# Application of Wavelets in Few-Body Problems

A THESIS
SUBMITTED TO THE FACULTY OF
UNIVERSITY OF MINNESOTA
BY

Kuravi Hewawasam

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Advisor: John R Hiller

August 2012

## Acknowledgements

First I would like to pay my sincere gratitude to my advisor, Dr. John R Hiller, for his continuous support and guidance throughout my graduate study. I would also like to thank my committee, Dr. Jonathan Maps and Dr. Mohammed A Hasan, for their valuable time. I would like to acknowledge Dr. Wayne N Polyzou of The University of Iowa for helping me in the initial stages of this project. Last but not least, I would like to thank the Department of Physics of University of Minnesota Duluth for giving me the opportunity for my graduate studies.

To my loving wife, my parents and my brother

## Abstract

This study is an application of wavelet numerical techniques in solving a non-perturbative Yukawa Hamiltonian in light-front quantum field theory. Once the problem is stated in the form of an integral equation, a wavelet basis of a particular scale is used to discretize the problem into a dense matrix. Wavelets are a class of functions with special properties. Daubachies wavelets are a subset of wavelets defined to have vanishing lower order moments, enabling Daubachies 2 and 3 wavelet bases to exactly represent polynomials of degree up to two. These properties make them useful as a basis set for various numerical methods. It was observed that a kernel containing structure in fine scales requires a fine scaling function basis to converge closer to analytical results. Once the kernel matrix is obtained, the wavelet transform followed by an absolute thresholding filters the dense kernel matrix to a sparse matrix. The sparse matrix eigenvalue problem was then solved and compared with the original eigenvalue problem. It was observed that as long as the problem is discretized with a scale fine enough to resolve the features of the kernel, higher levels of filtering would still reproduce eigenvalues that agree with the unfiltered problem.

## Table of Contents

## List of Tables

# List of Figures

# Chapter 01 - Introduction

Wavelets are synonymous for their utility in data compression. The JPEG2000 standard relies on discrete wavelet transform for compression of digital images, and the FBI fingerprint archive uses wavelet-based algorithms in digitizing images. The properties that make wavelets useful in data compression also lead to interesting applications in numerical analysis. In digital signal processing wavelets are used extensively in filtering and compression but in recent years wavelets have been found to be very powerful numerical analysis tools. In [1], wavelets are used as an orthonormal function basis in solving the momentum-space integral equation for a two-body problem with a Malfliet-Tjon potential.

This study tries to apply the same techniques discussed in [1] to use wavelets as an orthonormal function basis for solving a non-perturbative Yukawa Hamiltonian in light-front (LF) quantum field theory. The problem is formulated just as in [2] where the eigenstates of the invariant LF Hamiltonian $H_{LF} = P^- P^+ - P_\perp^2$ with $P^\pm = E \pm P^z$, $\vec{P}_\perp = (P^x, P^y)$, satisfy the LF Schrödinger equation;

$$H_{LF}\Phi_+ = M^2\Phi_+ \tag{1}$$

$E$ denotes energy and $P^x, P^y, P^z$ denote components of momentum. The resulting wave function can be computed analytically [2] or numerically using discretized light-front quantization (DLCQ) [3]. This analysis aims to be an alternative to DLCQ in solving (1) numerically.

The problem statement uses the simplest form of the non-perturbative LF Yukawa Hamiltonian with one Pauli-Villars (PV) fermion and one PV boson. One PV fermion and one PV boson is sufficient to assure perturbative equivalence to Feynman methods without the need for any counter terms. Once the problem statement is specified, it is reduced to a one-dimensional problem by removing the transverse components of the equations. The one-dimensional problem is then projected onto a scaling function basis at a specified scale resulting in a dense kernel matrix. The eigenvalues of the kernel matrix are compared with the analytical results of [2] to check for accuracy.

The actual advantage of using wavelets lies not in the discretization but in the wavelet transformation. The wavelet transform, detailed in chapter 2, followed by an absolute thresholding, filters the kernel matrix to a sparse matrix. The sparse kernel can then be used to find the eigenvalues of the system, which will be much more efficient than solving the original dense kernel matrix eigenvalue problem. We also investigate how filtering at different levels effect the eigenvalues of the problem.

**Wavelets**

Wavelets are a class of functions with certain properties. As the name suggests they have a brief oscillatory behavior. Wavelets are tailor made with certain properties making them very useful in numerical analysis. They can also be used in solving integral and differential equations as basis functions for representing the solution of the equations. While there are many types of wavelets, this study uses a class of orthonormal compactly supported wavelets, which are also orthogonal to low-degree polynomials, introduced by Ingrid Daubechies [4].

The Fourier series is a powerful mathematical tool that can be used in analyzing functions with periodic behavior. Fourier series use an infinite basis set of simple oscillating functions; sines and cosines, of different frequencies, and amplitudes to approximate functions. In 1807 Joseph Fourier introduced this technique for the purpose of solving the heat equation in a metal plate. Fourier series is somewhat ineffective in representing localized functions due to the continuous nature of sines and cosines. Wavelets aim to provide an efficient means for representing localized functions. A superposition of a finite number of wavelets with compact support can represent a localized function, where Fourier analysis will require an infinite sum over the basis functions. Figure 1 is a simple demonstration of a constant being represented exactly by 4 elements of a scaling function basis.

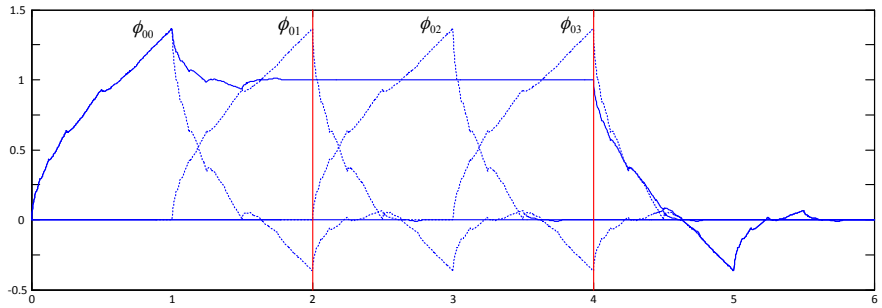



**Figure 1 - Using scaling functions to represent a constant in the interval [2,4].**
Four translations of the scaling function are used to represent the constant localized in the interval [2,4]. The corresponding coefficients for the scaling functions are obtained by projecting the constant onto the scaling functions.

Analogues to the sines and cosines of the Fourier transform basis in the wavelet basis consist of two functions, called the scaling function (father wavelet) denoted by $\phi(x)$ and the mother wavelet function denoted by $\psi(x)$, and analogues to the different frequencies of sines and cosines in Fourier analysis, in the wavelet basis are the scaling functions $\phi_{ij}(x)$ and the wavelet functions $\psi_{ij}(x)$ of all discrete scales and translations. The wavelet transformation relates these basis functions of different scales to each other. The scaling function and the wavelet themselves are fractal like functions that contain structure in both fine and coarse scales and constructing

them is computationally intensive, therefore we will not go into detail on constructing these functions, but all moments of wavelets and scaling functions at any scale can be computed exactly and efficiently. These moments are used in the calculations in this study. Finite linear combinations of scaling basis functions can locally represent low degree polynomials. Therefore functions that can be approximated by low-degree polynomials can be accurately represented by expansion in a scaling function. Since the wavelets are capable of resolving smooth structure in different scales, this method will be useful in problems with such behavior. Even though the Daubachies wavelets are different than smooth functions that are typically used in conventional numerical analysis techniques, wavelets have some special properties, that will be discussed here, that make them powerful tools in numerical techniques.

