Applications," *SIAM Journal on Optimization*, vol. 20, no. 1, p. 216, 2009.

[99] M. D. Lutovac, J. D. Ćertić, and L. D. Milić, "Digital Filter Design Using Computer Algebra Systems," *Circuits, Systems and Signal Processing*, vol. 29, no. 1, pp. 51-64, Jun. 2009.

[100] W. L. Briggs and V. E. Henson, *The DFT: an owner's manual for the discrete Fourier transform*. SIAM, 1995.

[101] J. Cooley and J. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297-301, 1965.

[102] R. Piessens, "The Hankel Transform," in *The Transforms and Applications Handbook, Second Edition*, 2nd ed., CRC Press, 2000, pp. 9.1 - 9.30.

[103] G. B. Arfken and H.-J. Weber, *Mathematical methods for physicists*. Elsevier, 2005.

[104] "Help - Maplesoft." [Online]. Available: http://www.maplesoft.com/support/help/. [Accessed: 11-May-2011].

[105] R. Piessens, "The Hankel Transform," in *The Transforms and Applications Handbook*, 2nd ed., A. D. Poularikas, Ed. CRC Press, 2000.

Appendices

Appendix A

Maple ideas and commands

library - a Maple repository for storing procedures and packages for later retrieval

package - a collection of routines/functions and possibly other data. It usually provides a range of functionality to solve problems in some well-defined problem domain.

with - a Maple command that loads a particular package for use. It makes the short form names of the specified commands within the Maple package available at the interactive level.

unwith - removes the package exports from the global namespace. It cancels the effects of the "*with*" command

procedures - the name of a valid 'function call' in Maple/ a valid expression that can be assigned to a name. This is what is often referred to as a mathematical function. Examples of expressions of this type include

```
> type(sin, procedure)
                              true
> type(sin(x), procedure)
                              false
> type(f, procedure)
                              false
> type(f(x), procedure)
                              false
> g(x) := x^2
                           g := x → x^2
> type(g, procedure)
                              true
> type(g(x), procedure)
                              false
```

functions - application of a procedure to arguments. It is equivalent to what is known as a 'function call'. Examples:

```
> type(sin, function)
                                false
> type(sin(x), function)
                                true
> type(f, function)
                                false
> type(f(x), function)
                                true
> h(x) := x^3
                              h := x → x^3
> type(h, function)
                                false
> type(h(x), function)
                                false
```

operators - functional operators are special forms of functions in Maple; often written using the *arrow* notation. Examples:

```
> type(sin, operator)
                                false
> type(sin(x), operator)
                                false
> type(f, operator)
                                false
> type(f(x), operator)
                                false
> p(x) := x^3
                              p := x → x^3
> type(p, operator)
                                true
> type(p(x), operator)
                                false
```

proc - command for defining a procedure in Maple. There are many different parameters that can be specified in a procedure such as the formal parameters specifying what to enter to the procedure, the type of the return value expected, a description of the procedure, the local and global variables, among others.

map - a Maple command that applies a procedure to each operand of an expression

table - any compilation of the values of a function for a range of arguments. Tables are created in Maple in one of three ways: 1) using the "*table*" constructor, 2) assigning it to an indexed name (implicitly created) or 3) calling a procedure with the "*remember*" option

lookup tables - tables with two columns that are related by some operation or function. A user may obtain results from the second column of such tables if the first column expression matches the one in question.

compiletable(table_name)

- a Maple command for creating a lookup table

insertpattern - a Maple command that appends to an already existing lookup table

patmatch - a Maple command that allows the matching of a specific pattern to the entered expression

LibraryTools - a collection of utilities for manipulating libraries

march('create', "C:\\toolbox_path");

- creates a library archive in Maple. "*march*" is a library managing command that can also be used to add/insert variables or files, remove a variable or file from a repository, list the contents of a library, among other things

libname - allows for accessing a given repository

savelibname - assigns the location/library to save to. This allows for saving easily without repeating the library path each time in the same document.

savelib - a Maple command that saves a file to a specified repository

apply - applies a procedure to arguments in order to construct a function in Maple

subs - Maple command that substitutes one subexpression for another in a given expression

eval - evaluates an expression at given point(s). This Maple command produces mathematically sounds results which is not always the case with "s*ubs*".

simplify - Maple command that applies simplification rules to an expression. These simplification rules could either be those built into Maple or user-defined. Some groups of such rules include 'trig' (trigonometric expressions), 'ln' (expressions involving logarithms), 'sqrt' (square roots), 'Ei' (expressions involving exponential integrals), etc.

convert - helps convert an expression to a different form. One may choose to change units, numerical types, or coordinate systems; all of which can be done with "*convert*".

piecewise - allows for defining piecewise-continuous functions

nargs - gives number of arguments passed to a procedure

nops - gives number of operands of an expression

op - extracts operand(s) from an expression

trace - a debugging tool that allows the user to the execution of a procedure

plot3d - allows for three-dimensional plotting

- serves as an indication that whatever comes after should not be evaluated; for commenting

; - indicates the end of a line/command

:: - used to assign types

:= - referred to as the *hard assign*; gives a variable (LHS) a value (RHS)

Appendix B

Constructing a toolbox

B.1: Building your own toolbox

First and foremost, it is important to know the path to which you wish to develop and save the toolbox. This is where you would store everything in your toolbox. This is quite crucial because it allows for easy access to the toolbox from any page, a huge plus for manipulating and using any toolbox. A toolbox could grow in size very quickly when written in a one-page file. Knowing the path to store the contents of this toolbox allows you to easily add and save procedures, tables, etc. to the toolbox from any file at any time. Once you have identified the location and therefore path, you can create the toolbox using appropriate program-specific syntax. Upon creating the toolbox, you can begin initializing packages, procedures or tables to be contained in it.

In Maple the command for creating a toolbox is "*march('create', "C:\\toolbox_path");*". Packages are initialized as tables i.e. "*package_name := table();*" and you can also add to them as well as edit whatever they contain. As shown in Figure 2, operations/procedures can be added to the toolbox without being put in packages. Note: The number sign is used to make comments in Maple and anything after a number sign is left unevaluated.

There usually exists an effective way of saving packages, procedures, tables, etc. to a toolbox without having to state the path repeatedly. In Maple this is achieved by assigning the path to save to as follows: "*savelibname :="C:\\toolbox_path";*" and then each time you need to save to this path, you use "*savelib('procedure_name');*". Of course this can be done using the long version of having to write the path of the toolbox each time you need to save.

Maple also possesses a library called "*LibraryTools*" that can be used to manipulate libraries. The "*with*" operator is used to load this library as you would any other library or toolbox such as your own. To leave the toolbox, "*unwith*" is used. The long form of using the procedures in the library is to use the library name each time i.e. "*library_name[procedure_name](arguments);*".

The step by step process of building a toolbox is therefore summarized as follows:

1. Identify filename and desired path of toolbox
2. Open a new empty document and create the toolbox as it pertains to the your chosen CAS system
3. Initialize packages, procedures and/ or tables to be included in toolbox
4. Define previously listed tools
5. Edit said tools where necessary
6. Save tools in sub-packages and/ or toolbox as desired
7. Save toolbox at afore mentioned location

In order for anyone to use the toolbox after it is built, its location must be loaded whenever the system is restarted (or when a different sheet/page is opened). The toolbox can be accessed multiple times this way. The calling sequence that accomplishes this in Maple is "*libname :="C:\\toolbox_path";*". There are similarities between building your own toolbox from scratch and uploading an already existing one. The latter is discussed in the following subsection.

B.2: Uploading existing toolbox: SCAToolbox

As previously mentioned, ensuring that the appropriate CAS software (one compatible with the toolbox to be uploaded) is properly installed is crucial. It is also vital to know the path to which you wish to download and save the toolbox so that it is easily accessible from anywhere and thereby, making it rather easy to use and manage.
Knowing the location of the toolbox allows for adding new functionality and building on existing operations of said toolbox to suite your needs.

Once you have successfully downloaded the toolbox unto your system, you can proceed to open and use it. SCAToolbox is accessible online via the link: ..\..\..\Maple docs\IntegralTrans_package worksheet4.mw.
In Maple, "*libname:=libname,"C:\\toolbox_path");*"is the command line for initializing a toolbox. The toolbox must be loaded before it can be used. To do that, use the command line: "*with(package_name)*". The "*unwith*" operator, also from the "*LibraryTools*" library, is used for unloading the package.

Additions can be made to the toolbox as well. Create the desired structure (table, procedure, etc.) as shown in the preceding subsection and then save it to the toolbox by assigning the path to save to "*savelibname :="C:\\toolbox_path";*" and then saving the structure to it "*savelib('procedure_name');*" or by using the long form "*Save('procedure_name', "C:\\toolbox_path");*".

Summarized in steps below is the process of uploading an existing toolbox:

1. Download toolbox
2. Identify desired path for toolbox
3. Open toolbox for use
4. Query and/or add to toolbox
5. Save toolbox at afore mentioned location

Each time the toolbox is needed; the user must initialize it and load it as discussed above. The toolbox can indeed be loaded and used anywhere this way.

Appendix C

Maple code

IntegralTrans_package worksheet4.mw

Creating the Toolbox/ Archive

```
> restart
> # with(LibraryTools) :
> # Create("C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolb
    mla");
> # create archive
> march('create',
    "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
    mla");
> IntegralTrans := table( ) :
> # assign the lib to save to
> savelibname
    := "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbo
    mla":
```

Operations on functions

1D Cartesian Convolution

```
>
OneDCartConv := proc(f :: algebraic, g :: algebraic, t :: algebraic);
 local Conv1D, nf, ng, τ;
 description "1D Convolution in Cartesian coordinates"

 if (not type(f, {constant, name}) and type(f, 'freeof'(t)) ) or (not type(g, {constant, name})
     and type(g, 'freeof'(t)) )
then error "Both functions must be dependent on %3", f, g, t

 else

 nf := subs(t = τ, f); #``(t→f(t))(τ);
     # using apply takes f as an operator ie.   t⇒t(τ) instead of t ⇒ τ
 ng := subs(t = t−τ, g); #``(t→g(t))(t − τ); # return nf.ng;

# 1D cartesian convolution definition
 Conv1D := int(nf·ng, τ = −∞..∞);
 end if;

 return simplify(eval(Conv1D)); #subs(Heaviside(t) = 1, Heaviside(−t) = 0, Heaviside(0) = 1,
     Conv1D);
end proc:


> savelib('OneDCartConv');
> # Save('OneDCartConv',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

2D Cartesian Convolution

```
>
TwoDCartConv := proc(f :: algebraic, g :: algebraic, x :: algebraic, y :: algebraic);
 local Conv2D, nf, ng, x0, y0;
 description "2D Convolution in Cartesian coordinates";

if (not type(f, {constant, name}) and type(f, 'freeof'(x)) and type(f, 'freeof'(y))) or
(not type(g, {constant, name}) and type(g, 'freeof'(x)) and type(g, 'freeof'(y)))
then error "Both functions must be dependent on either %1 or %2", x, y

else

 nf := subs(x = x0, y = y0, f);
 ng := subs(x = x - x0, y = y - y0, g);

 Conv2D := int(int(nf·ng, x0 = -∞..∞), y0 = -∞..∞) assuming -∞ < x < ∞, -∞ < y < ∞;
 end if;

 return simplify(eval(Conv2D));
end proc:

>
savelib('TwoDCartConv');
>
# Save('TwoDCartConv',
     "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
     mla");
>
```

2D Polar Convolution

```
>
TwoDPolarConv := proc(f :: algebraic, g :: algebraic, r :: algebraic, y :: algebraic);
 local ConvPol2D, nf, ng, r0, y0;
 description "2D Convolution in Polar coordinates";

if (not type(f, {constant, name}) and type(f, 'freeof'(r)) and type(f, 'freeof'(y))) or
(not type(g, {constant, name}) and type(g, 'freeof'(r)) and type(g, 'freeof'(y)))
then error "Both functions must be dependent on either %1 or %2", r, y

else
#NumericEventHandler(invalid_operation = `Heaviside/EventHandler`(value_at_zero = 1));

 nf := subs(r = r0, y = y0, f);
 ng := subs(r = r - r0, y = y - y0, g);

 #ConvPol2D := takeAlook(subs(r0 = x, subs(x = y0, takeAlook(subs(y0 = x, nf·ng·r0),
     DiracTable))), DiracTable);
 ConvPol2D := int(int(eval(subs('Dirac(r0)'='EdDirac(r0)', 'Dirac(y0)'='EdDirac(y0)', nf·ng
     ·r0)), y0 = 0..2·π), r0 = 0..∞) assuming 0 < r ≤ ∞, 0 < y ≤ 2·π;
 end if;

 return simplify(eval(ConvPol2D));
end proc:


> savelib('TwoDPolarConv');
> # Save('TwoDPolarConv',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

Angular or Circular Convolution

```
>
AngConv := proc(f :: algebraic, g :: algebraic, θ :: algebraic);
 local ConvAng, nf, ng, θ0;
 description "Angular Convolution,"

if (not type(f, {constant, name}) and type(f, 'freeof'(θ))) or (not type(g, {constant, name})
    and type(g, 'freeof'(θ)))
then error "Both functions must be dependent on %1,"θ

else

 nf := subs(θ = θ0, f);
 ng := subs(θ = θ − θ0, g);

 ConvAng := 1/(2·π) · int(nf·ng, θ0 = 0..2·π) assuming 0 ≤ θ ≤ 2·π;
 end if;

 return simplify(eval(ConvAng));
end proc:

> savelib('AngConv');
> # Save('AngConv',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

Radial Convolution

```
> RadConv := proc(f :: algebraic, g :: algebraic, r :: algebraic);
  local ConvRad, nf, ng, r0;
  description "Radial Convolution";

  if (not type(f, {constant, name}) and type(f, 'freeof'(r))) or (not type(g, {constant, name})
      and type(g, 'freeof'(r)) )
  then error "Both functions must be dependent on %1", r

  else

  nf := subs(r = r0, f);
  ng := subs(r = r - r0, g);

  ConvRad := int(nf·ng·r0, r0 = 0 .. ∞) assuming 0 ≤ r ≤ ∞;
  end if;

  return simplify(eval(ConvRad));
 end proc:

> savelib('RadConv');
> # Save('RadConv',
       "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
```

Series Convolution

```
> SerConv := proc(f :: algebraic, g :: algebraic, n :: algebraic);
  local ConvSer, fn, gn, m;
  description "Series Convolution;"

  if (not type(f, {constant, name}) and type(f, 'freeof'(n))) or (not type(g, {constant, name})
      and type(g, 'freeof'(n)))
  then error "Both functions must be dependent on %1","n

  else

  fn := subs(n = n - m, f);
  gn := subs(n = m, g);
  NumericEventHandler(invalid_operation = `Heaviside/EventHandler`(value_at_zero = 1));

  ConvSer := sum(fn·gn, m = -∞..∞);
  end if;

  return subs(m = n, simplify(ConvSer));

  end proc:

> savelib('SerConv');
> # Save('SerConv',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
```

"Bracket" Convolution

```
> Conv := proc(f :: algebraic, g :: algebraic, s :: list, condition :: anything);
  local res, var1, var2;
  description "Umbrella procedure for convolving functions";

  if nops(s) = 1 then var1 := op(1, s); var2 := var1;
  elif nops(s) = 2 then var1 := op(1, s); var2 := op(2, s);
  end if;

  NumericEventHandler(invalid_operation = `Heaviside/EventHandler`(value_at_zero = 1));

  if (not type(f, {constant, name}) and type(f, 'freeof'(var1)) and type(f, 'freeof'(var2))) or
   (not type(g, {constant, name}) and type(g, 'freeof'(var1)) and type(g, 'freeof'(var2)))
  then error "Both functions must be dependent on an element in the third argument"

  else
   if (condition = '1dcartesian') then res := OneDCartConv(f, g, var1)
   elif (condition = '2dcartesian') then res := TwoDCartConv(f, g, var1, var2)
   elif (condition = '2dpolar') then res := TwoDPolarConv(f, g, var1, var2)
   elif (condition = 'angular') then res := AngConv(f, g, var1)
   elif (condition = 'radial') then res := RadConv(f, g, var1)
   elif (condition = 'series') then res := SerConv(f, g, var1)
   else error "Please enter a valid convolution type"
   end if;

   end if;

   return res;
   end proc:

> savelib('Conv');
> # Save('Conv',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
```

Integral transforms and tables

>

Procedures in *IntegralTrans* package

>

Looking into a transform's table

>

```
IntegralTrans[takeAlook] := proc(expr :: anything, tab)
 local res;
 description "getting a result from a Transform's table;"

# look into the 'tab' table and return what matches the expression
 res := tablelook(expr, tab);

return res;
 end proc:
```

> `savelib('takeAlook');`

> `# Save('takeAlook',`
`"C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.li`
`;`

>

The Kronecker Delta function

>

```
IntegralTrans[EKronecker] := proc(n :: algebraic, m :: algebraic)
description "...the Kronecker Delta";

 return piecewise(n = m, 1);

 end proc:
```

> `savelib('EKronecker');`

> `# Save('EKronecker',`
`"C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.li`
`;`

>

Modifying the Dirac Delta function at end points

(where integration is usually evaluated)

```
> IntegralTrans[EdDirac] := proc(a :: algebraic) return 2·Dirac(a); end proc:
> savelib('EdDirac');
> # Save('EdDirac',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.lib
      ;
>
```

Adding to a transform's table

```
>
  IntegralTrans[addToTable] := proc(pro :: name, expr :: algebraic, ans :: algebraic, r :: name, s
      :: algebraic)
  local pattern, expression, tableEntry;

  if type(pro, 'indexed') then error "%1 must be an unindexed name","pro end if;

  # check that the pattern and expression is are functions of the variables entered (unless just a
      constant)
  if (not type(expr, {constant, name}) and type(expr,'freeof'(r))) or (not type(ans, {constant,
      name}) and type(expr,'freeof'(s)))
   then error "Second and third parameters %1, %2 must be function of %3, %4 resp,"expr, ans, r, s
   end if;

  # adding to the hankel table
  if pro ='Hankel' then
  pattern := subs(r = RR, expr); # use the variables already in the hankel table (uniformity)
  expression := subs(s = ρ, ans);
  tableEntry := pattern = expression;