# Chapter 02 - Wavelets

**Scaling function**

The scaling function has a compact support. The scaling function basis consists of translations and dilations of a single function. Both wavelets and scaling functions can be translated and scaled. The scale and translate operators are defined as follows. For properties of the operators see Appendix A.

$$Df(x) = \frac{1}{\sqrt{2}} f\left(\frac{x}{2}\right) \tag{2}$$

$$Tf(x) = f(x-1) \tag{3}$$

For a scaling function $\phi(x)$ scaling and translation is denoted as follows

$$\phi_{jk}(x) = D^j T^k \phi(x) = D^j \phi(x-k) = 2^{-\frac{j}{2}} \phi\left(\frac{x}{2^j} - k\right)$$

where $j$ and $k$ are integers. $j < 0$ give functions that are finer than the original unscaled function. The scaling function of a coarse scale $D\phi(x)$ (i.e. $j = 1$) can be represented using a finite number of scaling functions themselves of the immediate fine scale $\phi(x)$ (i.e. $j = 0$) in different translations

$$D\phi(x) = \sum_{r=0}^{2K-1} h_r T^r \phi(x) \tag{4}$$

Using (2)

$$\frac{1}{\sqrt{2}} \phi\left(\frac{x}{2}\right) = \sum_{r=0}^{2K-1} h_r T^r \phi(x)$$

where the scaling function can be viewed as the solution of the scaling equation (5)

$$\phi(x) = \sum_{r=0}^{2K-1} \sqrt{2} h_r \phi(2x - r) \tag{5}$$

The scaling function is also normalized

$$\int_{-\infty}^{+\infty} \phi(x)\, dx = 1 \tag{6}$$

Solutions of the scaling equation (5) with finite $K$ are of special interest because these solutions have compact support in the interval $[0, 2K-1]$ such as the Daubechies wavelets. The coefficients $h_l$, called the scaling coefficients, are numerical coefficients, which define the type of scaling function. For most computations, knowledge of the $h_l$'s is all that is needed.

The scaling function $\phi(x)$ is defined so that the translates of the $\phi_{jk}(x)$ on a fixed scale $j$ are orthonormal;

$$\int \phi_{jk}(x)\phi_{jl}(x)dx = \delta_{kl} \tag{7}$$

Self-consistency of the scaling equation (5) and the orthonormality requirement (7) constrain the coefficients $h_l$ so that

$$\sum_{r=0}^{2K-1} h_r = \sqrt{2} \tag{8}$$

$$\sum_{r=0}^{2K-1} h_r h_{r-2n} = \delta_{n0} \tag{9}$$

The $K = 1$ case is known as the Haar wavelet and the Haar scaling coefficients are $h_0 = 1/\sqrt{2}$ and $h_1 = 1/\sqrt{2}$. The Haar wavelet and the scaling function are shown in Figure 2.

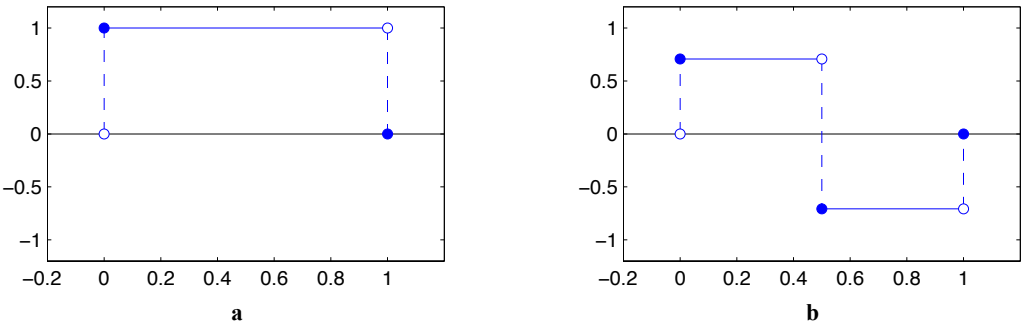



**Figure 2 - a) Haar scaling function and b) wavelet.**

**Wavelet function**

The subspace of square integrable functions in the real number space, $L^2(\mathbb{R})$, spanned by $\phi_{jk}(x)$ of a given scale $j$ is the approximation space used in this study. The scale gets finer as $j$ decreases. If this subspace is denoted by $\mathcal{V}_j$,

$$\mathcal{V}_{j-1} \supset \mathcal{V}_j \tag{10}$$

The wavelet $\psi(x)$ is defined to be orthogonal to the scaling function $\phi(x)$. If the subspace spanned by $\psi_{jk}(x)$ is $\mathcal{W}_j$ then

$$\mathcal{V}_{j-1} = \mathcal{V}_j \oplus \mathcal{W}_j \tag{11}$$

Figure 3 illustrates the relationships (10) and (11).

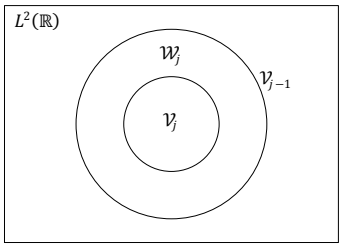



**Figure 3 - The wavelet function space.**
The scaling function basis $\mathcal{V}_{j-1}$ at a scale $(j-1)$ consists of the scaling function basis $\mathcal{V}_j$ and the wavlet basis $\mathcal{W}_j$ at the scale $j$

Orthonormal basis functions of $\mathcal{W}_j$ are linear combinations of the scaling functions of $\mathcal{V}_{j-1}$. The wavelet function $\psi(x)$ from the space $\mathcal{W}_0$ is defined by the following linear combination of the scaling functions $\phi(x)$ of different translations from the space $\mathcal{V}_0$.

$$\psi(x) = D^{-1} \sum_{r=0}^{2K-1} g_r T^r \phi(x) \tag{12}$$

where the coefficients

$$g_r = (-1)^r h_{2K-1-r} \tag{13}$$

In order to calculate the scaling coefficients for higher $K$, more constraints (other than (8) and (9)) are required. By posing certain restrictions different types of wavelets can be generated. The Daubechies wavelets of order $K$ are defined by the conditions that the mother wavelet satisfy

$$\int x^k \psi(x)\, dx = 0 \quad 0 \le k \le K-1 \tag{14}$$

which provides the additional constraint

$$\sum_{r=0}^{2K-1} g_r r^k = \sum_{r=0}^{2K-1} (-1)^r h_{2K-1-r} r^k = 0 \quad 0 \le k \le K-1 \tag{15}$$

The equations (8), (9) and (15) can give the scaling coefficients for $K = 2$ and $K = 3$, shown in Table 1. For a fine scale $j$ the relationship (11) can be iterated to obtain the following

$$\mathcal{V}_j = \mathcal{W}_{j+1} \oplus \mathcal{W}_{j+2} \oplus \cdots \mathcal{W}_{j+m} \oplus \mathcal{V}_{j+m} \tag{16}$$

This process is called the multi-resolution decomposition of $\mathcal{V}_j$ and its illustrated in Figure 4. In order to represent the problem in the scaling function basis, the functions of the problem are projected onto the basis $\mathcal{V}_j$ where $j$ is determined by the finest scale of the problem. This is achieved by quadrature rules and various scaling function moments. The properties used for calculating these quantities are detailed in Appendix A.

**Table 1 - Scaling Coefficients for Daubachies 1, 2 and 3 scaling functions.**

| | DB1 | DB2 | DB3 |
|---|---|---|---|
| $h_0$ | $1/\sqrt{2}$ | $(1+\sqrt{3})/4\sqrt{2}$ | $\left(1+\sqrt{10}+\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$ |
| $h_1$ | $1/\sqrt{2}$ | $(3+\sqrt{3})/4\sqrt{2}$ | $\left(5+\sqrt{10}+\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$ |
| $h_2$ | | $(3-\sqrt{3})/4\sqrt{2}$ | $\left(10-2\sqrt{10}+2\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$ |
| $h_3$ | | $(1-\sqrt{3})/4\sqrt{2}$ | $\left(10-2\sqrt{10}-2\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$ |
| $h_4$ | | | $\left(5+\sqrt{10}-\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$ |
| $h_5$ | | | $\left(1+\sqrt{10}-\sqrt{5+2\sqrt{10}}\right)/16\sqrt{2}$ |

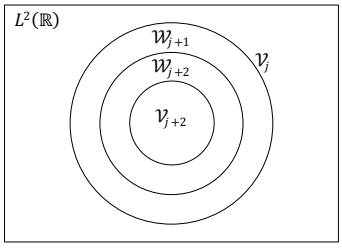



**Figure 4 - Multi-resolution decomposition of scaling function bases.**
The scaling function basis $\mathcal{V}_j$ at a scale $j$ consists of the scaling function basis $\mathcal{V}_{j+2}$ at the scale $(j+2)$ and the wavelet bases from the scale $(j+2)$ to $j$, $\mathcal{W}_{j+2}$ and $\mathcal{W}_{j+1}$

The projection of function $f(x)$ onto the approximation space $\mathcal{V}_j$ spanned by the scaling functions $\phi_{jk}(x)$ can be written as follows

$$f(x) = \sum_l c_l \phi_{jl}(x) \tag{17}$$

where the coefficients

$$c_l = \int f(x)\phi_{jl}(x)\,dx \tag{18}$$

Once $f(x)$ is projected onto $\mathcal{V}_j$ it can be decomposed using (16)

$$f(x) = \sum_l c_l \phi_{jl}(x) = \sum_l d_l \phi_{(j+m)l}(x) + \sum_{k=j+1}^{j+m} \sum_l d_{kl}\psi_{kl}(x) \tag{19}$$

where the coefficients

$$d_l = \int f(x)\phi_{(j+m)l}(x)\,dx \qquad\qquad d_{k,l} = \int f(x)\psi_{kl}(x)\,dx$$

For $f(x)$ that can be approximated by polynomials of order less than $K$, $d_{k,l} \approx 0$, giving an efficient approximation of $f(x)$ by the $d_l$'s. This mapping of coefficients in the fine scale ($c_l$'s) to the coefficients in a coarser scale ($d_l$'s - called approximation coefficients) and the wavelet

coefficients in all the intermediate scales ($d_{k,l}$'s - called detail coefficients) is known as the wavelet transform.

**Wavelet transform**

Given a set of scaling function coefficients in a scale-$j$ ($c_{lj}$'s) let us consider a wavelet transform that maps the $c_{lj}$'s to the adjacent coarse scale-$(j+1)$ scaling functions and wavelets. To obtain the wavelet transform let us map a scaling function of scale-$j$ to the scaling functions and wavelets of adjacent coarse scale-$(j+1)$. Using (19)

$$\phi_{jm}(x) = \sum_l a_l \phi_{(j+1)l}(x) + \sum_l b_l \psi_{(j+1)l}(x)$$

where the coefficient

$$a_l = \int \phi_{jm}(x)\phi_{(j+1)l}(x)\,dx$$

used with (105)

$$a_l = \int \phi_{jm}(x) \sum_{r=0}^{2K-1} h_r \phi_{j(2l+r)}(x)\,dx = \sum_{r=0}^{2K-1} h_r \int \phi_{jm}(x)\phi_{j(2l+r)}(x)\,dx$$

Using the orthonormality of the scaling functions

$$a_l = \sum_{r=0}^{2K-1} h_r \delta_{m(2l+r)} = h_{m-2l}$$

Likewise the coefficient

$$b_l = \int \phi_{jm}(x)\psi_{(j+1)l}\,dx$$

used with (106)

$$b_l = \int \phi_{jm}(x) \sum_{r=0}^{2K-1} g_r \phi_{j(2l+r)}(x)\,dx = \sum_{r=0}^{2K-1} g_r \int \phi_{jm}(x)\phi_{j(2l+r)}(x)\,dx$$

Again using the orthonormality of the scaling functions

$$b_l = \sum_{r=0}^{2K-1} g_r \delta_{m(2l+r)} = g_{m-2l} \tag{20}$$

Therefore

$$\phi_{jm}(x) = \sum_{l=0}^{2K-1} h_{m-2l}\phi_{(j+1)l}(x) + \sum_{l=0}^{2K-1} g_{m-2l}\psi_{(j+1)l}(x) \tag{21}$$

This represents the wavelet transform. Used with (19) the transform for a set of $c_{jm}$'s

$$\sum_m c_{jm}\phi_{jm}(x) = \sum_m c_{jm} \sum_{l=0}^{2K-1} h_{m-2l}\phi_{(j+1)l}(x) + \sum_{l=0}^{2K-1} g_{m-2l}\psi_{(j+1)l}(x)$$

$$\sum_m c_{jm}\phi_{jm}(x) = \sum_l c_{(j+1)m}\phi_{(j+1)l}(x) + \sum_l d_{(j+1)m}\psi_{(j+1)l}(x) \tag{22}$$

where the coefficients

$$c_{(j+1)m} = \sum_{m=2l}^{2l+2K-1} c_{jm}h_{m-2l} \qquad\qquad d_{(j+1)m} = \sum_{m=2l}^{2l+2K-1} c_{jm}g_{m-2l}$$

This gives the most important transform relationship, which is shown in the following matrix form

$$\begin{bmatrix} c_{(j+1)0} \\ c_{(j+1)1} \\ c_{(j+1)2} \\ c_{(j+1)3} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ h_2 & h_3 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & 0 \\ g_2 & g_3 & 0 & 0 & 0 & 0 & g_0 & g_1 \end{bmatrix} \begin{bmatrix} c_{j0} \\ c_{j1} \\ c_{j2} \\ c_{j3} \\ c_{j4} \\ c_{j5} \\ c_{j6} \\ c_{j7} \end{bmatrix} \tag{23}$$

for the Daubachies 2 wavelets. The coefficients are wrapped around at the edges. This is a single level decomposition. In order to reach the coarsest scale the result of this must be transformed recursively

$$\begin{bmatrix} c_{j0} \\ c_{j1} \\ c_{j2} \\ c_{j3} \\ c_{j4} \\ c_{j5} \\ c_{j6} \\ c_{j7} \end{bmatrix} \rightarrow \begin{bmatrix} c_{(j+1)0} \\ c_{(j+1)1} \\ c_{(j+1)2} \\ c_{(j+1)3} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix} \rightarrow \begin{bmatrix} c_{(j+2)0} \\ c_{(j+2)1} \\ d_{(j+2)0} \\ d_{(j+2)1} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix} \rightarrow \begin{bmatrix} c_{(j+3)0} \\ d_{(j+3)0} \\ d_{(j+2)0} \\ d_{(j+2)1} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix}$$

In each successive iteration the transform matrix is changed so that the transformed $d_{(j+1)n}$'s do not transform again.

$$\begin{bmatrix} c_{(j+2)0} \\ c_{(j+2)1} \\ d_{(j+2)0} \\ d_{(j+2)1} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix} = \begin{bmatrix} h_0 & h_1 & h_2 & h_2 & 0 & 0 & 0 & 0 \\ h_2 & h_3 & h_0 & h_1 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ g_3 & g_2 & g_0 & g_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{(j+1)0} \\ c_{(j+1)1} \\ c_{(j+1)2} \\ c_{(j+1)3} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix} \tag{24}$$