  # check that the pattern doesn't already exist in the hankel table
   if IntegralTrans[takeAlook](pattern, Integral[HankelTable]) ≠ expression then
    insertpattern(tableEntry, IntegralTrans[HankelTable]);
   end if;

   #elif pro='Fourier' then
    #insertpattern(expr = ans, IntegralTrans[FourierTable])
  end if;

  # ` `find a way to take the expression apart and generalizing before putting into the table
    end proc:


> savelib('addToTable');
> # Save('addToTable',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.lib
      ;
```

Hankel Tables (the *lookup tables*)

>

$$HankelTab0 := \left[aa :: realcons = aa \cdot \frac{\mathrm{Dirac}(\rho)}{\rho}, \frac{a :: atomic}{RR} = \frac{a}{\rho}, RR^{b :: realcons} = \right.$$

$$\begin{cases} \dfrac{2^{b+1} \cdot \Gamma\left(\frac{b}{2} + 1\right)}{\rho^{b+2} \cdot \Gamma\left(-\frac{b}{2}\right)} & \frac{1}{2} < -b < 2 \\ FAIL & otherwise \end{cases}, \mathrm{Heaviside}((bb :: algebraic) - RR) = \frac{bb}{\rho} \cdot \mathrm{BesselJ}(1, bb$$

$$\cdot \rho), \frac{cc :: atomic}{\sqrt{RR^2 + c :: atomic}} = cc \cdot \frac{e^{-\rho \cdot evalf(\mathrm{abs}(\mathrm{sqrt}(c)))}}{\rho}, \frac{dd :: atomic}{RR^2 + d :: radnum} = dd$$

$$\cdot evalf\left(\mathrm{BesselK}(0, \rho \cdot \mathrm{abs}(\mathrm{sqrt}(d)))\right), \frac{1}{\sqrt{RR^4 + (ff :: algebraic)^4}} = \mathrm{BesselK}\left(0, \frac{ff \cdot \rho}{\sqrt{2}}\right)$$

$$\cdot BesselJ\left(0, \frac{ff \cdot \rho}{\sqrt{2}}\right), e^{(gg :: algebraic) \cdot RR} = \frac{(-gg)}{\left(\rho^2 + (-gg)^2\right)^{\frac{3}{2}}}, e^{(g :: algebraic) \cdot RR^2} = \frac{e^{-\frac{\rho^2}{-2 \cdot 2 \cdot g}}}{-2 \cdot g},$$

$$RR^{p :: algebraic} \cdot e^{(hh :: algebraic) \cdot RR} = simplify\left(\frac{2^{p+1} \cdot \rho^{p+1} \cdot \left(p + 1 - \frac{1}{2}\right)!}{\sqrt{\pi} \cdot \left((-hh)^2 + \rho^2\right)^{p+1+\frac{1}{2}}}\right),$$

$$\frac{1 - e^{(h :: algebraic) \cdot RR}}{RR^2} = \log\left(\frac{-h + \sqrt{(-h)^2 + \rho^2}}{\rho}\right), \log\left(1 + \frac{(kk :: algebraic)^2}{RR^2}\right) = \frac{2}{\rho} \cdot \left(\frac{1}{\rho}\right.$$

$$\left. - kk \cdot \mathrm{BesselK}(1, kk \cdot \rho)\right), \frac{\sin((ll :: realcons) \cdot RR)}{RR} = \begin{cases} \dfrac{1}{\sqrt{ll^2 - \rho^2}} & \rho < ll \\ 0 & \rho > ll \end{cases},$$

$$\frac{\sin((l :: realcons) \cdot RR)}{RR^2} = \begin{cases} \frac{\pi}{2} & \rho \le l \\ \arcsin\left(\frac{l}{\rho}\right) & \rho > l \end{cases}, \frac{\sin((mm :: realcons) \cdot RR)}{RR^2 + (m :: algebraic)^2} =$$

$$\begin{cases} \dfrac{\pi}{2} \cdot e^{-mm \cdot m} \cdot \mathrm{BesselI}(0, m \cdot \rho) & 0 < \rho < mm \\ FAIL & otherwise \end{cases},$$

$$\frac{\cos((nn :: realcons)\cdot RR)}{RR^2 + (n :: algebraic)^2} = \begin{cases} \cosh(nn\cdot n)\cdot \mathrm{BesselK}(0, n\cdot\rho) & nn < \rho < \infty \\ FAIL & otherwise \end{cases},$$

$$\frac{1 - \mathrm{BesselJ}(0, (pp :: algebraic)\cdot RR)}{RR^2} = \begin{cases} \log\left(\frac{pp}{\rho}\right) & \rho \le pp \\ 0 & \rho > pp \end{cases},$$

$$\frac{qq :: algebraic}{RR}\cdot \mathrm{BesselJ}(1, qq\cdot RR) = \begin{cases} 1 & 0 < \rho < qq \\ 0 & \rho > qq \end{cases}, \frac{1}{RR}\cdot \mathrm{BesselJ}\left(0, 2\cdot\sqrt{(q :: algebraic)\cdot RR}\right) = \frac{1}{\rho}\cdot \mathrm{BesselJ}\left(0, \frac{q}{\rho}\right)\Bigg]:$$

>

$$HankelTab := \Bigg[aa :: realcons = aa\cdot\frac{\mathrm{Dirac}(\rho)}{\rho}, \frac{a :: atomic}{RR} = \frac{a}{\rho}, RR^{b :: realcons} =$$

$$\begin{cases} \frac{2^{1+b}\cdot\Gamma\left(\frac{\nu+2+b}{2}\right)}{\rho^{2+b}\cdot\Gamma\left(\frac{\nu-b}{2}\right)} & \frac{1}{2} < -b < \nu + 2 \\ FAIL & otherwise \end{cases}, RR^{(bb :: realcons)-1}$$

$$= \frac{2^{bb}\cdot\left(\frac{\nu}{2} + \frac{bb}{2} - \frac{1}{2}\right)!}{\rho^{bb+1}\cdot\left(\frac{\nu}{2} - \frac{bb}{2} - \frac{1}{2}\right)!}, \frac{\sin((cc :: realcons)\cdot RR)}{RR} =$$

$$\begin{cases} \frac{1}{\left(\rho^2 - cc^2\right)^{\frac{1}{2}}}\cdot\sin\left(\nu\cdot\arcsin\left(\frac{cc}{\rho}\right)\right) & \rho > cc \\ \frac{1}{\left(cc^2 - \rho^2\right)^{\frac{1}{2}}}\cdot\frac{\rho^{\nu}}{\left(cc + \left(cc^2 - \rho^2\right)^{\frac{1}{2}}\right)^{\nu}}\cdot\cos\left(\frac{cc\cdot\nu}{2}\right) & \rho < cc \end{cases},$$

$$\frac{\sin((c :: realcons)\cdot RR)}{RR^2} = \begin{cases} \dfrac{\nu^{-1}\cdot\rho^{\nu}}{\left(l+\sqrt{c^2-\rho^2}\right)^{\nu}}\cdot\sin\left(\dfrac{\nu\cdot\pi}{2}\right) & \rho \le c \\ \nu^{-1}\cdot\sin\left(\nu\cdot\arcsin\left(\dfrac{c}{\rho}\right)\right) & \rho > c \end{cases}, \frac{e^{(dd :: algebraic)\cdot RR}}{RR}$$

$$= \frac{\left(\sqrt{\rho^2+(-dd)^2}+dd\right)^{\nu}}{\rho^{\nu}\cdot\sqrt{\rho^2+(-dd)^2}}, \frac{e^{(d :: algebraic)\cdot RR}}{RR^2} = \frac{\left(\sqrt{\rho^2+(-d)^2}+d\right)^{\nu}}{\nu\cdot\rho^{\nu}}, RR^{g :: algebraic}$$

$$\cdot e^{(gg :: algebraic)\cdot RR} = \begin{cases} simplify\left(\dfrac{2\cdot(-gg)\cdot(2\cdot\rho)^{\nu}\cdot\Gamma\left(\nu+\frac{3}{2}\right)}{\left(\rho^2+(-gg)^2\right)^{\nu+\frac{3}{2}}\cdot\sqrt{\pi}}\right) & \nu = g \\ FAIL & otherwise \end{cases},$$

$$RR^{(h :: algebraic)-1}\cdot e^{(hh :: algebraic)\cdot RR} = \begin{cases} simplify\left(\dfrac{(2\cdot\rho)^{\nu}\cdot\Gamma\left(\nu+\frac{1}{2}\right)}{\left(\rho^2+(-hh)^2\right)^{\nu+\frac{1}{2}}\cdot\sqrt{\pi}}\right) & \nu = h \\ FAIL & otherwise \end{cases},$$

$$RR^{k :: algebraic}\cdot e^{(kk :: algebraic)^2\cdot RR^2} = \begin{cases} \dfrac{\rho^{\nu}}{\left(2\cdot(-kk)^2\right)^{\nu+1}}\cdot e^{-\frac{\rho^2}{4\cdot(-kk)^2}} & \nu = k \\ FAIL & otherwise \end{cases},$$

$$\frac{RR^{lll :: algebraic}}{\left(RR^2+(ll :: algebraic)^2\right)^{(l :: algebraic)+1}} =$$

$$\begin{cases} simplify\left(\dfrac{\rho^{l}\cdot ll^{lll-l}}{2^{l}\cdot\Gamma(l+1)}\cdot\mathrm{BesselK}(lll-l, ll\cdot\rho)\right) & \nu = lll \\ FAIL & otherwise \end{cases},$$

$$\frac{RR^{mm::\,algebraic}}{\left(RR^4+4\cdot(m::\,algebraic)^4\right)^{mm+\frac{1}{2}}}=$$

$$\begin{cases} simplify\left(\dfrac{\left(\frac{1}{2}\cdot\rho\right)^{mm}\cdot\sqrt{\pi}}{(2\cdot m)^{2\cdot mm}\cdot\Gamma\left(mm+\frac{1}{2}\right)}\cdot\mathrm{BesselJ}\left(mm,\ m\cdot\rho\right)\cdot\mathrm{BesselK}\left(mm,\ m\cdot\rho\right)\right) & \nu=mm \\ FAIL & otherwise \end{cases},$$

$$\frac{RR^{(nn::\,algebraic)+2}}{\left(RR^4+4\cdot(pp::\,algebraic)^4\right)^{nn+\frac{1}{2}}}=$$

$$\begin{cases} simplify\left(\dfrac{\left(\frac{1}{2}\cdot\rho\right)^{nn}\cdot\sqrt{\pi}}{2\cdot(2\cdot n)^{2\cdot nn-2}\cdot\Gamma\left(nn+\frac{1}{2}\right)}\cdot\mathrm{BesselJ}\left(nn-1,pp\cdot\rho\right)\cdot\mathrm{BesselK}\left(nn-1,pp\cdot\rho\right)\right) & \nu=nn \\ FAIL & otherwise \end{cases},\ \frac{q::\,atomic}{RR}\cdot\mathrm{Dirac}(RR)$$

$$\cdot piecewise(n=0,1)=q\cdot\mathrm{BesselJ}(\nu,0)\cdot piecewise(\nu=0,1)\Big|:$$

```
> HankelTab0 := subs(RR = r::name, HankelTab0):
> HankelTab := subs(RR = r::name, HankelTab):
> # Creating the Hankel tables
> IntegralTrans[HankelTable0] := compiletable(HankelTab0):
> IntegralTrans[HankelTable] := compiletable(HankelTab):
>
> savelib('HankelTable0','HankelTable');
> # Save('HankelTable',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
    mla");
```

The Hankel Transform procedure

```
> # writing a procedure to evaluate the Hankel transform
>
IntegralTrans[Hankel] := proc(expr :: {list, algebraic, equation}, r :: name, s :: algebraic, n
      :: algebraic)
 local x, RR :: positive, res, fin, ect, rules1, xpr;
 option remember;
description "...the Hankel Transform procedure";

# print info to user if necessary
  userinfo(5, 'hankel', Entering hankel with, expr);
# get the 5th argument if the first statement is true and the3rd statement if it fails; #
     ect := 'if'(nargs = 5, args[5], 'INT');

if not type(s,'freeof'(r))
 then error "Third parameter %1 must be independent of second parameter %2", s, r
end if;

# apply hankel to each operand of the expression passed
if type(expr, {list, equation})
 then return map(procname, expr, r, s, n)
elif hastype(expr, float)
 then return evalf(procname(convert(expr, 'rational', 'exact'), r, s, n))
end if;


  x := subs(Heaviside(r) = 1, Heaviside(-r) = 0, Heaviside(0) = 1, expr);
  x := expand(x); # x := expandc(xpr, T);
  x := simplify(x, power); # simplify powers
  x := combine(x, trig); # combine trig terms
```

```
with(IntegralTrans) :
 if type(x,'freeof'(r)) then res := x·takeAlook (1, HankelTable )
else
if (n = 0) then res := takeAlook (x, HankelTable0 );
     #short hand "IntegralTrans[takeAlook]" doesn't always give the desired results!!!
              if (res = FAIL) then res := takeAlook (x, HankelTable ) end if;
 else  res := takeAlook (x, HankelTable ) end if;
 end if;

 fin := eval(res, ν = n);
if (fin = FAIL)
 then xpr := eval(subs('Dirac(r)'='EdDirac(r)', x));
 ect := int(subs(RR = r, ρ = s, xpr)·BesselJ(n, s·r)·r, r = 0 .. ∞ );
 assume(ρ0 :: algebraic );
rules1 := [int(BesselJ(n, s·r)·BesselJ(n, ρ0·r)·r, r = 0 .. ∞ ) = 1/s·Dirac(s − ρ0), int(BesselJ(n,
      s·r)·r, r = 0 .. ∞ ) = 1/(BesselJ(n, 0·r)·s)·Dirac(s)];
ect := applyrule (rules1, ect);


 if (ect = FAIL) then return 'Hankel'(subs(RR = r, normal(x)), r, s, n)
 else  return ect; end if;
else  return subs(RR = r, ρ = s, normal(fin))
end if;

 end proc:
>
>  savelib('Hankel');
>   # Save('Hankel',
       "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
       mla");
>
```

The InvHankel Transform procedure

```
>  # writing a procedure to evaluate the Hankel The InvHankel Transform procedure... DONE!!
```

```
>
IntegralTrans[InvHankel] := proc(expr :: {list, algebraic, equation}, s :: algebraic, r :: name, n
      :: algebraic)
 local res, fin, ect;
 option remember;
description "...the Inverse Hankel Transform procedure";

if not type(s,'freeof'(r))
 then error "Second parameter %1 must be independent of third parameter %2",s, r
end if;

# apply hankel to each operand of the expression passed
if type(expr, {list, equation})
 then return map(procname, expr, s, r, n)
elif hastype(expr, float)
 then return evalf(procname(convert(expr, 'rational', 'exact'), s, r, n))
end if;

res := IntegralTrans[Hankel](expr, s, r, n);

if (res ='Hankel(expr, s, r, n)') then return 'InvHankel'(normal(expr), s, r, n)
 else return res;
end if;

 end proc:


>
>
savelib('InvHankel');
>
# Save('InvHankel',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

1D FS Table (the lookup table)

```
>
FS1DTab := [aa :: realcons = aa·IntegralTrans[EKronecker](n, 0), Dirac(XX) = 1/L, exp(I·(m
      :: algebraic)·XX) = IntegralTrans[EKronecker](n, m), bb :: realcons·Dirac(XX − b
      :: algebraic) = bb/L·exp(−I·n·b)]:
>
>
FS1DTab := subs(XX = x :: name, FS1DTab):
>
# Creating the 1D FS table
>
IntegralTrans[FS1DTable] := compiletable(FS1DTab):
>
>
savelib('FS1DTable');
```

```
> # Save('FS1DTable',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

The 1D Fourier Series procedure

```
> # writing a procedure to obtain the fourier coefficient for FS (exponential, trig) or the sum
>
IntegralTrans[FS1D] := proc(expr :: {list, algebraic, equation}, x :: name, n :: algebraic, ran
      :: range, condition :: anything)
 local c, CC, xpr, XX :: positive, period, newExpr, constCoeffExp, fxnCoeffExp, exprExp, powExp,
      rules1, FS1DRules, Ao, An, Bn, Co, Cn, CnFxn, totCn, fs, ran1, ran2;
 option remember;
description "...the Fourier Series procedure";


 # check that fourth parameter is a range
if not type(ran, range) then error "Fourth parameter %1 must be a range", ran end if;

 # apply fourier series to each operand of the expression passed
if type(expr, {list, equation})
 then if nargs = 4 then return map(procname, expr, x, n, ran)
      elif nargs = 5 then return map(procname, expr, x, n, ran, condition) end if;
elif hastype(expr, float)
 then if nargs = 4 then return evalf(procname(convert(expr, 'rational', 'exact'), x, n, ran))
      elif nargs = 5 then return evalf(procname(convert(expr, 'rational', 'exact'), x, n, ran,
      condition)) end if;
end if;

 period := op(nops(ran), ran) - op(1, ran);
```

```
with(IntegralTrans) :

#if (nargs = 4 #or( nargs = 5 #and condition='coefficientReal' )) #then
 # did not work because variables used elsewhere later are lost in loop
 if (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(x)) and op(0, op(2,
      expr)) ='Conv' and nops(op(2, expr)) = 4 and op(4, op(2, expr)) ='angular') then
CnFxn := op(1, expr)·procname(op(1, op(2, expr)), x, n, ran, coefficientComplex)
     ·procname(op(2, op(2, expr)), x, n, ran, coefficientComplex);
elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(x)) and op(0, op(1,
      expr)) ='Conv' and nops(op(1, expr)) = 4 and op(4, op(1, expr)) ='angular') then
 CnFxn := op(2, expr)·procname(op(1, op(1, expr)), x, n, ran, coefficientComplex)
     ·procname(op(2, op(1, expr)), x, n, ran, coefficientComplex);

elif (op(0, expr) ='Conv' and nops(expr) = 4 and op(4, expr) ='angular') then
  CnFxn := procname(op(1, expr), x, n, ran, coefficientComplex)·procname(op(2, expr), x, n,
      ran, coefficientComplex);

 elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(x)) and op(0, op(2,
      expr)) ='Shift' and nops(op(2, expr)) = 3) then
 CnFxn := op(1, expr)·exp(−I·n·op(3, op(2, expr)))·procname(op(1, op(2, expr)), x, n, ran,
      coefficientComplex);
elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(x)) and op(0, op(1,
      expr)) ='Shift' and nops(op(1, expr)) = 3) then
  CnFxn := op(2, expr)·exp(−I·n·op(3, op(1, expr)))·procname(op(1, op(1, expr)), x, n, ran,
      coefficientComplex);

elif (op(0, expr) ='Shift' and nops(expr) = 3) then
 CnFxn := exp(−I·n·op(3, expr))·procname(op(1, expr), x, n, ran, coefficientComplex);



elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(x)) and op(0, op(2,
      expr)) ='Scale' and nops(op(2, expr)) = 3) then
  ran1 := op(3, op(2, expr))·op(1, ran); ran2 := op(3, op(2, expr))·op(nops(ran), ran);
CnFxn := op(1, expr)·procname(op(1, op(2, expr)), x, n, ran1 ..ran2, coefficientComplex);
elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(x)) and op(0, op(1,
      expr)) ='Scale' and nops(op(1, expr)) = 3) then
 ran1 := op(3, op(1, expr))·op(1, ran); ran2 := op(3, op(1, expr))·op(nops(ran), ran);
CnFxn := op(2, expr)·procname(op(1, op(1, expr)), x, n, ran1 ..ran2, coefficientComplex);

elif (op(0, expr) ='Scale' and nops(expr) = 3) then
  #procname( subs(x = op(3, expr)·x, op(1, expr)), x, n, op(1, ran) / op(3, expr) ..op(nops(ran), ran) / op(3, expr) );
  ran1 := op(3, expr)·op(1, ran); ran2 := op(3, expr)·op(nops(ran), ran);
CnFxn := procname(op(1, expr), x, n, ran1 ..ran2, coefficientComplex);

elif type(expr, 'freeof'(x))
then CnFxn := expr·takeAlook(1, FS1DTable)
     #"free of" not working properly in table- helps avert 'type' issues

 elif (coeff(expr, Dirac(x)) ≠ 0 and type(coeff(expr, Dirac(x)), freeof(x)))
then CnFxn := coeff(expr, Dirac(x))·takeAlook(Dirac(x), FS1DTable);
```

```
elif (coeff(expr, exp(diff(expr, x)/expr · x)) ≠ 0 and type(coeff(diff(expr, x)/expr, I), integer)
    and type(coeff(expr, exp(diff(expr, x)/expr · x)), freeof(x)))
then constCoeffExp := coeff(expr, exp(diff(expr, x)/expr · x));
     CnFxn := constCoeffExp · EKronecker(n, coeff(diff(expr, x)/expr, I));
    #takeAlook(expr/coeffComplex, FS1DTable)

elif (op(0, expr) = `*` and nops(expr) = 2 and not type(op(1, expr), freeof(x)) and
    not type(op(2, expr), freeof(x))) then
if (op(0, op(1, expr)) ='exp') then fxnCoeffExp := op(2, expr); exprExp := op(1, expr);
elif (op(0, op(2, expr)) ='exp') then fxnCoeffExp := op(1, expr); exprExp := op(2, expr); end
    if;
powExp := diff(exprExp, x)/exprExp;
     if (coeff(powExp, I) ≠ 0) then CnFxn := subs(n = n − coeff(powExp, I),
    procname(fxnCoeffExp, x, n, ran, coefficientComplex));
     end if;


else CnFxn := takeAlook(expr, FS1DTable);
end if;

if (CnFxn = FAIL or not assigned(CnFxn)) then

 xpr := subs(Heaviside(x) = 1, Heaviside(−x) = 0, Heaviside(0) = 1, expr);
 xpr := expand(xpr); # epand out expression
 xpr := simplify(xpr, power); # simplify powers
 xpr := combine(xpr, trig); # combine trig terms
 #xpr :=subs(XX = x, normal(xpr)); #substitute in "x" again for the integration

ran1 := op(1, ran); ran2 := op(nops(ran), ran);
if (ran1 = 0 or ran2 = 0) then xpr := eval(subs('Dirac(x)'='EdDirac(x)', xpr));
elif (ran1 ≠ -∞ or ran2 ≠ ∞) then xpr := eval(subs('Dirac(x − ran1)'='EdDirac(x − ran1)',
    'Dirac(x − ran2)'='EdDirac(x − ran2)', xpr));
end if;
Co := 1/period · int(xpr, x = ran);
Cn := 1/period · int(xpr · exp(−I·2·π·n·x/period), x = ran);

#totCn :=convert(piecewise(n = 0, Co, Cn), piecewise, n);

totCn := convert(Co · EKronecker(n, 0) + Cn · (1 − EKronecker(n, 0)), piecewise, n);
#had to find a way to assign n without conflicting with integer n

else totCn := convert(eval(CnFxn, L = period), piecewise, n);
end if;
```

```
#need to incorporate the conversion between Cn and Ao, An, Bn
Ao := 2/period · int(xpr, x = ran);
An := 2/period · int(xpr · cos(2·π·n·x/period), x = ran);
Bn := 2/period · int(xpr · sin(2·π·n·x/period), x = ran);

# to allow for integer values to be entered for n...
if type(n, integer) and n ≠ 0 then fs := sum(subs(n = nn, Cn) · exp(I·nn·x), nn = n);
 elif n = 0 then fs := Co;
 else fs := simplify(convert(sum(totCn · exp(I·n·x), n = -∞..∞), piecewise, n));
 end if;

assume(x0 :: realcons);
rules1 := [exp(2·I·π·n) - 1 = 2·π·n·I·EKronecker(n, 0), Heaviside(x0 - 2·π)
    = Heaviside(x0) - 1]; #the inbuilt kronecker  doesn't simplify at all
  #δ[n, 0] · exp(-2·π·n·I) = I^n · BesselJ(n, 0)

# how should the result be given? depending on condition given...
if (nargs = 5 and condition ='coefficientReal') then return Ao + An, Bn;
elif (nargs = 5 and condition ='coefficientComplex') then return totCn;
    #simplify(applyrule(rules1, expand(totCn)))
elif (nargs = 5 and condition ='series') then return fs;
elif (nargs = 4) then return totCn; #simplify(applyrule(rules1, expand(totCn)))
else
    error "Procedure must have 4 or 5 arguments. Fifth argument can only be 'coefficientReal',
    'coefficientComplex' or 'series'"
end if;

end proc:
```

```
>
>
> savelib('FS1D');
> # Save('FS1D',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

1D Inverse FS Table (the lookup table)

> $$InvFS1DTab := \left[\frac{1}{2\cdot NN\cdot \mathrm{I}} = \begin{cases} -\frac{1}{2}\cdot(\pi + x) & -\pi \le x < 0 \\ \frac{1}{2}\cdot(\pi - x) & 0 \le x < \pi \end{cases}, (-1)^{NN+1}\cdot\frac{1}{2\cdot NN\cdot \mathrm{I}} = piecewise\left(-\pi < x < \pi, \frac{1}{2}\cdot x\right), \frac{1}{(4\cdot NN+2)\cdot \mathrm{I}} = \begin{cases} -\frac{\pi}{4} & -\pi < x < 0 \\ \frac{\pi}{4} & 0 < x < \pi \end{cases}, \frac{1}{2\cdot NN} = piecewise\left(-\pi < x < \pi, -\ln\left(2\cdot\sin\left(\frac{|x|}{2}\right)\right)\right), (-1)^{NN}\cdot\frac{1}{2\cdot NN} = piecewise\left(-\pi < x < \pi, -\ln\left(2\cdot\cos\left(\frac{x}{2}\right)\right)\right), \frac{1}{4\cdot NN+2} = piecewise\left(-\pi < x < \pi, \ln\left(2\cdot\cot\left(\frac{|x|}{2}\right)\right)\right)\right]:$$

> $InvFS1DTab := subs(NN = n :: name, InvFS1DTab):$

> $\#\ Creating\ the\ Hankel\ table$

> $IntegralTrans[InvFS1DTable] := compiletable(InvFS1DTab):$

>

> $savelib('InvFS1DTable');$

> $\#\ Save('InvFS1DTable',$
"C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.mla");

>

The 1D Inverse Fourier Series procedure

> $\#\ writing\ a\ procedure\ to\ obtain\ the\ original\ function\ of\ the\ fourier\ coefficient\ for\ FS$ (exponential) back

>

$IntegralTrans[InvFS1D] := \mathbf{proc}(exprCn :: \{list, algebraic, equation\}, n :: algebraic, x :: algebraic, ran :: range)$
local *expr*, *fxn*, *rules1*, *res*;
option *remember*;
description "...the Inverse Fourier Series procedure";

```
# apply inverse FS to each operand of the expression passed
if type(exprCn, {list, equation})
 then if nargs = 3 then return map(procname, exprCn, n, x)
       elif nargs = 4 then return map(procname, exprCn, n, x, ran) end if;
elif hastype(exprCn, float)
 then  if nargs = 3 then return evalf(procname(convert(exprCn, 'rational', 'exact'), n, x))
       elif nargs = 4 then return evalf(procname(convert(exprCn, 'rational', 'exact'), n, x, ran))
      end if;
end if;

assume(g :: algebraic, h :: algebraic, l :: algebraic, m :: algebraic);
rules1 := [sum(I^-n·BesselJ(n, g·h)·exp(-I·n·l)·exp(I·n·m), n = -∞..∞) = exp(-I·h·g·cos(m
      - l)), sum(exp(I·n·x), n = -∞..∞) = 2·π·Dirac(x)];

expr := subs(Heaviside(x) = 1, Heaviside(-x) = 0, Heaviside(0) = 1, exprCn);
expr := expand(expr); # epand out expression
expr := simplify(expr, power); # simplify powers

 with(IntegralTrans):
if (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(x)) and op(0, op(2, expr))
      ='Conv' and nops(op(2, expr)) = 4 and op(4, op(2, expr)) ='series') then
  if nargs = 3 then fxn := op(1, expr)·procname(op(1, op(2, expr)), n, x, -∞..∞)
     ·procname(op(2, op(2, expr)), n, x, -∞..∞);
  elif nargs = 4 then fxn := op(1, expr)·procname(op(1, op(2, expr)), n, x, ran)
     ·procname(op(2, op(2, expr)), n, x, ran); end if;
 elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(x)) and op(0, op(1,
      expr)) ='Conv' and nops(op(1, expr)) = 4 and op(4, op(1, expr)) ='series') then
  if nargs = 3 then fxn := op(2, expr)·procname(op(1, op(1, expr)), n, x, -∞..∞)
     ·procname(op(2, op(1, expr)), n, x, -∞..∞);
  elif nargs = 4 then fxn := op(2, expr)·procname(op(1, op(1, expr)), n, x, ran)
     ·procname(op(2, op(1, expr)), n, x, ran); end if;
```

```
elif type(expr,'freeof'(n)) then
 if (nargs = 3) then fxn := expr·applyrule(rules1, sum(exp(I·n·x), n = -∞..∞));
 elif (nargs = 4) then fxn := expr·applyrule(rules1, sum(exp(I·n·x), n = ran));
 end if;
else fxn := takeAlook(expr, InvFS1DTable);
 end if;
res := eval(fxn); #subs(XX = x, normal(fxn));

# For three input parameters
if (res = FAIL and nargs = 3) then
fxn := applyrule(rules1, convert(sum(expr·exp(I·n·x), n = -∞..∞), piecewise, n));

# For four input parameters  (fourth parameter is the range of "n")
 elif (res = FAIL and nargs = 4) then
 fxn := applyrule(rules1, convert(sum(expr·exp(I·n·x), n = ran), piecewise, n));

 else fxn := res;
 end if;

res := convert(fxn, piecewise, n);

 return res;

 end proc:
```

```
>
> savelib('InvFS1D');
> # Save('InvFS1D',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
```

2D FT Table

```
>
```

$$FT2DTab := \left[a :: realcons \cdot \frac{1}{RR} \cdot \mathrm{Dirac}(RR) \cdot \mathrm{Dirac}(\theta\theta) = a \cdot 1, aa :: realcons \cdot \frac{1}{RR} \cdot \mathrm{Dirac}(RR - b :: algebraic) \cdot \mathrm{Dirac}(\theta\theta - bb :: algebraic) = aa \cdot sum\left(I^{-n} \cdot \mathrm{BesselJ}(n, \rho\rho \cdot b) \cdot \exp(-I \cdot n \cdot bb) \cdot \exp(I \cdot n \cdot \psi\psi), n = -\infty..\infty\right), c :: realcons \cdot sum\left(I^{n} \cdot \mathrm{BesselJ}(n, d :: realcons \cdot RR) \cdot \exp(-1 \cdot n \cdot dd :: realcons) \cdot \exp(I \cdot n \cdot \theta\theta), n = -\infty..\infty\right) = (2 \cdot \pi)^2 \cdot \frac{1}{\rho\rho} \cdot \mathrm{Dirac}(\rho\rho - d) \cdot \mathrm{Dirac}(\psi\psi - dd)\right] :$$

```
> FT2DTab := subs(RR = r :: name, θθ = θ :: name, FT2DTab) :
> # Creating the 2D FT table
> IntegralTrans[FT2DTable] := compiletable(FT2DTab) :
>
> savelib('FT2DTable');
```

```
> # Save('FT2DTable',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

The 2D Polar Fourier Transform procedure

```
> # writing a procedure to evaluate the 2D Polar F
>
IntegralTrans[Polar2DFT] := proc(expr :: {list, algebraic, equation}, r :: name, θ :: name, ρ
      :: algebraic, ψ :: algebraic)
 local n :: algebraic;
 global fin, fn, Hfn, Fn, res;
 option remember;
description "...the 2D Polar Fourier Transform procedure;"

if not type(ρ,'freeof'(r)) or not type(ψ,'freeof'(θ))
 then
        error "Fourth parameter %1 must be independent of second parameter %2, and fifth
        parameter %3 must be independent of third parameter %4;",ρ, r
     , ψ, θ
end if;

 # apply FT to each operand of the expression passed
if type(expr, {list, equation})
 then return map(procname, expr, r, θ, ρ, ψ)
elif hastype(expr, float)
 then return evalf(procname(convert(expr, 'rational', 'exact'), r, θ, ρ, ψ))
end if;
```

```
with(IntegralTrans) :
    #short hand "IntegralTrans[takeAlook]" doesn't always give the desired results!!!

if (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(r)) and type(op(1, expr),
    freeof(θ)) and op(0, op(2, expr)) ='Conv' and nops(op(2, expr)) = 4 and op(4, op(2,
    expr)) ='2dpolar')
 then res := eval(op(1, expr)·procname(op(1, op(2, expr)), r, θ, ρ, ψ)·procname(op(2, op(2,
    expr)), r, θ, ρ, ψ));
elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(r)) and type(op(1,
    expr), freeof(θ)) and op(0, op(1, expr)) ='Conv' and nops(op(1, expr)) = 4 and op(4,
    op(1, expr)) ='2dpolar')
 then res := eval(op(2, expr)·procname(op(1, op(1, expr)), r, θ, ρ, ψ)·procname(op(2, op(1,
    expr)), r, θ, ρ, ψ));

elif (op(0, expr) ='Conv' and nops(expr) = 4 and op(4, expr) ='2dpolar')
 then res := eval(procname(op(1, expr), r, θ, ρ, ψ)·procname(op(2, expr), r, θ, ρ, ψ));


else
 # Dirac-Delta rule and complex exponentials
fin := takeAlook(expr, FT2DTable);

 if (fin = FAIL) then
 # Get 1D FS coefficient of the expression (func of r and    θ)
fn := FS1D(expr, θ, n, 0..2·π, coefficientComplex);

 # Find nth order HT of fn
Hfn := Hankel(fn, r, ρ, n);

 # Evaluate Fn
 Fn := 2·π·I^(-n)·Hfn;

 # Obtain result
 res := InvFS1D(Fn, n, ψ);


else res := eval(fin, [ρρ = ρ, ψψ = ψ]);
 end if;

end if;

if (res = FAIL)
 then return 'Polar2DFT'(expr, r, θ, ρ, ψ)
 else return res;
end if;

 end proc:

> savelib('Polar2DFT');
> # Save('Polar2DFT',
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
```

The 2D Inverse Polar Fourier Transform procedure

```
> # writing a procedure to evaluate the Inverse 2D Polar F
>
  IntegralTrans[InvPolar2DFT] := proc(expr :: {list, algebraic, equation}, ρ :: name, ψ :: name, r
        :: algebraic, θ :: algebraic)
   local n :: algebraic, s1, s2, s3, s4, s5, res, Fn, HFn, fn, rules1, rules2, rules3;
   option remember;
  description "...the 2D Inverse Polar Fourier Transform procedure";

  if not type(r,'freeof'(ρ)) or not type(θ,'freeof'(ψ))
   then
            error "Fourth parameter %1 must be independent of second parameter %2, and fifth
            parameter %3 must be independent of third parameter %4",r, ρ, θ, ψ
  end if;

   # apply Inverse FT to each operand of the expression passed
  if type(expr, {list, equation})
   then return map(procname, expr, ρ, ψ, r, θ)
  elif hastype(expr, float)
   then return evalf(procname(convert(expr, 'rational', 'exact'), ρ, ψ, r, θ))
  end if;


 with(IntegralTrans) : #short hand "IntegralTrans[...]" doesn't always give the desired results!!!
  # Get 1D FS coefficient of the expression (func of  ρ and  ψ)
 Fn := FS1D(expr, ψ, n, 0..2·π, coefficientComplex);

  # Find nth order HT of Fn
  HFn := Hankel(Fn, ρ, r, n);

  # Evaluate fn
  fn := I^n/(2·π)·HFn;

  # Obtain result
  res := InvFS1D(fn, n, θ);

 if (res = FAIL)
  then return 'InvPolar2DFT'(expr, ρ, ψ, r, θ)
  else return res;
 end if;

  end proc:


>
>  savelib('InvPolar2DFT');
>  # Save('InvPolar2DFT',
       "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
```

The Direct 2D Polar Fourier Transform procedure