The inverse transform that maps the coarse scale to the fine scale can be given by using (106) with (22)

$$\sum_m c_{jm}\phi_{jm}(x) = \sum_l c_{(j+1)m} \sum_{r=0}^{2K-1} h_r \phi_{(j-1)(2k+r)}(x) + \sum_l d_{(j+1)m} \sum_{r=0}^{2K-1} h_r \phi_{(j-1)(2k+r)}(x)$$

$$\sum_m c_{jm}\phi_{jm}(x) = \sum_m \sum_{l=0}^{2K-1} c_{(j+1)m} h_{m-2l} \phi_{jl}(x) + \sum_m \sum_{l=0}^{2K-1} d_{(j+1)m} g_{m-2l} \phi_{jl}(x)$$

where the coefficient

$$c_{jm} = \sum_{l=0}^{2K-1} c_{(j+1)m} h_{m-2l} \phi_{jl}(x) + \sum_{l=0}^{2K-1} d_{(j+1)m} g_{m-2l} \phi_{jl}(x)$$

which is shown in the following matrix form

$$\begin{bmatrix} c_{j0} \\ c_{j1} \\ c_{j2} \\ c_{j3} \\ c_{j4} \\ c_{j5} \\ c_{j6} \\ c_{j7} \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & h_2 & g_0 & 0 & 0 & g_2 \\ h_1 & 0 & 0 & h_3 & g_1 & 0 & 0 & g_3 \\ h_2 & h_0 & 0 & 0 & g_2 & g_0 & 0 & 0 \\ h_3 & h_1 & 0 & 0 & g_3 & g_1 & 0 & 0 \\ 0 & h_2 & h_0 & 0 & 0 & g_2 & g_0 & 0 \\ 0 & h_3 & h_1 & 0 & 0 & g_3 & g_1 & 0 \\ 0 & 0 & h_2 & h_0 & 0 & 0 & g_2 & g_0 \\ 0 & 0 & 0 & h_1 & 0 & 0 & g_3 & g_1 \end{bmatrix} \begin{bmatrix} c_{(j+1)0} \\ c_{(j+1)1} \\ c_{(j+1)2} \\ c_{(j+1)3} \\ d_{(j+1)0} \\ d_{(j+1)1} \\ d_{(j+1)2} \\ d_{(j+1)3} \end{bmatrix}$$

for the Daubachies 2 wavelets. The reverse transform also needs to be iterated to return to the required scale.

Even though this transform is represented using matrices, the wavelet transform algorithm can be implemented much more efficiently and economically by discrete convolution. By convolution of the array $[h_0, h_1, h_2, h_3]$ with the array of values of $[c_{jm}]$ followed by a decimation (or a down-sampling) by a factor of 2 will give the array $[c_{(j+1)m}]$. By convolution of the array $[g_0, g_1, g_2, g_3]$ with the array of values of $[c_{jm}]$ followed by a decimation (or a down-sampling)

by a factor of 2 will give the array $[d_{(j+1)m}]$. Periodic wrap around will be used when the convolution reaches the end. [5] When decomposing into more than one level, successive iterations must be conducted only on the $[c_{jm}]$ array of the previous iterations output. Reverse transform uses the same technique.

When applying the wavelet transform it is essential that the matrix is square with $2^N$ number of elements along one dimension. This may be achieved either by transforming the problem variables so that within the bounds of the problem the number of elements are a power of 2 or by padding of the matrix to the next power of 2. Padding the matrix may cause additional 0 eigenvalues.

**Linear Systems**

In this study the wavelet transform is used as a method of solving a linear system much more efficiently. In [5] this technique is discussed as a fast solution to linear systems. In the case of a linear system

$$\lambda A \cdot x = B \cdot x \tag{25}$$

where $\lambda$ is the eigenvalue of the generalized eigenvalue problem, the operators $A$ and $B$ are wavelet transformed and

$$\tilde{A} = W \cdot A \cdot W^T \qquad\qquad \tilde{B} = W \cdot B \cdot W^T$$

where $W$ is the wavelet transformation matrix given by a series of matrices as in (23) and (24). We then solve

$$\lambda \tilde{A} \cdot \tilde{x} = \tilde{B} \cdot \tilde{x} \tag{26}$$

Once solved the eigenvectors can transformed to the original basis

$$x = W^T \cdot \tilde{x} \tag{27}$$

The eigenvalues however remain the same for both the original and transformed systems, which becomes very useful in this study since the eigenvalues can be compared to check accuracy of this numerical method.

**Filtering**

Once the wavelet transform is conducted the transformed matrices $\tilde{A}$ and $\tilde{B}$ consist of approximation and detail coefficients since $f(x)$ which can be approximated by polynomials of order less than $K$, $d_{k,l} \approx 0$, the detail coefficients of $\tilde{A}$ and $\tilde{B}$ will also be small. Thus filtering these smaller values make $\tilde{A}$ and $\tilde{B}$ sparse matrices, which allows use of efficient techniques for solving sparse systems. The true effectiveness of this technique lies in the wavelet transform

followed by filtering to obtain a spare system from the original. Throwing away small detail coefficients will have little effect on the actual eigenvalues or vectors.

The filtering method used here is an absolute filtering.

$$\left|d_{jk}\right| \leq \epsilon \left|d_{jk}\right|_{max} \tag{28}$$

The detail coefficients are checked for specified threshold $\epsilon \left|d_{jk}\right|_{max}$ (where examples of $\epsilon$ maybe 1%, 5% or 10%) and the values below that threshold are set to zero. This thresholding method is applied after the full wavelet transform is applied.

In order to observe the effect of filtering on the eigenvalue, the threshold value is selected so that it filters a certain percentage of elements in the kernel. To achieve this, the threshold is obtained by sorting all the elements according to their absolute magnitudes and then obtaining the threshold at the specified index of the list (e.g. assume we need to filter out 80% of a 100×100 matrix. First we order the 10000 matrix elements in an ascending order and the 8000th element of this ordered list would have 80% elements below it; therefore, we pick the value of the 8000th element as the threshold. Since we are filtering values equal or below the given threshold, the selected threshold may filter out more than the intended 80% depending on how the magnitudes of the matrix elements are distributed).

## Chapter 03 - Eigenvalue Problem

The Yukawa light cone Hamiltonian, $H_{LF}$ including the Pauli-Villars fields is given by $H_{LF} = P^- P^+ - P_\perp^2$ with $P^\pm = E \pm P^z$, The Yukawa Hamiltonian depicts the interaction between fermion and meson fields, in this particular case the fermion would be a proton and the meson would be a force mediating boson such as a pion. The PV fields are included in this Hamiltonian but pair terms and any other terms that involve anti-fermions are excluded as in [2] and [6].

$$\begin{aligned} P^- = \sum_{i,s} \int d\underline{p} \frac{m_i^2 + \vec{p}_\perp^2}{p^+} (-1)^{i+1} b_{i,s}^\dagger \left(\underline{p}\right) b_{i,s} \left(\underline{p}\right) \\ + \sum_j d\underline{q} \frac{\mu_j^2 + \vec{q}_\perp^2}{q^+} (-1)^{j+1} a_j^\dagger \left(\underline{q}\right) a_j \left(\underline{q}\right) \\ + \sum_{i,j,k,s} \int d\underline{p} d\underline{q} \left\{ \left[ V_{-2s}^* \left(\underline{p}, \underline{q}\right) + V_{2s} \left(\underline{p} + \underline{q}, \underline{q}\right) \right] \right. \\ \times b_{j,s}^\dagger \left(\underline{p}\right) a_k^\dagger \left(\underline{q}\right) b_{i,-s} \left(\underline{p} + \underline{q}\right) \\ \left. + \left[ U_j \left(\underline{p}, \underline{q}\right) + U_i \left(\underline{p} + \underline{q}, \underline{q}\right) \right] \times b_{j,s}^\dagger \left(\underline{p}\right) a_k^\dagger \left(\underline{q}\right) b_{i,s} \left(\underline{p} + \underline{q}\right) + h.c. \right\} \end{aligned} \tag{29}$$

where

$$U_j \left(\underline{p}, \underline{q}\right) \equiv \frac{g}{\sqrt{16\pi^3}} \frac{m_j}{p^+ \sqrt{q^+}} \tag{30}$$

$$V_{2s} \left(\underline{p}, \underline{q}\right) \equiv \frac{g}{\sqrt{8\pi^3}} \frac{\vec{\epsilon}_{2s}^* \cdot \vec{p}_\perp}{p^+ \sqrt{q^+}} \tag{31}$$

$$\vec{\epsilon}_{2s} = -\frac{1}{\sqrt{2}} (2s, i) \tag{32}$$

The first two terms of the Hamiltonian include counts of fermions and bosons while the second and the third terms give measurements of boson emissions while the hermitian conjugate terms give measurements of boson absorptions. $m_0$ is the bare fermion mass and $\mu_0$ is the physical boson mass. $m_1$ and $\mu_1$ are the masses of the PV fermion and boson. The operator $b_{i,s}^\dagger$ creates a fermion of type $i$ and spin $s$, and operator $a_j^\dagger$ creates a boson of type $j$ from the vacuum state $|0\rangle$. The nonzero commutators are

$$\left[ a_j \left(\underline{q}\right), a_j^\dagger \left(\underline{q}'\right) \right] = (-1)^i \delta_{i,j} \delta \left(\underline{q} - \underline{q}'\right) \tag{33}$$

$$\left\{ b_{i,s} \left(\underline{p}\right), b_{j,s'}^\dagger \left(\underline{p}'\right) \right\} = (-1)^i \delta_{i,j} \delta_{s,s'} \delta \left(\underline{p} - \underline{p}'\right) \tag{34}$$