```
> # writing a procedure to evaluate the 2D Polar F
>
  IntegralTrans[DirectPolar2DFT] := proc(expr :: {list, algebraic, equation}, r :: name, θ :: name,
        ρ :: algebraic, ψ :: algebraic)
   local n :: algebraic;
   global xpr, res;
   option remember;
  description "...the Direct 2D Polar Fourier Transform procedure;"

  if not type(ρ,'freeof'(r)) or not type(ψ,'freeof'(θ))
   then
          error "Fourth parameter %1 must be independent of second parameter %2, and fifth
          parameter %3 must be independent of third parameter %4;",ρ, r
       , ψ, θ
  end if;

   # apply FT to each operand of the expression passed
  if type(expr, {list, equation})
   then return map(procname, expr, r, θ, ρ, ψ)
  elif hastype(expr, float)
   then return evalf(procname(convert(expr, 'rational', 'exact'), r, θ, ρ, ψ))
  end if;


with(IntegralTrans):
    #short hand "IntegralTrans[takeAlook]" doesn't always give the desired results!!!

if (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(r)) and type(op(1, expr),
     freeof(θ)) and op(0, op(2, expr)) ='Conv' and nops(op(2, expr)) = 4 and op(4, op(2,
     expr)) ='2dpolar')
 then res := eval(op(1, expr)·procname(op(1, op(2, expr)), r, θ, ρ, ψ)·procname(op(2, op(2,
     expr)), r, θ, ρ, ψ));
elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(r)) and type(op(1,
     expr), freeof(θ)) and op(0, op(1, expr)) ='Conv' and nops(op(1, expr)) = 4 and op(4,
     op(1, expr)) ='2dpolar')
 then res := eval(op(2, expr)·procname(op(1, op(1, expr)), r, θ, ρ, ψ)·procname(op(2, op(1,
     expr)), r, θ, ρ, ψ));

elif (op(0, expr) ='Conv' and nops(expr) = 4 and op(4, expr) ='2dpolar')
 then res := eval(procname(op(1, expr), r, θ, ρ, ψ)·procname(op(2, expr), r, θ, ρ, ψ));
```

```
else
 xpr := eval(subs('Dirac(r)'='EdDirac(r)', 'Dirac(θ)'='EdDirac(θ)', expr));

# Obtain result
 res := int(int(r·xpr·exp(-ρ·r·cos(ψ - θ)), θ = 0..2·π), r = 0.. ∞) assuming 0 ≤ ρ ≤ ∞, 0
      ≤ ψ ≤ 2·π;

 end if;

if (res = FAIL)
 then return 'DirectPolar2DFT'(expr, r, θ, ρ, ψ)
 else return res;
end if;

 end proc:
>
>  savelib('DirectPolar2DFT');
>  # Save('DirectPolar2DFT',
       "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>
```

The 2D Polar FSH Transform procedure

(future work)

```
> # writing a procedure to evaluate the 2 D Polar Fourier Series-Hankel Transfor
```

```
>
   IntegralTrans[FSH] := proc(expr :: {list, algebraic, equation}, r :: name, θ :: name, ρ
         :: algebraic, ψ :: algebraic)
    local n :: algebraic;
    global fn, Hfn, Fn, res;
    option remember;
   description "...the 2D Polar FSH Transform procedure";

   if not type(ρ,'freeof'(r)) or not type(ψ,'freeof'(θ))
    then
            error "Fourth parameter %1 must be independent of second parameter %2, and fifth
            parameter %3 must be independent of third parameter %4",ρ, r
        , ψ, θ
   end if;

    # apply FT to each operand of the expression passed
   if type(expr, {list, equation})
    then return map(procname, expr, r, θ, ρ, ψ)
   elif hastype(expr, float)
    then return evalf(procname(convert(expr, 'rational', 'exact'), r, θ, ρ, ψ))
   end if;


with(IntegralTrans):
    #short hand "IntegralTrans[takeAlook]" doesn't always give the desired results!!!

if (op(0, expr) = `*` and nops(expr) = 2 and type(op(1, expr), freeof(r)) and type(op(1, expr),
     freeof(θ)) and op(0, op(2, expr)) ='Conv' and nops(op(2, expr)) = 4 and op(4, op(2,
     expr)) ='2dpolar')
 then Fn := op(1, expr)·procname(op(1, op(2, expr)), r, θ, ρ, ψ)·procname(op(2, op(2, expr)),
     r, θ, ρ, ψ);

elif (op(0, expr) = `*` and nops(expr) = 2 and type(op(2, expr), freeof(r)) and type(op(1,
     expr), freeof(θ)) and op(0, op(1, expr)) ='Conv' and nops(op(1, expr)) = 4 and op(4,
     op(1, expr)) ='2dpolar')
 then Fn := op(2, expr)·procname(op(1, op(1, expr)), r, θ, ρ, ψ)·procname(op(2, op(1, expr)),
     r, θ, ρ, ψ);

elif (op(0, expr) ='Conv' and nops(expr) = 4 and op(4, expr) ='2dpolar')
 then res := eval(procname(op(1, expr), r, θ, ρ, ψ)·procname(op(2, expr), r, θ, ρ, ψ));

 else
 # Get 1D FS coefficient of the expression (func of r and     θ)
fn := FS1D(expr, θ, n, 0..2·π, coefficientComplex);
```

```
# Find nth order HT of fn
Hfn := Hankel(fn, r, ρ, n);

 # Evaluate Fn
 Fn := 2·π·I^-n·Hfn;

 end if;

if (Fn = FAIL)
 then return 'FSH'(expr, r, θ, ρ, ψ)
 else  return Fn;
end if;

 end proc:

>
>  savelib('FSH');
>  # Save('FSH',
        "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
        mla");
>
```

The 2D Inverse Polar FSH Transform procedure

(future work)

```
>  # writing a procedure to evaluate the Inverse 2D Polar F
>
   IntegralTrans[InvFSH] := proc(expr :: {list, algebraic, equation}, ρ :: name, ψ :: name, r
        :: algebraic, θ :: algebraic)
    local n :: algebraic, res, Fn, HFn, fn;
    option remember;
   description "...the 2D Inverse Polar Fourier Transform procedure;"

if not type(r,'freeof'(ρ)) or not type(θ,'freeof'(ψ))
 then
          error "Fourth parameter %1 must be independent of second parameter %2, and fifth
          parameter %3 must be independent of third parameter %4;",r, ρ, θ, ψ
end if;
```

```
# apply Inverse FT to each operand of the expression passed
if type(expr, {list, equation})
 then return map(procname, expr, ρ, ψ, r, θ)
elif hastype(expr, float)
 then return evalf(procname(convert(expr, 'rational', 'exact'), ρ, ψ, r, θ))
end if;


with(IntegralTrans) :#short hand "IntegralTrans[...]" doesn't always give the desired results!!!
 # Find nth order HT of Fn
 HFn := Hankel(expr, ρ, r, n);
 print(HFn);

# Evaluate fn
 fn := I^n/(2·π)·HFn;
 print(fn);

 # Obtain result
 res := InvFS1D(fn, n, θ);

if (res = FAIL)
 then return 'InvFSH'(expr, ρ, ψ, r, θ)
 else return res;
end if;

 end proc:

>
>  savelib('InvFSH');
>  # Save('InvFSH',
       "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla");
>


>
>  savelib('IntegralTrans');
>  # Save('IntegralTrans',
       "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox1.
      mla");
```

Using the Toolbox/ Archive

```
> restart
> libname := libname,
      "C:\\Documents and Settings\\Edem\\My Documents\\Thesis 2010\\Malpe docs\\SCAToolbox.
      mla":
> with(IntegralTrans) :
```

Examples for Convolution

>

Examples for 1D Cartesian Convolution

```
> Conv(2, Dirac(s), [s], 1 dcartesian)
```

$$2$$

```
> Conv(s^2, Dirac(s), [s], 1 dcartesian)
```

$$s^2$$

```
> OneDCartConv(s^2, Dirac(s), s)
```

$$s^2$$

```
> Conv(a, Dirac(s), [s], 1 dcartesian)
```

$$a$$

```
> NumericEventHandler(invalid_operation = `Heaviside/EventHandler`(value_at_zero = 1)) :
  OneDCartConv(Heaviside(x), Heaviside(x)·exp(2·x), x) assuming x ≥ 0
```

$$\frac{1}{2}\, e^{2x} - \frac{1}{2}$$

```
> OneDCartConv(1/x, Dirac(x - 5), x)
```

$$\frac{1}{x-5}$$

```
> OneDCartConv(Heaviside(x), Heaviside(x), x)
```

$$x\ \text{Heaviside}(x)$$

```
> convert( (4.1.1.7) , piecewise, x)
```

$$\begin{cases} 0 & x < 0 \\ x & 0 \le x \end{cases}$$