The eigenfunction for the dressed-fermion state in a Fock basis (truncating the expansion to two particles) is

$$\Phi_+(\underline{P}) = \sum_{i=0} z_i b_{i+}^\dagger(\underline{P})|0\rangle + \sum_{\substack{ij=0,1 \\ s=\pm}} \int d\underline{q} f_{ijs}\left(\underline{q}\right) b_{is}^\dagger\left(\underline{P}-\underline{q}\right) a_j^\dagger\left(\underline{q}\right)\left|0\right\rangle \tag{35}$$

The first term represents the one-fermion contribution and $z_i$ denotes probability amplitude for the one-particle contribution, while the second term represents the fermion-boson combined state. The wave functions $f_{ijs}$ that define this state must satisfy the coupled system of equations that result from the light-front Heisenberg equation, $P^+P^-\Phi_+ = M^2\Phi_+$. For the one-boson truncation the coupled equations can be reduced to eight equations [2]

$$\left[M^2 - \frac{m_i^2+q_\perp^2}{1-y} - \frac{\mu_j^2+q_\perp^2}{y}\right] f_{ijs}(y,q_\perp) = \frac{g^2}{16\pi^2}\sum_{\substack{ab=0,1 \\ s'=\pm}} \int_0^1 J^{(0)}_{ijs,abs'}(y,q_\perp{:}\,y',q'_\perp) f_{abs'}(y',q'_\perp) dy' {dq'_\perp}^2 \tag{36}$$

where $y = \frac{q^+}{P^+}$ and

$$J^{(0)}_{ij+,ab+}(y,q_\perp{:}\,y',q'_\perp) = \textstyle\sum_{i'=0}^1 \frac{(-1)^{i'+a+b}}{\sqrt{yy'}} \frac{1}{M^2-m_{i'}^2}\left(\frac{m_i}{1-y}+m_{i'}\right)\left(\frac{m_a}{1-y'}+m_{i'}\right) \tag{37}$$

$$J^{(0)}_{ij+,ab-}(y,q_\perp{:}\,y',q'_\perp) = \textstyle\sum_{i'=0}^1 \frac{(-1)^{i'+a+b}}{\sqrt{yy'}} \frac{1}{M^2-m_{i'}^2}\left(\frac{m_i}{1-y}+m_{i'}\right)\left(\frac{q'_\perp}{1-y'}\right) \tag{38}$$

$$J^{(0)}_{ij-,ab+}(y,q_\perp{:}\,y',q'_\perp) = \textstyle\sum_{i'=0}^1 \frac{(-1)^{i'+a+b}}{\sqrt{yy'}} \frac{1}{M^2-m_{i'}^2}\left(\frac{q_\perp}{1-y}\right)\left(\frac{m_a}{1-y'}+m_{i'}\right) \tag{39}$$

$$J^{(0)}_{ij-,ab-}(y,q_\perp{:}\,y',q'_\perp) = \textstyle\sum_{i'=0}^1 \frac{(-1)^{i'+a+b}}{\sqrt{yy'}} \frac{1}{M^2-m_{i'}^2}\left(\frac{q_\perp}{1-y}\right)\left(\frac{q'_\perp}{1-y'}\right) \tag{40}$$

The analytical solutions for the one-boson wave functions that correspond respectively to the states where the fermion constituent is aligned and anti-aligned with the total spin $J_z$ are [2]

$$f_{ij+}\left(\underline{q}\right) = \frac{P^+}{M^2-\frac{m_i^2+q_\perp^2}{1-q^+/P^+}-\frac{\mu_j^2+q_\perp^2}{q^+/P^+}}\left[\left\{\textstyle\sum_{k=0}^1(-1)^k z_k\right\}U_i\left(\underline{P}-\underline{q},\underline{q}\right) + \textstyle\sum_{k=0}^1(-1)^k z_k U_k\left(\underline{P},\underline{q}\right)\right] \tag{41}$$

$$f_{ij-}\left(\underline{q}\right) = \frac{P^+}{M^2-\frac{m_i^2+q_\perp^2}{1-q^+/P^+}-\frac{\mu_j^2+q_\perp^2}{q^+/P^+}}\left\{\textstyle\sum_{k=0}^1(-1)^k z_k\right\}V_+^*\left(\underline{P}-\underline{q},\underline{q}\right) \tag{42}$$

The analytical result for the coupling constant is [2]

$$g^2 = \frac{(M\mp m_0)(M\mp m_1)}{(m_1-m_0)(\mu_0 I_1 \pm M I_0)} \tag{43}$$

with

$$I_0 = \frac{1}{16\pi^2}\sum\nolimits_{i'b}(-1)^{i'+b}(L_{0i'b} - L_{1i'b}) \tag{44}$$

$$I_1 = \frac{1}{16\pi^2}\sum\nolimits_{i'b}(-1)^{i'+b}\frac{m_{i'}}{\mu_0}L_{0i'b} \tag{45}$$

and

$$L_{ni'b} = \int_0^1 z^n \ln\left[\frac{zm_{i'}^2}{M_j^2} + \frac{(1-z)\mu_b^2}{M_j^2} + z(1-z)\right]dz \tag{46}$$

These can be used to compare the accuracy of the numerical method. The integral (46) is calculated numerically using adaptive Simpson quadrature. The quadrature technique is applied for a range of tolerances to extrapolate the final value.

The angular dependence of these functions can be removed by $\sqrt{P^+}f_{ij+}\left(\underline{q}\right) = f_{ij+}(q^+, q_\perp)$ and $\sqrt{P^+}f_{ij-}\left(\underline{q}\right) = f_{ij-}(q^+, q_\perp)e^{i\phi}$.

By using (30), (31), (32) and $y$ in (41) and simplifying

$$f_{ij+}\left(\underline{q}\right) =$$

$$\frac{P^+}{M^2 - \frac{m_i^2+q_\perp^2}{1-y} - \frac{\mu_j^2+q_\perp^2}{y}}\frac{g}{\sqrt{16\pi^3}}\left[\left\{\sum_{k=0}^{1}(-1)^k z_k\right\}\frac{m_i}{P^+\sqrt{q^+}(1-q^+/P^+)} + \sum_{k=0}^{1}(-1)^k z_k \frac{m_k}{P^+\sqrt{q^+}}\right]$$

Since$1/P^+\sqrt{q^+}$ is common

$$f_{ij+}\left(\underline{q}\right) = \frac{P^+}{M^2 - \frac{m_i^2+q_\perp^2}{1-y} - \frac{\mu_j^2+q_\perp^2}{y}}\frac{g}{\sqrt{16\pi^3}}\left[\left\{\sum_{k=0}^{1}(-1)^k z_k\right\}\frac{m_i}{(1-y)} + \sum_{k=0}^{1}(-1)^k z_k m_k\right]\frac{1}{P^+\sqrt{q^+}}$$

Multiply by $\sqrt{P^+}$

$$\sqrt{P^+}f_{ij+}\left(\underline{q}\right) = \frac{1}{M^2 - \frac{m_i^2+q_\perp^2}{1-y} - \frac{\mu_j^2+q_\perp^2}{y}}\frac{g}{\sqrt{16\pi^3}}\left[\left\{\sum_{k=0}^{1}(-1)^k z_k\right\}\frac{m_i}{(1-y)} + \sum_{k=0}^{1}(-1)^k z_k m_k\right]\frac{\sqrt{P^+}}{\sqrt{q^+}}$$