```
> #trace(OneDCartConv)
>
>
```

Examples for 2D Cartesian Convolution

> $Conv(\mathrm{Dirac}(y), \mathrm{Dirac}(x), [x, y], 2\ dcartesian)$

$$1$$

> $TwoDCartConv(\mathrm{Dirac}(x), \mathrm{Dirac}(y), x, y)$

$$1$$

> $\#Conv(x^2, \mathrm{Dirac}(y), [x, y], 2\ dcartesian)$

> $Conv(a, \mathrm{Dirac}(x) \cdot \mathrm{Dirac}(y), [x, y], 2\ dcartesian)$

$$a$$

> $Conv(\mathrm{Dirac}(x) \cdot \mathrm{Dirac}(y), a, [x, y], 2\ dcartesian)$

$$a$$

> $TwoDCartConv(x \cdot y, \mathrm{Dirac}(x-1) \cdot \mathrm{Dirac}(y-2), x, y)$

$$(x-1)\ (y-2)$$

> $TwoDCartConv(\mathrm{Heaviside}(x) \cdot \mathrm{Heaviside}(y), \mathrm{Heaviside}(x) \cdot \mathrm{Heaviside}(y), x, y)$

$$\begin{cases} x\ y\ \mathrm{Heaviside}(x) & 0 < y \\ 0 & otherwise \end{cases}$$

> $\#trace(TwoDCartConv)$

> $\#TwoDCartConv(a, \mathrm{Dirac}(x) \cdot \mathrm{Dirac}(y), x, y)$

> $\#Conv(\mathrm{Dirac}(x-1) \cdot \mathrm{Dirac}(y), \mathrm{Dirac}(x-2), [x, y], 2\ dcartesian)$

> $\#Conv(\mathrm{Dirac}(x-1), \mathrm{Dirac}(x-2), [x, y], 2\ dcartesian)$

>

>

Examples for 2D Polar Convolution

> $addcoords(z\_cylindrical, [z, r, \theta], [r \cdot \cos(\theta), r \cdot \sin(\theta), z])$

> $NumericEventHandler(invalid\_operation = `Heaviside/EventHandler`(value\_at\_zero = 1)):$

> $Conv\left(\mathrm{Dirac}(r-2), \frac{1}{r} \cdot \mathrm{Dirac}(r-1) \cdot \mathrm{Dirac}(\theta-\pi), [r, \theta], 2dpolar\right)\ \#assuming\ r>0,\ \theta>0$

$$2\ \mathrm{Heaviside}(\theta-\pi)\ \mathrm{Dirac}(r-3)$$

> $convert(\textbf{(4.1.3.1)}, piecewise, \theta)$

$$\begin{cases} 0 & \theta < \pi \\ 2\ \mathrm{Dirac}(r-3) & \pi \le \theta \end{cases}$$

> $TwoDPolarConv\left(\mathrm{Dirac}(r-2), \frac{1}{r} \cdot \mathrm{Dirac}(r-1) \cdot \mathrm{Dirac}(\theta-\pi), r, \theta\right)$

$$2\ \mathrm{Heaviside}(\theta-\pi)\ \mathrm{Dirac}(r-3)$$

> $convert(\textbf{(4.1.3.3)}, piecewise, \theta)$

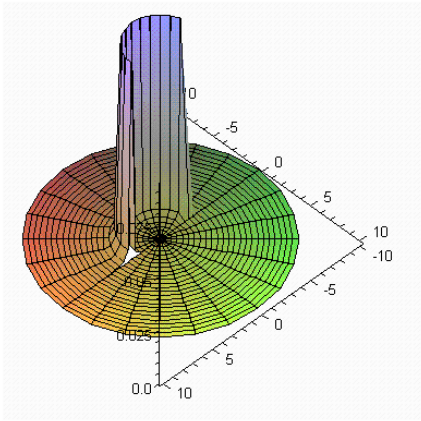
$$\begin{cases} 0 & \theta < \pi \\ 2\ \mathrm{Dirac}(r-3) & \pi \le \theta \end{cases}$$

> $DiracApprox(x, \varepsilon) := \frac{1}{\pi} \cdot \frac{\varepsilon}{(x^2 + \varepsilon^2)}$

$$DiracApprox := (x, \varepsilon) \to \frac{\varepsilon}{\pi\ (\varepsilon^2 + x^2)}$$

> $\#plot3d(4 \cdot \theta \cdot \pi - 4 \cdot \pi^2, r = 0..10, \theta = 0..2 \cdot \pi, coords = z\_cylindrical, axes = framed)$

> $plot3d\left(2\cdot \text{Heaviside}(\theta-\pi)\cdot DiracApprox\,(r-3,0.001)\,,r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical\,,axes=framed\right)$



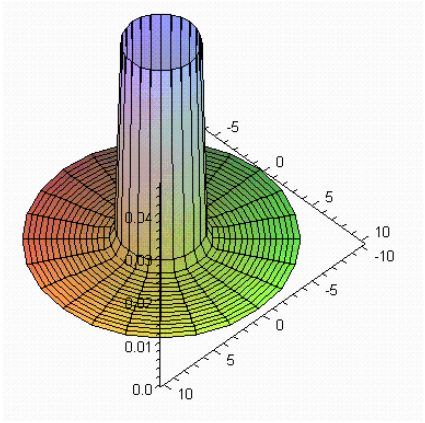
> $TwoDPolarConv\left(\frac{1}{r}\cdot \text{Dirac}(r-1)\cdot \text{Dirac}(\theta-\pi),\text{Dirac}(r-2),r,\theta\right)$

$$\text{Dirac}(r-3)$$

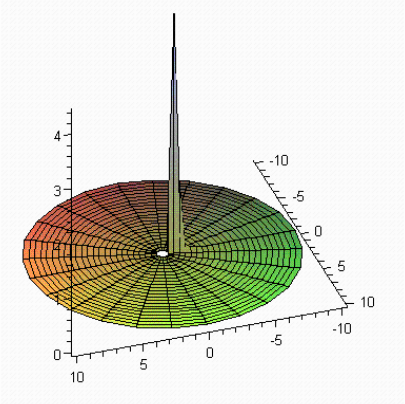
> $Conv\left(\frac{1}{r}\cdot \text{Dirac}(r-1)\cdot \text{Dirac}(\theta-\pi),\text{Dirac}(r-2),[r,\theta],2dpolar\right)$

$$\text{Dirac}(r-3)$$

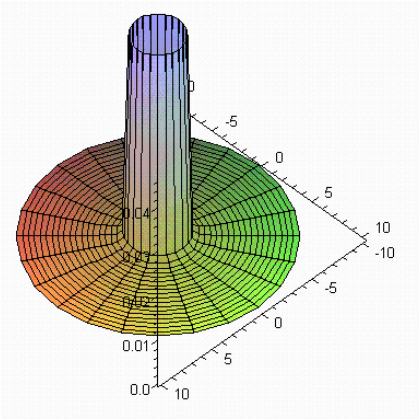
> $plot3d\left(DiracApprox\,(r-3,0.001),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical\,,axes=framed\right)$



> $plot3d\left(\frac{1}{r}\cdot DiracApprox\,(r-1,0.001)\cdot DiracApprox\,(\theta-\pi,0.001),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical\,,axes=framed\right)$



> $plot3d\left(DiracApprox\left(r-2,0.001\right),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical,axes=framed\right)$



>

> $TwoDPolarConv\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1),1,r,\theta\right)$

$$2\,\pi$$

> $TwoDPolarConv\left(1,\frac{1}{r}\cdot\mathrm{Dirac}(r-1),r,\theta\right)$

$$2\,\pi\,\mathrm{Heaviside}(r-1)\,(r-1)$$

> $convert\left(\textbf{(4.1.3.9)},piecewise,r\right)$

$$\begin{cases} 0 & r<1 \\ 2\,\pi\,(r-1) & 1\le r \end{cases}$$

> $\#Conv\left(\exp(r),\frac{1}{r}\cdot\mathrm{Dirac}(r-1),[r,\theta],2\,dpolar\right)$

> $\#convert\left(\textbf{??},piecewise,r\right)$

> $\#TwoDPolarConv\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1),\exp(r),r,\theta\right)$

> $\#plot3d\left(\exp(r),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical,axes=framed\right)$

> $\#plot3d\left(2\ \pi\ e^{r-1},r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical,axes=framed\right)$

> $\#plot3d\left(2\ \pi\ \mathrm{Heaviside}(r-1)\ e^{r-1}\ (r-1),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical,axes=framed\right)$

> $Conv\left(\exp\left(r\cdot\cos(\theta-\pi)\right),\frac{1}{r}\cdot\mathrm{Dirac}(r-1)\cdot\mathrm{Dirac}(\theta-\pi),[r,\theta],2\,dpolar\right)$

$$\text{Heaviside}(\theta-\pi)\ \text{Heaviside}(r-1)\ (r-1)\ e^{(r-1)\cos(\theta)}$$

> $convert\left(\ \textbf{(4.1.3.11)}\ , piecewise, r\right)$

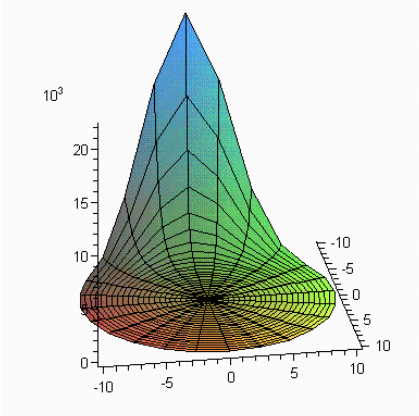
$$\begin{cases} 0 & r<1 \\ \text{Heaviside}(\theta-\pi)\ (r-1)\ e^{(r-1)\cos(\theta)} & 1\le r \end{cases}$$

> $TwoDPolarConv\left(\frac{1}{r}\cdot \text{Dirac}(r-1)\cdot \text{Dirac}(\theta-\pi), \exp(r\cdot\cos(\theta-\pi)), r, \theta\right)$

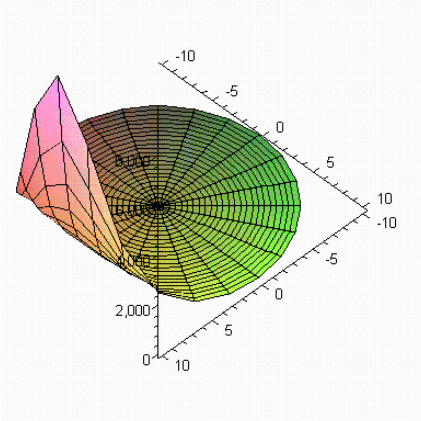
$$e^{(r-1)\cos(\theta)}$$

>

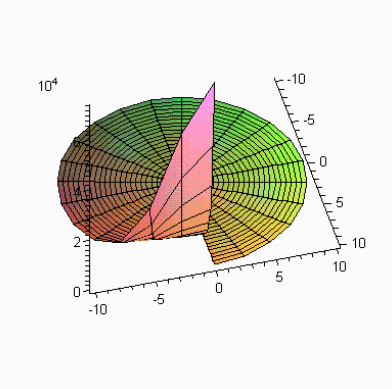
> $plot3d\left(\exp(r\cdot\cos(\theta-\pi)), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(e^{(r-1)\cdot\cos(\theta)}, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(\text{Heaviside}(\theta-\pi)\cdot\text{Heaviside}(r-1)\cdot(r-1)\cdot e^{(r-1)\cdot\cos(\theta)}, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $TwoDPolarConv\left(r, \frac{1}{r}\cdot \mathrm{Dirac}(r-1), r, \theta\right)$

$$2\,\pi\,\mathrm{Heaviside}(r-1)\,\left(r^2-2\,r+1\right)$$

> $convert\left(\textbf{(4.1.3.14)}, piecewise, r\right)$

$$\begin{cases} 0 & r<1 \\ 2\,\pi\,\left(r^2-2\,r+1\right) & 1\le r \end{cases}$$

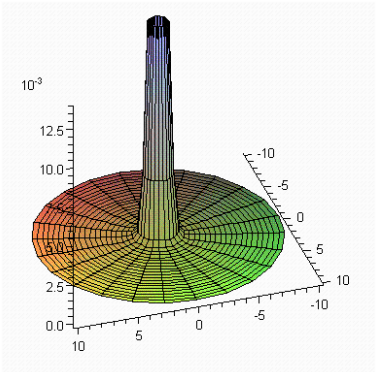
> $TwoDPolarConv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1), r, r, \theta\right)$

$$2\,\pi\,(r-1)$$

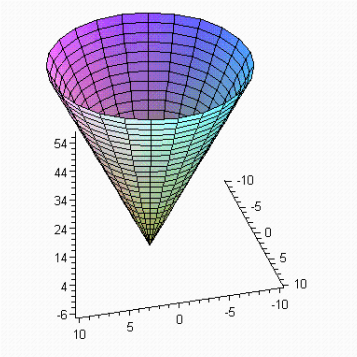
> $Conv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1), r, [r,\theta], 2\,dpolar\right)$

$$2\,\pi\,(r-1)$$

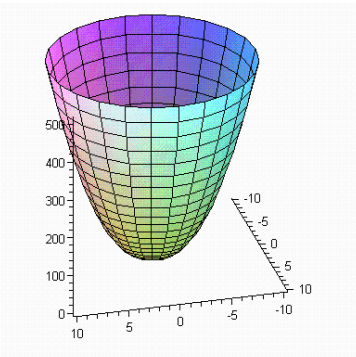
> $plot3d\left(\frac{1}{r}\cdot DiracApprox\,(r-1, 0.001), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(2\cdot\pi\cdot(r-1), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(2\cdot\pi\cdot\text{Heaviside}(r-1)\cdot\left(r^2-2\,r+1\right), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $TwoDPolarConv\left(\frac{1}{r}\cdot\text{Dirac}(r-1)\cdot\text{Dirac}(\theta-\pi), r, r, \theta\right)$

$$r-1$$

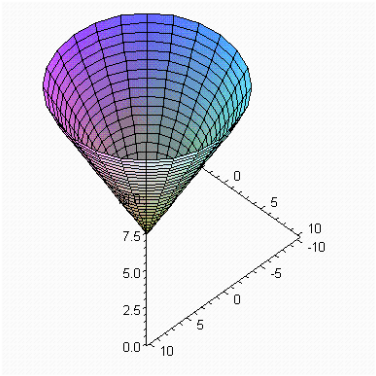
> $TwoDPolarConv\left(r, \frac{1}{r}\cdot\text{Dirac}(r-1)\cdot\text{Dirac}(\theta-\pi), r, \theta\right)$

$$\text{Heaviside}(\theta-\pi)\ \text{Heaviside}(r-1)\ \left(r^2-2\,r+1\right)$$

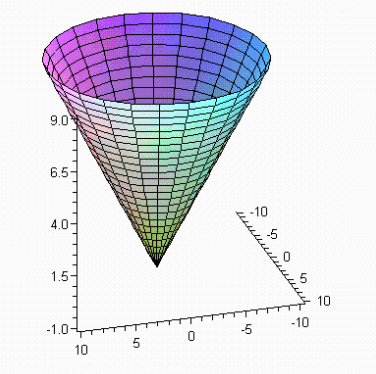
> $convert\left(\textbf{(4.1.3.19)}, piecewise, r\right)$

$$\begin{cases} 0 & r<1 \\ \text{Heaviside}(\theta-\pi)\ \left(r^2-2\,r+1\right) & 1\le r \end{cases}$$

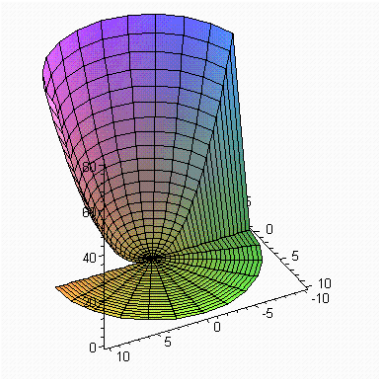
> $plot3d\left(r, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(r-1, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(\text{Heaviside}(\theta-\pi)\cdot\text{Heaviside}(r-1)\cdot\left(r^2-2\,r+1\right), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



>

>

Examples for Angular Convolution

> $Conv(\sin(\theta), \cos(\theta), [\theta], angular)$

$$\frac{1}{2}\sin(\theta)$$

> $AngConv(\cos(\theta), \sin(\theta), \theta)$

$$\frac{1}{2}\sin(\theta)$$

> $\#Conv(\theta, 1, [\theta], angular)$

> $\#Conv(1, \theta, [\theta], angular)$

>

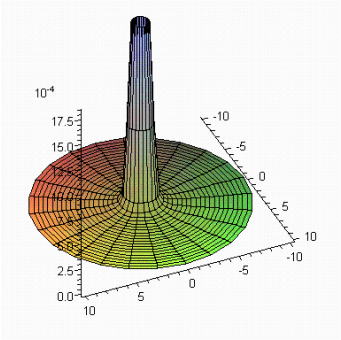
> $AngConv\left(\frac{1}{r}\cdot\text{Dirac}(r)\cdot\text{Dirac}(\theta-\pi), r\cdot\sin(\theta), \theta\right)$

$$-\frac{1}{2}\,\frac{\text{Dirac}(r)\,\sin(\theta)}{\pi}$$

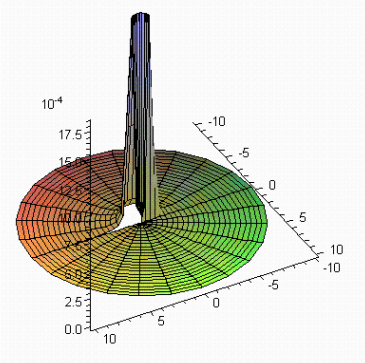
> $AngConv\left(r\cdot\sin(\theta), \frac{1}{r}\cdot\text{Dirac}(r)\cdot\text{Dirac}(\theta-\pi), \theta\right)$

$$-\frac{1}{2}\,\frac{\text{Dirac}(r)\,\text{Heaviside}(\theta-\pi)\,\sin(\theta)}{\pi}$$

> $plot3d\left(\frac{1}{2}\cdot\frac{DiracApprox\,(r-1,0.001)}{\pi}, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(\frac{1}{2}\cdot\frac{\text{Heaviside}(\theta-\pi)\cdot DiracApprox\,(r-1,0.001)}{\pi}, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



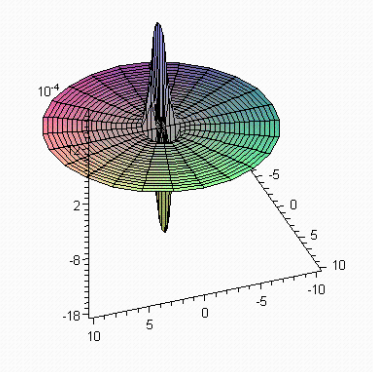
> $Conv\left(\frac{1}{r}\cdot\text{Dirac}(r-1)\cdot\text{Dirac}(\theta-\pi), \sin(\theta), [\theta], angular\right)$

$$-\frac{1}{2}\,\frac{\sin(\theta)\ \text{Dirac}(r-1)}{\pi}$$

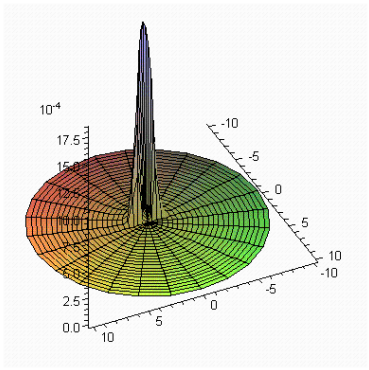
> $Conv\left(\sin(\theta), \frac{1}{r}\cdot\text{Dirac}(r-1)\cdot\text{Dirac}(\theta-\pi), [\theta], angular\right)$

$$-\frac{1}{2}\,\frac{\text{Heaviside}(\theta-\pi)\ \sin(\theta)\ \text{Dirac}(r-1)}{\pi}$$

> $plot3d\left(-\frac{1}{2}\cdot\frac{\sin(\theta)\cdot DiracApprox\,(r-1,0.001)}{\pi}, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(-\frac{1}{2}\cdot\frac{\text{Heaviside}(\theta-\pi)\cdot\sin(\theta)\cdot DiracApprox\,(r-1,0.001)}{\pi}\,, r=0..10, \theta=0..2\cdot\pi, \right.$
> $\left. coords=z\_cylindrical, axes=framed \right)$



> $simplify\left(FS1D(1,\theta,n,0..2\cdot\pi, coefficientComplex)\cdot FS1D\left(\text{Dirac}\left(\theta-\frac{\pi}{2}\right),\theta,n,0..2\cdot\pi, \right.\right.$
> $\left.\left. coefficientComplex \right)\right)$

$$\begin{cases} \frac{1}{2\,\pi} & n=0 \\ 0 & otherwise \end{cases}$$

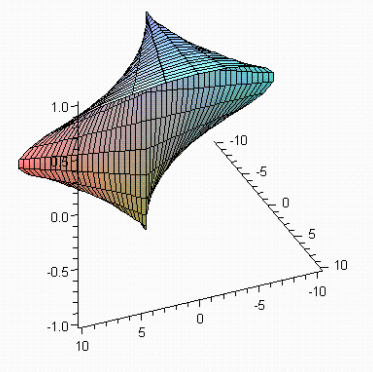
> $FS1D\left(Conv\left(\text{Dirac}\left(\theta-\frac{\pi}{2}\right),1,[\theta],angular\right),\theta,n,0..2\cdot\pi, coefficientComplex \right)$

$$\begin{cases} \frac{1}{2\,\pi} & n=0 \\ 0 & otherwise \end{cases}$$

> $\#simplify(FS1D(\sin(\theta),\theta,n,0..2\cdot\pi, coefficientComplex)\cdot FS1D(\text{Dirac}(\theta-\pi),\theta,n,0..2\cdot\pi,$
> $coefficientComplex))$

> $\#FS1D(Conv(\text{Dirac}(\theta-\pi),\sin(\theta),[\theta],angular),\theta,n,0..2\cdot\pi, coefficientComplex)$

> $plot3d(\sin(\theta), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed)$



>

Examples for Radial Convolution

> $Conv(\text{Dirac}(r-2), r, [r], radial)$

$$2r-4$$

> $RadConv(\text{Dirac}(r-2), r, r)$

$$2r-4$$

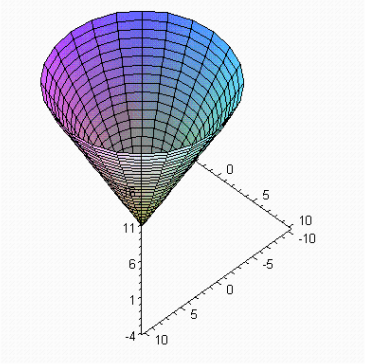
> $RadConv(r, \text{Dirac}(r-2), r)$

$$\text{Heaviside}(r-2)\left(r^2-4r+4\right)$$

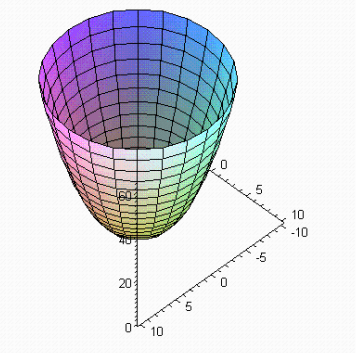
> $\#Conv\left(r^2, \text{Dirac}(r), [r], radial\right)$

> $\#Conv(a, \text{Dirac}(r), [r], radial)$

> $plot3d\left(2\cdot r-4, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



> $plot3d\left(\text{Heaviside}(r-2)\cdot(r-2)^2, r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, axes=framed\right)$



Examples for *Series Convolution*

> $Conv\left(2, \frac{1}{n}, [n], series\right)$

$$\sum_{n=-\infty}^{\infty} \frac{2}{n}$$

>

> $SerConv\left(2, \frac{1}{n}, n\right)$

$$\sum_{n=-\infty}^{\infty} \frac{2}{n}$$

> $\#Conv\left(n^2, n!, [n], series\right)$

> $\#Conv\left(a, \frac{1}{n}, [n], series\right)$

>

Examples - looking into Hankel table

(using our created procedure)

> $takeAlook\ (2, HankelTable\ )$

$$\frac{2\ \mathrm{Dirac}(\rho)}{\rho}$$

> $takeAlook\left(\frac{A}{r}, HankelTable\right)$

$$\frac{A}{\rho}$$

> $takeAlook\left(\ r^{-1.5}, HankelTable0\ \right)$

$$\frac{2.092099241}{\rho^{0.5}}$$

> $takeAlook\left(\frac{A}{\sqrt{r^2+2^2}}, HankelTable0\right)$

$$\frac{A\ e^{-\rho\sqrt{4}}}{\rho}$$

> $takeAlook\left(\frac{1}{r^2+2^2}, HankelTable0\right)$

$$\mathrm{BesselK}\left(0., \rho\sqrt{4}\right)$$

> $takeAlook\left(e^{-\frac{2^2 \cdot r^2}{2}}, HankelTable0\right)$

$$\frac{1}{4}\ e^{-\frac{1}{8}\rho^2}$$

> $takeAlook\ (r^{0-1} \cdot e^{-2 \cdot r}, HankelTable\ )$

$$\frac{\left(\sqrt{\rho^2+4}-2\right)^{\nu}}{\rho^{\nu}\sqrt{\rho^2+4}}$$

> *takeAlook* $\left(r^{-1}\cdot e^{-2\cdot r}, \mathit{HankelTable}\right)$

$$\frac{\left(\sqrt{\rho^2+4}-2\right)^{\nu}}{\rho^{\nu}\sqrt{\rho^2+4}}$$

>

Examples for Hankel procedure

(Entering a set does not keep the result in order... need to use list instead!!)

> *Hankel* $\left(\left[2, \frac{2}{r}, r^{-1.5}\right], r, s, 0\right)$

$$\left[\frac{2\ \mathrm{Dirac}(s)}{s}, \frac{2}{s}, \frac{2.092099241}{\sqrt{s}}\right]$$

> *Hankel* $\left(r^{2-1}, r, s, \frac{1}{2}\right)$

```
Error, (in Hankel) numeric exception: division by zero
```

> *Hankel* $\left(\left[\frac{1}{\sqrt{r^2+2^2}}, \frac{1}{r^2+1^2}\right], r, s, 0\right)$

$$\left[\frac{e^{-s\sqrt{4}}}{s}, \mathrm{BesselK}(0., s)\right]$$

> *Hankel* $\left(\left[e^{-\frac{1^2\cdot r^2}{2}}, r^{-1}\cdot e^{-2\cdot r}, r^{0-1}\cdot e^{-3\cdot r}, e^{0-1\cdot r^2}\cdot \mathrm{BesselJ}(0, 2\cdot r)\right], r, s, 0\right)$

$$\left[e^{-\frac{1}{2}s^2}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{4+s^2}}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{9+s^2}}, \int_0^{\infty} e^{-r^2}\ \mathrm{BesselJ}(0, 2\,r)\ \mathrm{BesselJ}(0, s\,r)\ r\,dr\right]$$

> *Hankel* $\left(r^{-2.3}, r, s, 2\right)$

$$0.4210717840 s^{3/10}$$

> *Hankel* $\left(\frac{1}{2}\cdot\frac{\mathrm{Dirac}(r)}{r\cdot\pi}, r, s, n\right)$

$$\frac{1}{2}\ \frac{\mathrm{BesselJ}(n, 0)}{\pi}$$

> *Hankel* $(\mathrm{Dirac}(r-3), r, s, 0)$

$$3\ \mathrm{BesselJ}(0, 3\,s)$$

Examples for InvHankel procedure

> $InvHankel\left(\left[2, \frac{2}{s}, s^{-1.5}\right], s, r, 0\right)$

$$\left[\frac{2\,\mathrm{Dirac}(r)}{r}, \frac{2}{r}, \frac{2.092099241}{\sqrt{r}}\right]$$

> $InvHankel\left(s^2 - 1, s, r, \frac{1}{2}\right)$