Using $y = \frac{q^+}{P^+}$

$$\sqrt{P^+}f_{ij+}\left(\underline{q}\right) = \frac{1}{M^2 - \frac{m_i^2+q_\perp^2}{1-y} - \frac{\mu_j^2+q_\perp^2}{y}}\frac{g}{\sqrt{16\pi^3}}\left[\left\{\sum_{k=0}^{1}(-1)^k z_k\right\}\frac{m_i}{(1-y)} + \sum_{k=0}^{1}(-1)^k z_k m_k\right]\frac{1}{\sqrt{y}} \tag{47}$$

Again by using (30), (31), (32) and $y$ in (42) and simplifying ($\underline{P}_\perp = 0$)

$$f_{ij-}\left(\underline{q}\right) = \frac{P^+}{M^2 - \frac{m_i^2+q_\perp^2}{1-y} - \frac{\mu_j^2+q_\perp^2}{y}}\left\{\sum_{k=0}^{1}(-1)^k z_k\right\}\frac{g}{\sqrt{8\pi^3}}\frac{-\vec{\epsilon}_+ \cdot \underline{q}_\perp}{P^+\sqrt{q^+}(1-q^+/P^+)}$$

Since $y = \frac{q^+}{P^+}$, and $1/P^+\sqrt{q^+}$ is common

$$f_{ij-}\left(\underline{q}\right) = \frac{P^+}{M^2 - \frac{m_i^2 + q_\perp^2}{1-y} - \frac{\mu_j^2 + q_\perp^2}{y}} \left\{\sum_{k=0}^{1} (-1)^k z_k\right\} \frac{g}{\sqrt{8\pi^3}} \frac{-\vec{\epsilon}_+ \cdot \underline{q}_\perp}{(1-y)} \frac{1}{P^+\sqrt{q^+}}$$

Multiply by $\sqrt{P^+}$ and using $y = \frac{q^+}{P^+}$

$$\sqrt{P^+} f_{ij-}\left(\underline{q}\right) = \frac{-\vec{\epsilon}_+ \cdot \underline{q}_\perp}{M^2 - \frac{m_i^2 + q_\perp^2}{1-y} - \frac{\mu_j^2 + q_\perp^2}{y}} \frac{g}{\sqrt{8\pi^3}} \left\{\sum_{k=0}^{1} (-1)^k z_k\right\} \frac{1}{(1-y)} \frac{1}{\sqrt{y}} \tag{48}$$

To focus on the longitudinal behavior, we take advantage of the known transverse momentum dependence to write

$$\sqrt{P^+} f_{ij+}\left(\underline{q}\right) = f_{ij+}(y, q_\perp) = F_{i+}(y) \left[\frac{1}{M^2 - \frac{m_i^2 + q_\perp^2}{1-y} - \frac{\mu_j^2 + q_\perp^2}{y}}\right] \tag{49}$$

and

$$\sqrt{P^+} f_{ij-}\left(\underline{q}\right) = f_{ij-}(y, q_\perp) e^{i\phi} = F_-(y) \left[\frac{-\vec{\epsilon}_+ \cdot \underline{q}_\perp}{M^2 - \frac{m_i^2 + q_\perp^2}{1-y} - \frac{\mu_j^2 + q_\perp^2}{y}}\right] \tag{50}$$

where $\vec{\epsilon}_+ = \frac{1}{\sqrt{2}}(1, i)$, $(\cos\phi + i \sin\phi) = e^{i\phi}$ and $\left(-\vec{\epsilon}_+ \cdot \underline{q}_\perp\right) e^{i\phi} = q_\perp$ are used. This reduces the original equation (36) to 3 one-dimensional equations.

We have

$$\frac{16\pi^2}{g^2} F_{i+}(y) = S_{++} + S_{+-} \tag{51}$$

$$\frac{16\pi^2}{g^2} q_\perp F_-(y) = S_{-+} + S_{--} \tag{52}$$

with

$$S_{\pm\pm} = \sum_{ab} \int_0^\infty \int_0^1 J^{(0)}_{ij\pm,ab\pm}(y, q_\perp : y', q'_\perp) f_{ab\pm}(y', q'_\perp) dy'\, dq'^2_\perp \tag{53}$$

Using (49) on the right hand side

$S_{++} =$

$$\sum_a \int_0^1 \sum_{i'=0}^{1} \frac{(-1)^{i'+a}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{m_i}{1-y} + m_{i'}\right) \left(\frac{m_a}{1-y'} + m_{i'}\right) F_{a+}(y') \left[\sum_{b=0}^{1} \int_0^\infty \frac{(-1)^b dq'^2_\perp}{M^2 - \frac{m_a^2 + q'^2_\perp}{1-y'} - \frac{\mu_b^2 + q'^2_\perp}{y'}}\right] dy'$$

Define

$$Q_a(y') = \left[\sum_{b=0}^{1}\int_0^{\infty} \frac{(-1)^b dq_\perp'^{\,2}}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] \tag{54}$$

The $Q_a(y')$ integral is calculated to be

$$Q_a(y') = -y'(1-y')\sum_{b=0}^{1}(-1)^b \log\left[y' m_a^2 + (1-y')\mu_b^2 - y'(1-y')M^2\right] \tag{55}$$

Define new

$$J_{i+,a+}(y{:}\, y') = \sum_{i'=0}^{1} \frac{(-1)^{i'+a}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2}\left(\frac{m_i}{1-y} + m_{i'}\right)\left(\frac{m_a}{1-y'} + m_{i'}\right) \tag{56}$$

and find

$$S_{++} = \sum_{a=0}^{1}\int_0^1 J_{i+,a+}(y{:}\, y') Q_a(y') F_{a+}(y') dy' \tag{57}$$

Using (50) on the right hand side

$S_{+-} =$

$$\sum_{ab}\int_0^{\infty}\int_0^1 \sum_{i'=0}^{1} \frac{(-1)^{i'+a+b}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2}\left(\frac{m_i}{1-y} + m_{i'}\right)\left(\frac{q_\perp'}{1-y'}\right) F_-(y') \left[\frac{-\vec{\epsilon}_+ \cdot \underline{q}_\perp'}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] e^{-i\phi} dy'\, dq_\perp'^{\,2}$$

Using $\vec{\epsilon}_+ = \frac{1}{\sqrt{2}}(1, i)$ and $(\cos\phi + i\sin\phi) = e^{i\phi}$

$S_{+-} =$

$$\int_0^1 \sum_{i'=0}^{1} \frac{(-1)^{i'}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2}\left(\frac{m_i}{1-y} + m_{i'}\right)\left(\frac{1}{1-y'}\right) F_-(y') \left[\sum_{ab}\int_0^{\infty} \frac{(-1)^{a+b} q_\perp'^{\,2} dq_\perp'^{\,2}}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] dy'$$

Define

$$P(y') = \left[\sum_{ab}\int_0^{\infty} \frac{(-1)^{a+b} q_\perp'^{\,2} dq_\perp'^{\,2}}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] \tag{58}$$

The $P(y')$ integral is calculated to be

$$P(y') = -y'(1-y')\sum_{a,b=0}^{1} \frac{(-1)^{a+b}\left(y' m_a^2 + (1-y')\mu_b^2 - y'(1-y')M^2\right)}{\log\left[y' m_a^2 + (1-y')\mu_b^2 - y'(1-y')M^2\right]} \tag{59}$$

Define new

$$J_{i+,-}(y:y') = \sum_{i'=0}^{1} \frac{(-1)^{i'}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{m_i}{1-y} + m_{i'}\right)\left(\frac{1}{1-y'}\right) \tag{60}$$

and find

$$S_{+-} = \int_0^1 J_{i+,-}(y:y') P(y') F_{a-}(y') dy' \tag{61}$$

Using (49) on the right hand side

$S_{-+} =$

$$q_\perp \sum_a \int_0^1 \sum_{i'=0}^{1} \frac{(-1)^{i'+a}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{1}{1-y}\right)\left(\frac{m_a}{1-y'} + m_{i'}\right) F_{a+}(y') \left[\sum_{b=0}^{1} \int_0^\infty \frac{(-1)^b dq_\perp'^{\,2}}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] dy'$$

Define new

$$J_{-,a+}(y:y') = \sum_{i'=0}^{1} \frac{(-1)^{i'+a}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{1}{1-y}\right)\left(\frac{m_a}{1-y'} + m_{i'}\right) \tag{62}$$

to have

$$S_{-+} = q_\perp \sum_{a=0}^{1} \int_0^1 J_{-,a+}(y:y') Q_a(y') F_{a+}(y') dy' \tag{63}$$

We defined $Q_a(y')$ in (54) and calculated the integral in (55)

Using (50) on the right hand side

$S_{--} =$

$$\sum_{ab} \int_0^\infty \int_0^1 \sum_{i'=0}^{1} \frac{(-1)^{i'+a+b}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{q_\perp}{1-y}\right)\left(\frac{q_\perp'}{1-y'}\right) F_-(y') \left[\frac{-\vec{\epsilon}_+ \cdot \vec{q}_\perp'}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] dy'\, dq_\perp'^{\,2}$$

Using $\vec{\epsilon}_+ = \frac{1}{\sqrt{2}}(1, i)$ and $(\cos\phi + i \sin\phi) = e^{i\phi}$

$$S_{--} = q_\perp \int_0^1 \sum_{i'=0}^{1} \frac{(-1)^{i'}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{1}{1-y}\right)\left(\frac{1}{1-y'}\right) F_-(y') \left[\sum_{ab} \int_0^\infty \frac{(-1)^{a+b} q_\perp'^{\,2} dq_\perp'^{\,2}}{M^2 - \frac{m_a^2 + q_\perp'^{\,2}}{1-y'} - \frac{\mu_b^2 + q_\perp'^{\,2}}{y'}}\right] dy'$$

Define new

$$J_{-,-}(y:y') = \sum_{i'=0}^{1} \frac{(-1)^{i'}}{\sqrt{yy'}} \frac{1}{M^2 - m_{i'}^2} \left(\frac{1}{1-y}\right)\left(\frac{1}{1-y'}\right) \tag{64}$$

to have

$$S_{--} = q_{\perp} \int_0^1 J_{i-,-}(y:y')P(y')F_-(y')dy' \tag{65}$$

We defined $P(y')$ in (58) and calculated the integral in (59)

By using (57), (61), (63) and (65) with (51) and (52)

$$\begin{aligned} \frac{16\pi^2}{g^2} F_{i+}(y) = &\sum_{a=0}^{1} \int_0^1 J_{i+,a+}(y:y')Q_a(y')F_{a+}(y')dy' \\ &+ \int_0^1 J_{i+,-}(y:y')P(y')F_-(y')dy' \end{aligned} \tag{66}$$

$$\frac{16\pi^2}{g^2} F_-(y) = \sum_{a=0}^{1} \int_0^1 J_{-,a+}(y:y')Q_a(y')F_{a+}(y')dy' + \int_0^1 J_{-,-}(y:y')P(y')F_-(y')dy' \tag{67}$$

From the complete analytic solutions (47) and (48) we have the analytic solutions for these equations.

$$F_{i+}(y) = \frac{g}{\sqrt{16\pi^3}} \left[ \left\{ \sum_{k=0}^{1} (-1)^k z_k \right\} \frac{m_i}{(1-y)} + \sum_{k=0}^{1} (-1)^k z_k m_k \right] \frac{1}{\sqrt{y}} \tag{68}$$

and

$$F_-(y) = \frac{g}{\sqrt{8\pi^3}} \left\{ \sum_{k=0}^{1} (-1)^k z_k \right\} \frac{1}{(1-y)} \frac{1}{\sqrt{y}} \tag{69}$$

# Chapter 04 - Numerical Method

**General Formulation**

The integral equations are of the general form

$$\beta f(y) = \int_a^b K(y:y')f(y')dy' \tag{70}$$

where $\beta = \frac{16\pi^2}{g^2}$ is the eigenvalue. To represent the function $f(y)$ in the scaling function basis, we can write

$$f(y) \approx \sum_n f_n \phi_{jn}(y) \tag{71}$$

where $f_n$ is the projection of $f(y)$ onto the scaling function basis at an appropriate scale

$$f_n = \int f(y)\phi_{jn}(y)dy \tag{72}$$

The problem can then be written as follows

$$\beta \sum_n f_n \phi_{jn}(y) = \int_a^b K(y:y') \sum_n f_n \phi_{jn}(y')\, dy' \tag{73}$$

By multiplying by $\phi_{jm}(y)$ and integrating over the region, this becomes

$$\beta \sum_n f_n \int_a^b \phi_{jm}(y)\phi_{jn}(y)dy = \sum_n f_n \int_a^b \int_a^b K(y:y')\phi_{jm}(y)\phi_{jn}(y')dy\, dy' \tag{74}$$

We define an overlap matrix

$$N_{mn} = \int_a^b \phi_{jm}(y)\phi_{jn}(y)dy \tag{75}$$

and a matrix that represents the kernel by $L_{mn}$

$$L_{mn} = \int_a^b \int_a^b K(y:y')\phi_{jm}(y)\phi_{jn}(y')dy\, dy' \tag{76}$$

The elements of $L_{mn}$ are calculated using moments and quadrature techniques discussed below. Denote the obtained quadrature points and weights by $\{y_i, w_i\}$. These can be used to calculate the kernel matrix $L_{mn}$.

$$L_{mn} = \sum_{i=0}^{N} \sum_{j=0}^{N} K(y_i:y_j) \times w_i \times w_j \tag{77}$$

The discretized problem will now take the following form

$$\beta \sum_n N_{mn} f_n = \sum_n L_{mn} f_n \tag{78}$$

where the matrix form is a generalized eigenvalue problem.

Once the generalized eigenvalue problem is solved, the eigenvalues $\beta$ are known. Using the eigenvalues $\beta$ and eigenvectors $f_n$ and substituting back in equation (73)

$$\beta f(y_s) = \int_a^b K(y_s : y') \sum_n f_n \phi_{jn}(y')\, dy' \tag{79}$$

The solution $f(y)$ can be evaluated at any set of $y_s$ points.

$$f(y_s) = \frac{1}{\beta} \sum_n f_n \int_a^b K(y_s : y') \phi_{jn}(y') dy' \tag{80}$$

The integral $\int_a^b K(y_s : y') \phi_{jn}(y') dy'$ can be evaluated using moments and the same quadrature techniques as in (77)

$$f(y_s) = \frac{1}{\beta} \sum_n f_n \sum_{j=0}^{N} K(y_s : y_j) \times w_j \tag{81}$$

The eigenvalue problem considered in this study (36) is of the form (70). It can be written as follows

$$\beta \begin{bmatrix} F_{0+} \\ F_{1+} \\ F_{-} \end{bmatrix} = \begin{bmatrix} J_{0+,0+} \cdot Q_0 & J_{0+,1+} \cdot Q_1 & J_{0+,-} \cdot P \\ J_{1+,0+} \cdot Q_0 & J_{1+,1+} \cdot Q_1 & J_{1+,-} \cdot P \\ J_{-,0+} \cdot Q_0 & J_{-,1+} \cdot Q_1 & J_{-,-} \cdot P \end{bmatrix} \begin{bmatrix} F_{0+} \\ F_{1+} \\ F_{-} \end{bmatrix} \tag{82}$$

where the $J_{a\pm,b\pm}$, $J_{\pm,b\pm}$ and $J_{\pm,\pm}$ terms are given by (56), (60), (62) and (64). Comparing with the general form (70), $f(y) \in \{F_{0+}, F_{1+}, F_{-}\}$ and the eigenvalue is

$$\beta = \frac{16\pi^2}{g^2} \tag{83}$$

Wavelet numerical analysis techniques are used to discretize the problem. The resulting eigenvalue problem is solved using a standard solver to obtain $\beta$ and then $g = \frac{4\pi}{\sqrt{\beta}}$. Since there is an analytical expression (43) for $g$, the results from this method can be compared with the analytical results to check the accuracy of the method.