```
Error, (in IntegralTrans[Hankel]) numeric exception:
division by zero
```

> $InvHankel\left(\left[\frac{1}{\sqrt{s^2 + 2^2}}, \frac{1}{s^2 + 1^2}\right], s, r, 0\right)$

$$\left[\frac{e^{-r\sqrt{4}}}{r}, \mathrm{BesselK}(0., r)\right]$$

> $InvHankel\left(\left[e^{-\frac{1^2 \cdot s^2}{2}}, s^{-1} \cdot e^{-2 \cdot s}, s^{0-1} \cdot e^{-3 \cdot s}\right], s, r, 0\right)$

$$\left[e^{-\frac{1}{2}r^2}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{r^2+4}}, \frac{\left(-\frac{1}{2}\right)!}{\sqrt{\pi}\sqrt{9+r^2}}\right]$$

>

Examples for FS procedure

> $FS1D(1, x, n, 0..2 \cdot \pi, coefficientComplex)$

$$\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(\frac{2}{x}, x, n, -\frac{L}{2}..\frac{L}{2}, coefficientReal\right) \text{ assuming } L > 0$

$$\frac{undefined}{L}, \frac{8\,\mathrm{Si}(\pi n)}{L}$$

> $FS1D(x^2, x, n, -2..2, coefficientComplex)$

$$\begin{cases} \frac{4}{3} & n = 0 \\ -\frac{2\left(-2\,I\,e^{2\,I\,\pi\,n} - 2\,e^{2\,I\,\pi\,n}\,\pi\,n + I\,e^{2\,I\,\pi\,n}\,\pi^2\,n^2 + 2\,I - 2\,\pi\,n - I\,\pi^2\,n^2\right)e^{-I\,\pi\,n}}{\pi^3\,n^3} & otherwise \end{cases}$$

> $FS1D\left(\frac{x^2}{4}, x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{1}{3} & n = 0 \\ -\frac{1}{2}\,\frac{1}{\pi^3\,n^3}\left(\left(-2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} - 2\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi\,n + \mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi^2\,n^2 + 2\,\mathrm{I} - 2\,\pi\,n\right.\right. & \text{otherwise} \\ \left.\left. - \mathrm{I}\,\pi^2\,n^2\right)\,e^{-\mathrm{I}\,\pi\,n}\right) \end{cases}$$

> $FS1D\left('Scale'\left(x^2, x, \frac{1}{2}\right), x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{1}{3} & n = 0 \\ -\frac{1}{2}\,\frac{1}{\pi^3\,n^3}\left(\left(-2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} - 2\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi\,n + \mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi^2\,n^2 + 2\,\mathrm{I} - 2\,\pi\,n\right.\right. & \text{otherwise} \\ \left.\left. - \mathrm{I}\,\pi^2\,n^2\right)\,e^{-\mathrm{I}\,\pi\,n}\right) \end{cases}$$

> $FS1D\left(x^2 + 2\cdot x, x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{4}{3} & n = 0 \\ \frac{2\left(e^{2\,\mathrm{I}\,\pi\,n}\,\pi\,n + 2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} + 3\,\pi\,n - 2\,\mathrm{I} + 2\,\mathrm{I}\,\pi^2\,n^2\right)\,e^{-\mathrm{I}\,\pi\,n}}{\pi^3\,n^3} & \text{otherwise} \end{cases}$$

> $FS1D\left(\frac{x^2}{4} + x, x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{1}{3} & n = 0 \\ \frac{1}{2}\,\frac{\left(2\,\mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n} + \mathrm{I}\,e^{2\,\mathrm{I}\,\pi\,n}\,\pi^2\,n^2 + 4\,\pi\,n - 2\,\mathrm{I} + 3\,\mathrm{I}\,\pi^2\,n^2\right)\,e^{-\mathrm{I}\,\pi\,n}}{\pi^3\,n^3} & \text{otherwise} \end{cases}$$

> $FS1D\left(x^{2-1}, x, n, 0..2\cdot\pi, series\right)$

$$\pi + \mathrm{I}\,\ln\left(1 - e^{-\mathrm{I}\,x}\right) - \mathrm{I}\,\ln\left(1 - e^{\mathrm{I}\,x}\right)$$

> $FS1D\left(\sin(x), x, n, 0..\pi, series\right)$

$$\sum_{n=-\infty}^{\infty}\left(-\frac{2\,e^{\mathrm{I}\,n\,x}}{\pi\left(4\,n^2 - 1\right)}\right)$$

> $FS1D\left(\sin\left(\frac{x}{2}\right), x, n, 0..\pi, series\right)$

$$\sum_{n=-\infty}^{\infty}\frac{2\left(-1 + 4\,\mathrm{I}\,n\right)\,e^{\mathrm{I}\,n\,x}}{\pi\left(16\,n^2 - 1\right)}$$

> $FS1D\left('Scale'\left(\sin(x), x, \frac{1}{2}\right), x, n, 0..\pi, series\right)$

$$\sum_{n=-\infty}^{\infty}\frac{2\left(-1 + 4\,\mathrm{I}\,n\right)\,e^{\mathrm{I}\,n\,x}}{\pi\left(16\,n^2 - 1\right)}$$

> $FS1D\left(e^{-\frac{1^2\cdot x^2}{2}}, x, n, -\pi..\pi, series\right)$

$$\sum_{n=-\infty}^{\infty} \frac{1}{4} \frac{e^{\frac{1}{2} n (-n+2 I x)} \sqrt{2} \left(\operatorname{erf}\left(\frac{1}{2} \pi \sqrt{2} - \frac{1}{2} I n \sqrt{2}\right) + \operatorname{erf}\left(\frac{1}{2} \pi \sqrt{2} + \frac{1}{2} I n \sqrt{2}\right)\right)}{\sqrt{\pi}}$$

> $FS1D\left(x^{-1} \cdot e^{-2 \cdot x}, x, n, -\pi..\pi, sequence\right)$

```
Error, (in FS1D) Procedure must have 4 or 5 arguments.
Fifth argument can only be 'coefficientReal',
'coefficientComplex' or 'series'
```

> $FS1D\left(e^{0-1 \cdot x^2} \cdot \text{BesselJ}(0, 2 \cdot x), x, n, -\pi..\pi\right)$

$$\frac{1}{2} \frac{\int_{-\pi}^{\pi} e^{-x^2} \text{BesselJ}(0, 2 x)\, e^{-I n x}\, dx}{\pi}$$

> $\#trace(FS1D)$

> $FS1D\left(2 \cdot Conv\left(\text{Dirac}(\theta - \pi), 1, [\theta], angular\right), \theta, n, 0..2 \cdot \pi, coefficientComplex\right)$

$$\begin{cases} \frac{1}{\pi} & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(\text{Dirac}(\theta - \pi), \theta, n, 0..2 \cdot \pi, coefficientComplex\right)$

$$\frac{1}{2} \frac{e^{-I \pi n}}{\pi}$$

> $FS1D\left(2 \cdot 'Conv'\left(1, \text{Dirac}(\theta - \pi), [\theta], angular\right), \theta, n, 0..2 \cdot \pi, coefficientComplex\right)$

$$\begin{cases} \frac{1}{\pi} & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(2 \cdot \exp(I \cdot 3 \cdot \theta), \theta, n, 0..2 \cdot \pi\right)$

$$\begin{cases} 2 & n = 3 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(x^2, x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{4}{3} & n = 0 \\ -\frac{2\left(-2 I e^{2 I \pi n} - 2 e^{2 I \pi n} \pi n + I e^{2 I \pi n} \pi^2 n^2 + 2 I - 2 \pi n - I \pi^2 n^2\right) e^{-I \pi n}}{\pi^3 n^3} & otherwise \end{cases}$$

> $FS1D\left(x^2 \cdot \exp(I \cdot 3 \cdot x), x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} \frac{4}{3} & n = 3 \\ -\frac{1}{\pi^3 (n-3)^3}\left(2\left(-2 I e^{2 I \pi (n-3)} - 2 e^{2 I \pi (n-3)} \pi (n-3)\right.\right. & otherwise \\ \left.\left. + I e^{2 I \pi (n-3)} \pi^2 (n-3)^2 + 2 I - 2 \pi (n-3) - I \pi^2 (n-3)^2\right) e^{-I \pi (n-3)}\right) & \end{cases}$$

> $FS1D\left(\exp(I \cdot 2 \cdot x), x, n, -2..2, coefficientComplex\right)$

$$\begin{cases} 1 & n = 2 \\ 0 & otherwise \end{cases}$$

> $FS1D(\exp(I\cdot 2\cdot x)\cdot\exp(I\cdot 3\cdot x), x, n, -2..2, coefficientComplex)$

$$\begin{cases} 1 & n = 5 \\ 0 & otherwise \end{cases}$$

> $FS1D(\exp(I\cdot(2+3)\cdot x), x, n, -2..2, coefficientComplex)$

$$\begin{cases} 1 & n = 5 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(\frac{1}{2}\cdot \mathrm{Dirac}(r), x, n, 0..2\cdot\pi\right)$

$$\begin{cases} \frac{1}{2}\,\mathrm{Dirac}(r) & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(\frac{1}{r}\cdot \mathrm{Dirac}(r), x, n, 0..2\cdot\pi\right)$

$$\begin{cases} \frac{\mathrm{Dirac}(r)}{r} & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(\frac{1}{3\cdot r}\cdot \mathrm{Dirac}(r-r0), x, n, 0..2\cdot\pi\right)$

$$\begin{cases} \frac{1}{3}\,\frac{\mathrm{Dirac}(r-r0)}{r} & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-r0)\cdot\mathrm{Dirac}(x), x, n, 0..2\cdot\pi\right)$

$$\frac{1}{2}\,\frac{\mathrm{Dirac}(r-r0)}{r\,\pi}$$

> $FS1D(\mathrm{Dirac}(x), x, n, -2..2, coefficientComplex)$

$$\frac{1}{4}$$

> $FS1D(\mathrm{Dirac}(x-1), x, n, -2..2, coefficientComplex)$

$$\frac{1}{4}\,e^{-I\,n}$$

> $FS1D('Shift'(\mathrm{Dirac}(x), x, 1), x, n, -2..2, coefficientComplex)$

$$\frac{1}{4}\,e^{-I\,n}$$

> $FS1D(\mathrm{Dirac}(x-1)\cdot\exp(I\cdot 5\cdot x), x, n, -2..2, coefficientComplex)$

$$\frac{1}{4}\,e^{-I\,(n-5)}$$

> $FS1D(Conv(\mathrm{Dirac}(r-1), \mathrm{Dirac}(r-2), [r, y], '2dpolar'), y, n, 0..2\cdot\pi)$

$$\begin{cases} 2\,\pi\,\mathrm{Dirac}(r-3) & n = 0 \\ 0 & otherwise \end{cases}$$

> $FS1D(Conv(\mathrm{Dirac}(r-2), \mathrm{Dirac}(r-1), [r, y], '2dpolar'), y, n, 0..2\cdot\pi)$

$$\begin{cases} 4\,\pi\,\mathrm{Dirac}(r-3) & n=0 \\ 0 & otherwise \end{cases}$$

> $\#trace(FS1D)$

>

Examples for Inverse FS procedure

> $InvFS1D\left(-\frac{i\cdot\left(e^{2\cdot i\cdot\pi\cdot n}-1\right)\cdot e^{-i\cdot\pi\cdot n}}{\pi\cdot n}, n, x, 0..\infty\right)$

$$2$$

> $InvFS1D\left(-I\cdot\frac{1}{2\cdot n}, n, x\right)$

$$\begin{cases} -\frac{1}{2}\,\pi-\frac{1}{2}\,x & -\pi\le x \text{ and } x<0 \\ \frac{1}{2}\,\pi-\frac{1}{2}\,x & 0\le x \text{ and } x<\pi \end{cases}$$

> $InvFS1D\left((-1)^n\cdot\frac{1}{2\cdot n}, n, x\right)$

$$\begin{cases} -\ln\left(2\cos\left(\frac{1}{2}\,x\right)\right) & -\pi<x \text{ and } x<\pi \\ 0 & otherwise \end{cases}$$

> $InvFS1D\left(\frac{1}{(4\cdot n+2)\cdot I}, n, x\right)$

$$\begin{cases} -\frac{1}{4}\,\pi & -\pi<x \text{ and } x<0 \\ \frac{1}{4}\,\pi & 0<x \text{ and } x<\pi \end{cases}$$

> $InvFS1D\left(\frac{2\,\pi\,I^{-n}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{Dirac}(\rho)}{\rho}, n, \psi\right)$

$$\frac{2\,\pi\,\mathrm{Dirac}(\rho)}{\rho}$$

> $InvFS1D\left(\frac{1}{2}\cdot\text{'}Conv\text{'}(piecewise(n=0,3), piecewise(n=3,2), [n], series), n, x, 0..2\cdot\pi\right)$

$$3\,e^{3\,I\,x}$$

> $InvFS1D(piecewise(n=3,2), n, x)$

$$2\,e^{3\,I\,x}$$

> $InvFS1D(piecewise(n=0,1), n, x)$

$$1$$

>

Examples for 2D Polar FT procedure

> $Polar2DFT(1, r, \theta, \rho, \psi)$

$$\sum_{n=-\infty}^{\infty} \frac{2\,\pi\, e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \mathrm{Dirac}(\rho)\, e^{I \psi n}}{\rho}$$

> $convert((4.7.1), 'piecewise', n)$

$$\frac{2\,\pi\,\mathrm{Dirac}(\rho)}{\rho}$$

> $Polar2DFT\left(\frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r), r, \theta, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \mathrm{BesselJ}(n, 0)\, e^{I \psi n}$$

> $convert((4.7.3), 'piecewise', n)$

$$1$$

> $\#trace(Polar2DFT)$

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$1$$

> $Polar2DFT(\exp(2\cdot I\cdot r), r, \theta, \rho, \psi)$

$$\sum_{n=-\infty}^{\infty} 2\,\pi\, e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \left(\int_0^{\infty} e^{2 I r}\, \mathrm{BesselJ}(n, \rho\, r)\, r\, dr \right) e^{I \psi n}$$

> $convert((4.7.6), 'piecewise')$

$$2\,\pi \left(\int_0^{\infty} e^{2 I r}\, \mathrm{BesselJ}(0, \rho\, r)\, r\, dr \right)$$

> $Polar2DFT\left(\frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-3), r, \theta, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} e^{-\frac{1}{2} I n \pi} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \mathrm{BesselJ}(n, 3\,\rho)\, e^{I \psi n}$$

> $convert((4.7.8), 'piecewise', n)$

$$\mathrm{BesselJ}(0, 3\,\rho)$$

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-3)\cdot \mathrm{Dirac}(x-\pi), r, x, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} e^{-\frac{3}{2} I n \pi}\, \mathrm{BesselJ}(n, 3\,\rho)\, e^{I \psi n}$$

> $Polar2DFT(sum(I^n\cdot \mathrm{BesselJ}(n, 2\cdot r)\cdot \exp(-I\cdot n\cdot \pi)\cdot \exp(I\cdot n\cdot \theta), n = -\infty .. \infty), r, \theta, \rho, \psi)$

$$\frac{4\,\pi^2\,\mathrm{Dirac}(\rho-2)\,\mathrm{Dirac}(\psi-\pi)}{\rho}$$

> $Polar2DFT\left(\frac{1}{3}\cdot 'Conv'\left(1, \frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-2), [r, x], '2dpolar'\right), r, x, \rho, \psi\right)$

$$\frac{1}{3}\left(\sum_{n=-\infty}^{\infty}\frac{2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{Dirac}(\rho)\,e^{I\,\psi\,n}}{\rho}\right)\left(\sum_{n=-\infty}^{\infty}e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n,2\,\rho)\,e^{I\,\psi\,n}\right)$$

> *convert* (**(4.7.12)** , '*piecewise*' , *n*)

$$\frac{2}{3}\,\frac{\pi\,\mathrm{Dirac}(\rho)\,\mathrm{BesselJ}(0,2\,\rho)}{\rho}$$

> *Polar2DFT* $\left(2\cdot\text{'}Conv\text{'}\left(1,\mathrm{Dirac}\left(x-\frac{\pi}{2}\right),[x],\text{'}angular\text{'}\right),r,x,\rho,\psi\right)$

$$\sum_{n=-\infty}^{\infty}\frac{2\left(e^{-\frac{1}{2}\,I\,n\,\pi}\right)^{2}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{Dirac}(\rho)\,e^{I\,\psi\,n}}{\rho}$$

> *convert* (**(4.7.14)** , *piecewise* , *n*)

$$\frac{2\,\mathrm{Dirac}(\rho)}{\rho}$$

> *#Polar2DFT* $\left(\mathrm{Dirac}\left(x-\frac{\pi}{4}\right),r,x,\rho,\psi\right)$

> *Polar2DFT* $\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-3),r,\theta,\rho,\psi\right)$

$$\sum_{n=-\infty}^{\infty}2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n,3\,\rho)\,e^{I\,\psi\,n}$$

> *convert* (**(4.7.16)** , '*piecewise*' , *n*)

$$2\,\pi\,\mathrm{BesselJ}(0,3\,\rho)$$

> *FSH* $\left(\text{'}Conv\text{'}\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1),r,[r,\theta],2\ dpolar\right),r,\theta,\rho,\psi\right)$

$$2\,\pi\,I^{-n}\left(\int_{0}^{\infty}r^{2}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n,\rho\,r)\,dr\right)$$

> *convert* (**(4.7.18)** , '*piecewise*' , *n*)

$$2\,\pi\,I^{-n}\left(\begin{cases}int\left(r^{2}\,\mathrm{BesselJ}(0,\rho\,r),r=0\,..\,\infty,opts\right) & n=0\\ 0 & otherwise\end{cases}\right)$$

> *Polar2DFT* $\left(Conv\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1),r,[r,\theta],2\ dpolar\right),r,\theta,\rho,\psi\right)$

$$\sum_{n=-\infty}^{\infty}4\,\pi^{2}\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}1 & n=0\\ 0 & otherwise\end{cases}\right)\left(\int_{0}^{\infty}\left(\mathrm{BesselJ}(n,\rho\,r)\,r^{2}-\mathrm{BesselJ}(n,\rho\,r)\,r\right)dr\right)e^{I\,\psi\,n}$$

> *convert* (**(4.7.20)** , '*piecewise*' , *n*)

$$4\,\pi^{2}\left(\int_{0}^{\infty}\left(\mathrm{BesselJ}(0,\rho\,r)\,r^{2}-\mathrm{BesselJ}(0,\rho\,r)\,r\right)dr\right)$$

> *Polar2DFT* $\left(\text{'}Conv\text{'}\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1),r,[r,\theta],2\ dpolar\right),r,\theta,\rho,\psi\right)$

$$\left(\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}\right)\left(\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(n,\rho\,r)\,dr\right)e^{I\,\psi\,n}\right)$$

> *convert* (**(4.7.22)** , '*piecewise*', *n*)

$$4\,\pi^2\,\mathrm{BesselJ}(0,\rho)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0,\rho\,r)\,dr\right)$$

> *Polar2DFT*$(r, r, \theta, \rho, \psi)$

$$\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(n,\rho\,r)\,dr\right)e^{I\,\psi\,n}$$

> *convert* (**(4.7.24)** , '*piecewise*', *n*)

$$2\,\pi\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0,\rho\,r)\,dr\right)$$

> *Polar2DFT*$\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1), r, \theta, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}$$

> *convert* (**(4.7.26)** , '*piecewise*', *n*)

$$2\,\pi\,\mathrm{BesselJ}(0,\rho)$$

> *#Polar2DFT*$\left(Conv\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1), \frac{1}{r}\cdot\mathrm{Dirac}(r), [r,\theta], 2\,dpolar\right), r, \theta, \rho, \psi\right)$

> *#convert* (**??** , '*piecewise*', *n*)

> *Polar2DFT*$\left(Conv\left(\frac{1}{r}\cdot\mathrm{Dirac}(r), \frac{1}{r}\cdot\mathrm{Dirac}(r-1), [r,\theta], 2\,dpolar\right), r, \theta, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} 4\,\pi^2\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}$$

> *convert* (**(4.7.28)** , '*piecewise*', *n*)

$$4\,\pi^2\,\mathrm{BesselJ}(0,\rho)$$

> *Polar2DFT*$\left('Conv'\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1), \frac{1}{r}\cdot\mathrm{Dirac}(r), [r,\theta], 2\,dpolar\right), r, \theta, \rho, \psi\right)$

$$2\left(\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}\right)\pi$$

> *convert* (**(4.7.30)** , '*piecewise*', *n*)

$$4\,\pi^2\,\mathrm{BesselJ}(0,\rho)$$

> *Polar2DFT*$\left(\frac{1}{r}\cdot\mathrm{Dirac}(r-1), r, \theta, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n,\rho)\,e^{I\,\psi\,n}$$

> *convert* (**(4.7.32)** , '*piecewise*', *n*)

$$2\,\pi\,\mathrm{BesselJ}(0,\rho)$$

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r), r, \theta, \rho, \psi\right)$

$$2\,\pi$$

> $\#plot3d\left(\frac{1}{r}\cdot DiracApprox\,(r, 0.001), r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, axes = framed\right)$

> $\#plot3d\left(\frac{1}{2\cdot\pi\cdot r}\cdot DiracApprox\,(r-3, 0.001), r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, axes = framed\right)$

> $\#plot3d\left(2\cdot\pi\cdot \mathrm{BesselJ}(0, 3\cdot\rho), \rho = 0..10, \psi = 0..2\cdot\pi, coords = z\_cylindrical, axes = framed\right)$

> $\#Polar2DFT\left(\mathrm{Heaviside}(r-1), r, \theta, \rho, \psi\right)$

> $\#convert\,(\ ??, piecewise\ )$

> $Polar2DFT\left('Conv'\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1)\cdot \mathrm{Dirac}(\theta-\pi), r, [r, \theta], '2dpolar'\right), r, \theta, \rho, \psi\right)$

$$\left(\sum_{n=-\infty}^{\infty} I^{-n}\,\mathrm{BesselJ}(n, \rho)\,e^{-I\,\pi\,n}\,e^{I\,\psi\,n}\right)\left(\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(n, \rho\,r)\,dr\right)e^{I\,\psi\,n}\right)$$

> $convert\,(\ \mathbf{(4.7.35)}, 'piecewise', n)$

$$2\left(\sum_{n=-\infty}^{\infty} I^{-n}\,\mathrm{BesselJ}(n, \rho)\,e^{-I\,\pi\,n}\,e^{I\,\psi\,n}\right)\pi\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0, \rho\,r)\,dr\right)$$

> $Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1)\cdot \mathrm{Dirac}(\theta-\pi), r, \theta, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} I^{-n}\,\mathrm{BesselJ}(n, \rho)\,e^{-I\,\pi\,n}\,e^{I\,\psi\,n}$$

> $Polar2DFT\left('Conv'\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(y), r, [r, y], '2dpolar'\right), r, y, \rho, \psi\right)$

$$\sum_{n=-\infty}^{\infty} 2\,\pi\,e^{-\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(n, \rho\,r)\,dr\right)e^{I\,\psi\,n}$$

> $convert\,(\ \mathbf{(4.7.38)}, 'piecewise', n)$

$$2\,\pi\left(\int_0^{\infty} r^2\,\mathrm{BesselJ}(0, \rho\,r)\,dr\right)$$

> $\#Polar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1)\cdot \mathrm{Dirac}(y-\pi), r, y, \rho, \psi\right)$

> $\#Polar2DFT\left(\mathrm{Dirac}(r-2), r, y, \rho, \psi\right)$

> $\#convert\,(\ ??, 'piecewise', n)$

> $\#FSH\left(\mathrm{Dirac}(r-2), r, y, \rho, \psi\right)$

> $\#convert\,(\ ??, 'piecewise', n)$

> $\#Polar2DFT\left(e^{1\cdot\rho 0\cdot r\cdot\cos(\phi 0-\theta)}, r, \theta, \rho, \phi\right)$ assuming $\rho 0 > 0$

> $\#Polar2DFT\left(Conv\left(\mathrm{Dirac}(r-1), \mathrm{Dirac}(r-2), [r, y], '2\ dcartesian'\right), r, y, \rho, \psi\right)$

> $\#convert\,(\ ??, 'piecewise', n)$

```
> #convert( I^(-n) BesselJ(n, ρ) · { 4 π BesselJ(0, 2 ρ)   n = 0
                                   { 0                     otherwise , 'piecewise', n )
> #convert( Polar2DFT( Dirac(r − 1) · Dirac(y), r, y, ρ, ψ) · FSH( Dirac(r − 2), r, y, ρ, ψ),
      'piecewise', n )
> #convert( FSH( Dirac(r − 1) · Dirac(y), r, y, ρ, ψ) · Polar2DFT( Dirac(r − 2), r, y, ρ, ψ),
      'piecewise', n )
>
```

Examples for 2D Inverse Polar FT procedure

```
> InvPolar2DFT( 1, ρ, ψ, r, θ )
```

$$\sum_{n=-\infty}^{\infty} \frac{1}{2} \frac{e^{\frac{1}{2} I n \pi} \, \mathrm{Dirac}(r) \left\{ \begin{array}{ll} 1 & n = 0 \\ 0 & otherwise \end{array} \right. e^{I \theta n}}{\pi r}$$

```
> convert( (4.8.1), 'piecewise', n )
```

$$\frac{1}{2} \frac{\mathrm{Dirac}(r)}{r \pi}$$

```
> InvPolar2DFT( BesselJ(0, ρ·r0), ρ, ψ, r, θ )
Warning,  computation interrupted
> convert( ??, piecewise, n )
```

$$\frac{\mathrm{BesselJ}(0, r\, r0)\ undefined}{\pi}$$

```
> #InvPolar2DFT( e^(-I·ρ·r0·cos(ψ − θ0)), ρ, ψ, r, θ )
> #InvPolar2DFT( (2·π)^2 · 1/ρ · Dirac(ρ) · Dirac(ψ), ρ, ψ, r, θ )
>
> #InvPolar2DFT( (2·π)^2 · 1/ρ · Dirac(ρ − ρ0) · Dirac(ψ − ψ0), ρ, ψ, r, θ ) assuming ρ0 > 0, ψ0
      :: positive
>
```

Examples for the Direct 2D Polar FT procedure

```
> DirectPolar2DFT( 1, r, θ, ρ, ψ )
```

$$\int_0^{\infty} \int_0^{2\pi} r \, e^{-\rho\, r \cos(\psi - \theta)} \, d\theta \, dr$$

```
> #trace( DirectPolar2DFT )
```

> $DirectPolar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r), r, \theta, \rho, \psi\right)$

$$2\,\pi$$

> $DirectPolar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$1$$

> $DirectPolar2DFT\left(\mathrm{Dirac}(r-3), r, \theta, \rho, \psi\right)$

$$3\left(\int_0^{2\,\pi} e^{-3\,\rho\,(\cos(\psi)\,\cos(\theta)+\sin(\psi)\,\sin(\theta))}\,d\theta\right)$$

> $DirectPolar2DFT\left(\frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-3), r, \theta, \rho, \psi\right)$

$$\frac{1}{2}\,\frac{\int_0^{2\,\pi} e^{-3\,\rho\,(\cos(\psi)\,\cos(\theta)+\sin(\psi)\,\sin(\theta))}\,d\theta}{\pi}$$

> $convert\left(\textbf{(4.9.5)}, 'piecewise', n\right)$

$$\frac{1}{2}\,\frac{\int_0^{2\,\pi} e^{-3\,\rho\,(\cos(\psi)\,\cos(\theta)+\sin(\psi)\,\sin(\theta))}\,d\theta}{\pi}$$

> $DirectPolar2DFT\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-3)\cdot \mathrm{Dirac}(x-\pi), r, x, \rho, \psi\right)$ *#assuming r0 >0, 0 <x0 <2·π*

$$e^{3\,\rho\,\cos(\psi)}$$

> $DirectPolar2DFT\left(\frac{1}{3}\cdot 'Conv'\left(1, \frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-2), [r], '2dpolar'\right), r, x, \rho, \psi\right)$

$$\frac{1}{6}\,\frac{\left(\int_0^{\infty}\int_0^{2\,\pi} r\,e^{-\rho\,r\,\cos(\psi-x)}\,dx\,dr\right)\left(\int_0^{2\,\pi} e^{-2\,\rho\,(\cos(\psi)\,\cos(x)+\sin(\psi)\,\sin(x))}\,dx\right)}{\pi}$$

> $DirectPolar2DFT\left(2\cdot 'Conv'\left(1, \mathrm{Dirac}\left(x-\frac{\pi}{4}\right), [x], 'angular'\right), r, x, \rho, \psi\right)$

$$\int_0^{\infty}\int_{\frac{1}{4}\,\pi}^{2\,\pi} \frac{r\,e^{-\rho\,r\,\cos(\psi-x)}}{\pi}\,dx\,dr$$

> $DirectPolar2DFT\left(\mathrm{Dirac}\left(x-\frac{\pi}{4}\right), r, x, \rho, \psi\right)$

$$\lim_{r \to \infty} \left(-\left(\sin(\psi)^2 e^{-\rho r \sin\left(\psi + \frac{1}{4}\pi\right)} + \sin(\psi)^2 e^{-\rho r \sin\left(\psi + \frac{1}{4}\pi\right)} \rho r \sin\left(\psi + \frac{1}{4}\pi\right) + 2\sin(\psi)\cos(\psi) e^{-\rho r \sin\left(\psi + \frac{1}{4}\pi\right)} + 2\sin(\psi)\cos(\psi) e^{-\rho r \sin\left(\psi + \frac{1}{4}\pi\right)} \rho r \sin\left(\psi + \frac{1}{4}\pi\right) + \cos(\psi)^2 e^{-\rho r \sin\left(\psi + \frac{1}{4}\pi\right)} + \cos(\psi)^2 e^{-\rho r \sin\left(\psi + \frac{1}{4}\pi\right)} \rho r \sin\left(\psi + \frac{1}{4}\pi\right) - 2\sin\left(\psi + \frac{1}{4}\pi\right)^2 \right) \Big/ \left(\rho^2 \left(\sin(\psi)^2 + 2\sin(\psi)\cos(\psi) + \cos(\psi)^2\right) \sin\left(\psi + \frac{1}{4}\pi\right)^2 \right) \right)$$

>

Examples for 2D Polar FSH procedure

> $FSH(1, r, \theta, \rho, \psi)$

$$\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases}$$

$$\frac{\left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \mathrm{Dirac}(\rho)}{\rho}$$

$$\frac{2\pi I^{-n} \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right) \mathrm{Dirac}(\rho)}{\rho}$$

> $convert\left(\mathbf{(4.10.1)}, 'piecewise', n \right)$

$$\begin{cases} \frac{2\pi \mathrm{Dirac}(\rho)}{\rho} & n = 0 \\ 0 & otherwise \end{cases}$$

> $FSH\left(\frac{1}{r} \cdot \mathrm{Dirac}(r), r, \theta, \rho, \psi \right)$

$$\frac{\mathrm{Dirac}(r) \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right)}{r}$$

$$\mathrm{BesselJ}(n, 0) \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right)$$

$$2\pi I^{-n} \mathrm{BesselJ}(n, 0) \left(\begin{cases} 1 & n = 0 \\ 0 & otherwise \end{cases} \right)$$

> $convert\left(\mathbf{(4.10.3)}, 'piecewise', n \right)$

$$\begin{cases} 2\pi & n = 0 \\ 0 & otherwise \end{cases}$$

> $FSH\left(\frac{1}{r}\cdot \mathrm{Dirac}(r)\cdot \mathrm{Dirac}(\theta), r, \theta, \rho, \psi\right)$

$$\frac{1}{2}\ \frac{\mathrm{Dirac}(r)}{r\,\pi}$$

$$\frac{1}{4}\ \frac{\mathrm{BesselJ}(n, 0)}{\pi}$$

$$\frac{1}{2}\ I^{-n}\ \mathrm{BesselJ}(n, 0)$$

> $\#FSH\left(\exp\left(2\cdot I\cdot r\cdot \cos(\theta)\right), r, \theta, \rho, \psi\right)$

> $FSH\left(\frac{1}{2\cdot \pi\cdot r}\cdot \mathrm{Dirac}(r-3), r, \theta, \rho, \psi\right)$

$$\frac{1}{2}\ \frac{\mathrm{Dirac}(r-3)\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)}{r\,\pi}$$

$$\frac{1}{2}\ \frac{\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n, 3\,\rho)}{\pi}$$

$$I^{-n}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n, 3\,\rho)$$

> $convert\left(\ \mathbf{(4.10.6)}, 'piecewise', n\right)$

$$\begin{cases} \mathrm{BesselJ}(0, 3\,\rho) & n=0 \\ 0 & otherwise \end{cases}$$

> $FSH\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-3)\cdot \mathrm{Dirac}(x-1), r, x, \rho, \psi\right)$ assuming $r0 > 0, 0 < x0 < 2\cdot\pi$

$$\frac{1}{2}\ \frac{\mathrm{Dirac}(r-3)\ e^{-I\,n}}{\pi\,r}$$

$$\frac{1}{2}\ \frac{e^{-I\,n}\ \mathrm{BesselJ}(n, 3\,\rho)}{\pi}$$

$$I^{-n}\ e^{-I\,n}\ \mathrm{BesselJ}(n, 3\,\rho)$$

>

> $FSH\left(\frac{1}{3}\cdot 'Conv'\left(1, \frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-2), [r], '2dpolar'\right), r, x, \rho, \psi\right)$

$$\frac{\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}}{\rho}\ \frac{\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{Dirac}(\rho)}{\rho}$$

$$\frac{1}{2}\ \frac{\mathrm{Dirac}(r-2)\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)}{r\,\pi}$$

$$\frac{1}{2}\,\frac{\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{BesselJ}(n,2\rho)}{\pi}$$

$$\frac{2}{3}\,\frac{\pi\,\mathrm{I}^{-n}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)^{2}\mathrm{Dirac}(\rho)\,\mathrm{I}^{-n}\,\mathrm{BesselJ}(n,2\rho)}{\rho}$$

> *convert* (**(4.10.9)** , '*piecewise*' , *n*)

$$\begin{cases} \frac{2}{3}\,\frac{\pi\,\mathrm{Dirac}(\rho)\,\mathrm{BesselJ}(0,2\rho)}{\rho} & n=0 \\ 0 & otherwise \end{cases}$$

> $FSH\left(2\cdot\text{'}Conv\text{'}\left(1,\mathrm{Dirac}\left(x-\frac{\pi}{4}\right),[x],\text{'}angular\text{'}\right),r,x,\rho,\psi\right)$

$$\frac{\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}}{\pi}$$

$$\frac{\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}\,\mathrm{Dirac}(\rho)}{\pi\,\rho}$$

$$\frac{2\,\mathrm{I}^{-n}\left(\begin{cases} 1 & n=0 \\ 0 & otherwise \end{cases}\right)\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}\,\mathrm{Dirac}(\rho)}{\rho}$$

> *convert* (**(4.10.11)** , *piecewise* , *n*)

$$\begin{cases} \frac{2\,\mathrm{Dirac}(\rho)}{\rho} & n=0 \\ 0 & otherwise \end{cases}$$

> $FSH\left(\mathrm{Dirac}\left(x-\frac{\pi}{4}\right),r,x,\rho,\psi\right)$

$$\frac{1}{2}\,\frac{\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}}{\pi}$$

$$\frac{1}{2}\,\frac{\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}\,\mathrm{Dirac}(\rho)}{\pi\,\rho}$$

$$\frac{\mathrm{I}^{-n}\,\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}\,\mathrm{Dirac}(\rho)}{\rho}$$

> *convert* (**(4.10.13)** , *piecewise* , *n*)

$$\frac{\mathrm{I}^{-n}\,\mathrm{e}^{-\frac{1}{4}\mathrm{I}\,n\,\pi}\,\mathrm{Dirac}(\rho)}{\rho}$$

>

> $\#FSH\left(e^{I\cdot\rho 0\cdot r\cdot\cos(\phi 0-\theta)}, r, \theta, \rho, \phi\right)$ assuming $\rho 0 > 0$

Examples for 2D Inverse Polar FT procedure

> $InvFSH\left(piecewise\left(n=0, \frac{2\cdot\pi\cdot\mathrm{Dirac}(\rho)}{\rho}\right), \rho, \psi, r, \theta\right)$

$$\int_0^{\infty}\left(\begin{cases}\frac{2\,\pi\,\mathrm{Dirac}(r)}{r} & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n, r\,\rho)\,\rho\,d\rho$$

$$\frac{1}{2}\,\frac{I^n\left(\int_0^{\infty}\left(\begin{cases}\frac{2\,\pi\,\mathrm{Dirac}(r)}{r} & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n, r\,\rho)\,\rho\,d\rho\right)}{\pi}$$

$$\sum_{n=-\infty}^{\infty}\frac{1}{2}\,\frac{e^{\frac{1}{2}I\,n\,\pi\left(\begin{cases}\frac{2\,\pi\,\mathrm{Dirac}(r)}{r} & n=0\\ 0 & otherwise\end{cases}\right)}\left(\int_0^{\infty}\mathrm{BesselJ}(n, r\,\rho)\,\rho\,d\rho\right)e^{I\,\theta\,n}}{\pi}$$

> $convert\,(\,\mathbf{(4.11.1)}, 'piecewise', n)$

$$\sum_{n=-\infty}^{\infty}\left(\begin{cases}\frac{undefined\ \mathrm{Dirac}(r)}{r} & n=0\\ 0 & otherwise\end{cases}\right)$$

> $InvFSH\left(piecewise\left(n=0, 2\cdot\pi\right), \rho, \psi, r, \theta\right)$

$$\frac{\left(\begin{cases}2\,\pi & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{Dirac}(r)}{r}$$

$$\frac{1}{2}\,\frac{I^n\left(\begin{cases}2\,\pi & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{Dirac}(r)}{\pi\,r}$$

$$\sum_{n=-\infty}^{\infty}\frac{1}{2}\,\frac{e^{\frac{1}{2}I\,n\,\pi\left(\begin{cases}2\,\pi & n=0\\ 0 & otherwise\end{cases}\right)}\mathrm{Dirac}(r)\,e^{I\,\theta\,n}}{\pi\,r}$$

> $convert\,(\,\mathbf{(4.11.3)}, 'piecewise', n)$

$$\frac{\mathrm{Dirac}(r)}{r}$$

> $InvFSH\left(piecewise\left(n=0, \mathrm{BesselJ}(0, 3\cdot\rho)\right), \rho, \psi, r, \theta\right)$

$$\int_0^{\infty}\left(\begin{cases}\mathrm{BesselJ}(0, 3\,r) & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n, r\,\rho)\,\rho\,d\rho$$

$$\frac{1}{2}\,\frac{I^{n}\left(\int_{0}^{\infty}\left(\begin{cases}\mathrm{BesselJ}(0,3\,r) & n=0\\ 0 & otherwise\end{cases}\right)\mathrm{BesselJ}(n,r\,\rho)\,\rho\,d\rho\right)}{\pi}$$

$$\sum_{n=-\infty}^{\infty}\frac{1}{2}\,\frac{e^{\frac{1}{2}\,I\,n\,\pi}\left(\begin{cases}\mathrm{BesselJ}(0,3\,r) & n=0\\ 0 & otherwise\end{cases}\right)\left(\int_{0}^{\infty}\mathrm{BesselJ}(n,r\,\rho)\,\rho\,d\rho\right)e^{I\,\theta\,n}}{\pi}$$

> *convert* (**(4.11.5)**, '*piecewise*', *n*)

$$\frac{\mathrm{BesselJ}(0,3\,r)\ undefined}{\pi}$$

> #*InvFSH* $\left(e^{-I\cdot\rho\cdot r0\cdot\cos(\psi-\theta0)},\rho,\psi,r,\theta\right)$

> #*InvFSH* $\left((2\cdot\pi)^2\cdot\frac{1}{\rho}\cdot\mathrm{Dirac}(\rho)\cdot\mathrm{Dirac}(\psi),\rho,\psi,r,\theta\right)$

>

> #*InvFSH* $\left((2\cdot\pi)^2\cdot\frac{1}{\rho}\cdot\mathrm{Dirac}(\rho-\rho0)\cdot\mathrm{Dirac}(\psi-\psi0),\rho,\psi,r,\theta\right)$ assuming $\rho0>0,\psi0::$ positiv

>

Plots for 2D Polar FT

> *addcoords* (*z_cylindrical*, $[z,r,\theta]$, $[r\cdot\cos(\theta),r\cdot\sin(\theta),z]$)

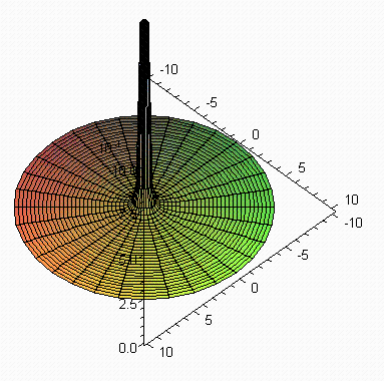
> *DiracApprox* $(x,\varepsilon):=\frac{1}{\pi}\cdot\frac{\varepsilon}{\left(x^2+\varepsilon^2\right)}$

$$DiracApprox:=(x,\varepsilon)\rightarrow\frac{\varepsilon}{\pi\left(x^2+\varepsilon^2\right)}$$

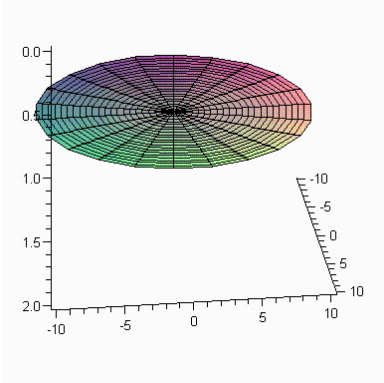
> *f1* $(r,\theta):=\frac{1}{r}\cdot DiracApprox\,(r,0.001)$

$$f1:=(r,\theta)\rightarrow\frac{DiracApprox\,(r,0.001)}{r}$$

> *plot3d* $(f1(r,\theta),r=0..10,\theta=0..2\cdot\pi,$ *coords* $=z\_cylindrical$, *numpoints* $=1000$, *axes* $=framed)$



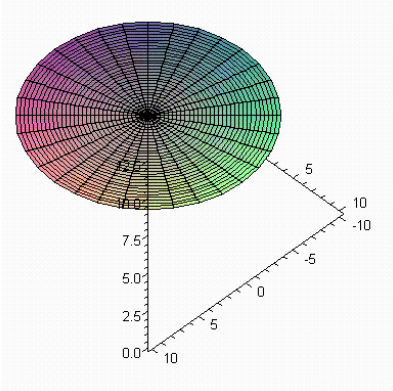
> *plot3d* $(1,r=0..10,\theta=0..2\cdot\pi,$ *coords* $=z\_cylindrical$, *numpoints* $=500$, *axes* $=framed)$



> $F1(\rho,\psi) := 2\cdot\pi$

$$F1 := (\rho,\psi) \to 2\,\pi$$

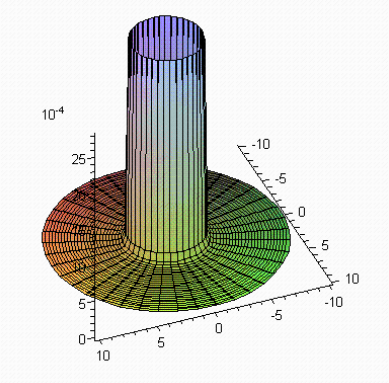
> $plot3d(F1(\rho,\psi), \rho = 0..10, \psi = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 1000, axes = framed)$



> $f2(r,\theta) := \frac{1}{2\cdot\pi\cdot r}\cdot DiracApprox(r-3, 0.001)$

$$f2 := (r,\theta) \to \frac{1}{2}\,\frac{DiracApprox(r-3, 0.001)}{\pi\, r}$$

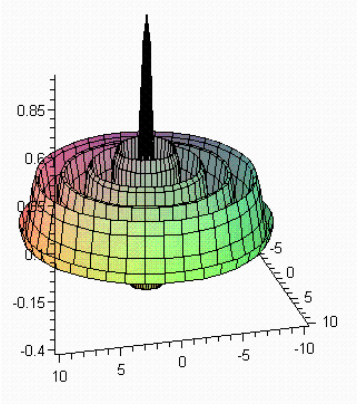
> $plot3d(f2(r,\theta), r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 1500, axes = framed)$



> $F2(\rho,\psi) := \text{BesselJ}(0, 3\cdot\rho)$

$$F2 := (\rho,\psi) \to \text{BesselJ}(0, 3\,\rho)$$

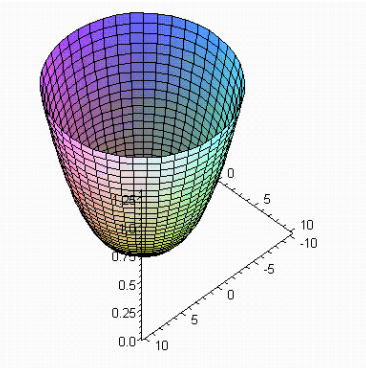
> $plot3d(F2(\rho,\psi), \rho = 0..10, \psi = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 1500, axes = framed)$



> $f3(r,\theta) := Conv\left(1, \frac{1}{2\cdot\pi\cdot r}\cdot \mathrm{Dirac}(r-3), [r], \text{'2 dpolar'}\right)$

$$f3 := (r,\theta) \to Conv\left(1, \frac{1}{2}\ \frac{\mathrm{Dirac}(r-3)}{\pi\, r}, [r], \text{'2 dpolar'}\right)$$

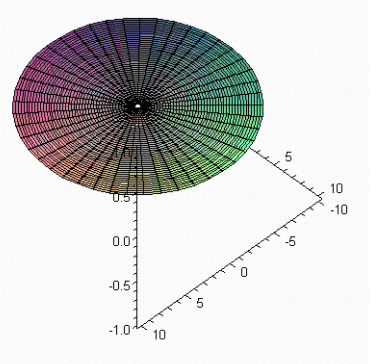
> $plot3d(f3(r,\theta), r = 0..10, \theta = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 2000, axes = framed)$



> $F3(\rho,\psi) := \frac{2\cdot\pi\cdot\mathrm{Dirac}(\rho)\cdot\mathrm{BesselJ}(0, 2\cdot\rho)}{\rho}$

$$F3 := (\rho,\psi) \to \frac{2\,\pi\,\mathrm{Dirac}(\rho)\ \mathrm{BesselJ}(0, 2\,\rho)}{\rho}$$

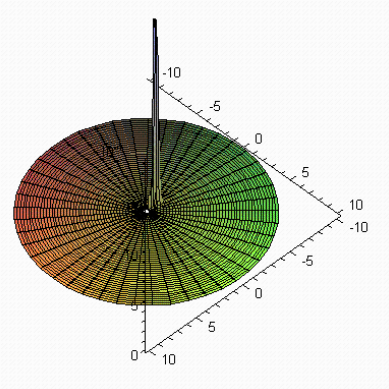
> $plot3d(F3(\rho,\psi), \rho = 0..10, \psi = 0..2\cdot\pi, coords = z\_cylindrical, numpoints = 2000, axes = framed)$



> $f4(r,\theta) := \frac{1}{r}\cdot DiracApprox(r-1, 0.001)\cdot DiracApprox(\theta-\pi, 0.001)$

$$f4 := (r,\theta) \to \frac{DiracApprox(r-1, 0.001)\ DiracApprox(\theta-\pi, 0.001)}{r}$$

> $plot3d\left(f4\left(r,\theta\right), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, numpoints=2000, axes=framed\right)$



> $F4\left(\rho,\psi\right) := \sum_{n=-100}^{100} \exp\left(-\frac{1}{2}\cdot I\cdot n\cdot\pi - I\cdot n\right)\cdot \mathrm{BesselJ}\left(n, 3\cdot\rho\right)\cdot\exp\left(I\cdot\psi\cdot n\right)$

$$F4 := \left(\rho,\psi\right)\rightarrow \sum_{n=-100}^{100} e^{-\frac{1}{2} I n \pi - I n}\ \mathrm{BesselJ}\left(n, 3\,\rho\right)\ e^{I\,\psi\,n}$$

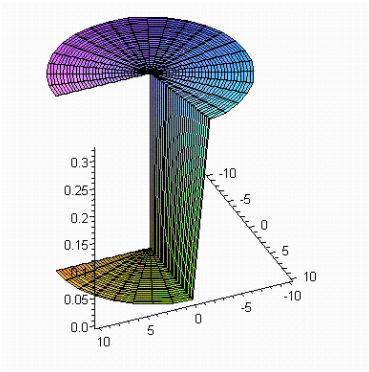
> $F4b\left(\rho,\psi\right) := \exp\left(-3\cdot I\cdot\rho\cdot\cos\left(\psi-1\right)\right)$

$$F4b := \left(\rho,\psi\right)\rightarrow e^{-3\,I\,\rho\,\cos\left(\psi-1\right)}$$

> $f5\left(r,\theta\right) := 2\cdot Conv\left(1, \mathrm{Dirac}\left(\theta-\frac{\pi}{2}\right), [\theta], 'angular'\right)$

$$f5 := \left(r,\theta\right)\rightarrow 2\ Conv\left(1, \mathrm{Dirac}\left(\theta-\frac{1}{2}\,\pi\right), [\theta], 'angular'\right)$$

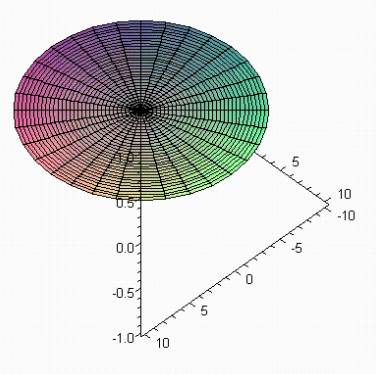
> $plot3d\left(f5\left(r,\theta\right), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, numpoints=1000, axes=framed\right)$



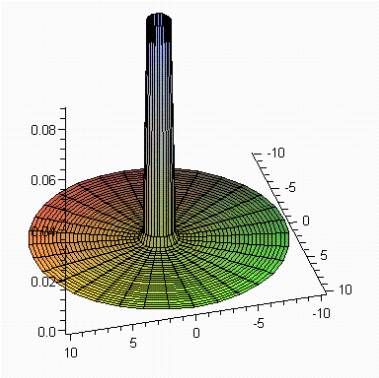
> $F5\left(\rho,\psi\right) := 2\cdot\frac{\mathrm{Dirac}\left(\rho\right)}{\rho}$

$$F5 := \left(\rho,\psi\right)\rightarrow \frac{2\ \mathrm{Dirac}\left(\rho\right)}{\rho}$$

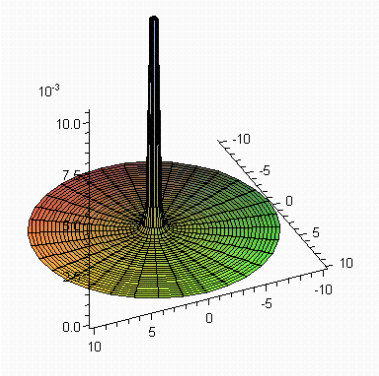
> plot3d(F5, $\rho=0..10, \psi=0..2\cdot\pi$, coords = z_cylindrical, numpoints= 1000, $axes=framed$)



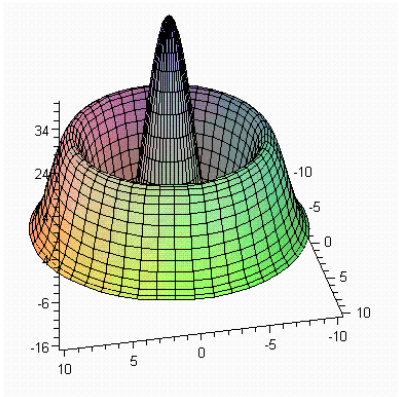
> $plot3d\left(\frac{1}{r}\cdot DiracApprox\,(r-1,0.001), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, numpoints=1000, axes=framed\right)$



> $plot3d\left(\frac{1}{r}\cdot DiracApprox\,(r,0.001), r=0..10, \theta=0..2\cdot\pi, coords=z\_cylindrical, numpoints=1000, axes=framed\right)$



> $plot3d\left(4\cdot\pi^2\cdot \mathrm{BesselJ}(0,\rho),\rho=0..10,\psi=0..2\cdot\pi,coords=z\_cylindrical,numpoints=2000,axes=framed\right)$



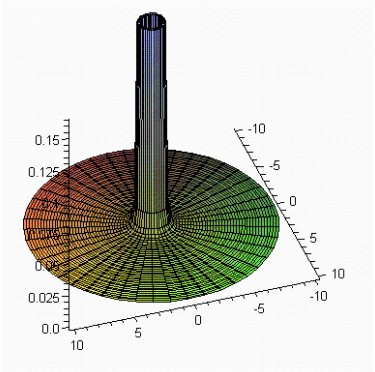
> $Conv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r-1),\frac{1}{r}\cdot \mathrm{Dirac}(r),[r,\theta],2\ dpolar\right)$

$$\frac{2\,\pi\,\mathrm{Dirac}(r-1)}{r-1}$$

> $Conv\left(\frac{1}{r}\cdot \mathrm{Dirac}(r),\frac{1}{r}\cdot \mathrm{Dirac}(r-1),[r,\theta],2\ dpolar\right)$

$$2\,\pi\,\mathrm{Dirac}(r-1)$$

> $plot3d\left(2\cdot\pi\cdot DiracApprox\,(r-1,0.001),r=0..10,\theta=0..2\cdot\pi,coords=z\_cylindrical,numpoints=2000,axes=framed\right)$



>