**Calculation of the overlap matrix**

By definition Daubachies wavelets are orthogonal, i.e.

$$\int \phi_{jm}(x) \phi_{jn}(x) dx = \delta_{mn}$$

but the integrals discussed, are definite integrals. The overlap will not be trivial at the limits; when the support of either $\phi_{jm}$ or $\phi_{jn}$ contains the limits $a$ or $b$. Let us write in general

$$N_{m,n}^{a,b} = \int_a^b \phi_{jm}(x)\phi_{jn}(x)dx$$

Since we will not encounter different scales the subscript $j$ can be omitted. Using (97)

$$N_{m,n}^{a,b} = \int_a^b \phi(x-m)\phi(x-n)dx$$

By using $y = x - m$

$$N_{m,n}^{a,b} = \int_{a-m}^{b-m} \phi(y)\phi(y+m-n)dy$$

Using property (105)

$$N_{m,n}^{a,b} = \int_{a-m}^{b-m} \sum_{l=0}^{2K-1} \sqrt{2}h_l\phi(2y-l) \sum_{l'=0}^{2K-1} \sqrt{2}h_{l'}\phi(2(y+m-n)-l')\,dy$$

$$= \sum_{l=0}^{2K-1}\sum_{l'=0}^{2K-1} h_l h_{l'} \int_{a-m}^{b-m} \phi(2y-l)\phi(2y+2m-2n-l')2dy$$

Using $x = 2y - l$

$$N_{m,n}^{a,b} = \sum_{l=0}^{2K-1}\sum_{l'=0}^{2K-1} h_l h_{l'} \int_{2(a-m)-l}^{2(b-m)-l} \phi(x)\phi(x+2m-2n-l'+l)dy$$

gives the linear system

$$N_{m,n}^{a,b} = \sum_{l=0}^{2K-1}\sum_{l'=0}^{2K-1} h_l h_{l'} N_{0,(-2m+2n+l'-l)}^{(2a-2m-l),(2b-2m-l)} \tag{84}$$

or

$$N_{0,n-m}^{a-m,b-m} = \sum_{l=0}^{2K-1}\sum_{l'=0}^{2K-1} h_l h_{l'} N_{0,(-2m+2n+l'-l)}^{(2a-2m-l),(2b-2m-l)}$$

which can be solved for the non-trivial $N_{0,n-m}^{a-m,b-m}$ (for $a-(2K-1) \le m \le a$ or for $b-(2K-1) \le m \le b$ i.e. when the support of the scaling function contain the limits) using the following trivial cases.

$$N_{0,n-m}^{a-m,b-m} = 1, \quad n = m, a \le m \le b-(2K-1) \tag{85}$$

$$N_{0,n-m}^{a-m,b-m} = 0, \quad n \ne m, a > m > b-(2K-1) \tag{86}$$

*Application*

For the case $K = 2$ (DB2 scaling function). Let us suppose the upper bound $b$ to be $10$. Using (84) to write $N_{-1,-1}^{0,10}$, $N_{-1,-2}^{0,10}$, $N_{-2,-2}^{0,10}$, $N_{-2,-1}^{0,10}$ and using the trivial cases (85) and (86) the following system is constructed also $N_{-1,-1}^{0,10} = N_{0,0}^{1,11}$, $N_{-1,-2}^{0,10} = N_{0,-1}^{1,11}$, $N_{-2,-2}^{0,10} = N_{0,0}^{2,12}$, $N_{-2,-1}^{0,10} = N_{0,1}^{2,12}$. All limits that are not in the supports of the scaling functions can be denoted by a $\times$

$$\begin{bmatrix} N_{0,-1}^{1,\times} \\ N_{0,0}^{1,\times} \\ N_{0,0}^{2,\times} \\ N_{0,1}^{2,\times} \end{bmatrix} = \begin{bmatrix} h_1h_2 & h_1h_3 & h_0h_2 & h_0h_3 \\ h_1h_0 & h_1h_1 & h_0h_0 & h_0h_1 \\ h_3h_2 & h_3h_3 & h_2h_2 & h_2h_3 \\ h_3h_0 & h_3h_1 & h_2h_0 & h_2h_1 \end{bmatrix} \begin{bmatrix} N_{0,-1}^{1,\times} \\ N_{0,0}^{1,\times} \\ N_{0,0}^{2,\times} \\ N_{0,1}^{2,\times} \end{bmatrix} + \begin{bmatrix} 0 \\ h_2^2 + h_3^2 \\ 0 \\ 0 \end{bmatrix}$$

which can be solved for the non-trivial terms $N_{0,-1}^{1,\times}$, $N_{0,0}^{1,\times}$, $N_{0,0}^{2,\times}$, $N_{0,1}^{2,\times}$. Likewise suppose the lower bound $a$ to be $0$.

$$\begin{bmatrix} N_{0,-1}^{\times,1} \\ N_{0,0}^{\times,1} \\ N_{0,0}^{\times,2} \\ N_{0,1}^{\times,2} \end{bmatrix} = \begin{bmatrix} h_1h_2 & h_1h_3 & h_0h_2 & h_0h_3 \\ h_1h_0 & h_1h_1 & h_0h_0 & h_0h_1 \\ h_3h_2 & h_3h_3 & h_2h_2 & h_2h_3 \\ h_3h_0 & h_3h_1 & h_2h_0 & h_2h_1 \end{bmatrix} \begin{bmatrix} N_{0,-1}^{\times,1} \\ N_{0,0}^{\times,1} \\ N_{0,0}^{\times,2} \\ N_{0,1}^{\times,2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ h_0^2 + h_1^2 \\ 0 \end{bmatrix}$$

which can be solved for the non-trivial terms $N_{0,-1}^{\times,1}$, $N_{0,-1}^{\times,1}$, $N_{0,-1}^{\times,1}$, $N_{0,-1}^{\times,1}$. The same process can be used for calculating the overlap matrix for $K = 3$ (DB3 scaling functions)

**Quadrature rules**

In projecting functions onto the wavelet basis as in (71), we have to calculate the coefficients $f_n$ from (72) i.e. the projection of the function onto each scaling function in the basis. The simplest quadrature rule to achieve this is the **single-point quadrature rule**. The quadrature point is defined by the first-order moment of the scaling function

$$\langle x \rangle_\phi = \int x\phi(x)\, dx$$

which gives

$$\int (a + bx)\phi(x)\, dx = a \int \phi(x)\, dx + b \int x\phi(x)\, dx = a + b\langle x \rangle_\phi$$

$$\int (a + bx)\phi(x)\, dx = a + b\langle x \rangle_\phi \tag{87}$$

A theorem proved in [7] states that for orthogonal scaling function bases

$$\langle x^2 \rangle_\phi = \langle x \rangle_\phi^2 \tag{88}$$

which gives

$$\int (a + bx + cx^2)\phi(x)\,dx = a + b\langle x\rangle_\phi + c\langle x\rangle_\phi^2 \tag{89}$$

The expressions (87) and (89) are exact and can be summarized as follows

$$\int P(x)\phi(x)\,dx = P\big(\langle x\rangle_\phi\big) \tag{90}$$

when $P(x)$ is of order less than 3. Also $\langle x\rangle_\phi$ can be considered as a single quadrature point, used with a weight, $w = 1$, that can integrate the product of a scaling function ($\phi(x)$) and a polynomial of degree less than 3, exactly, providing us a **single-point quadrature rule** with the point and weight

$$\{x_1, w_1\} = \big\{\langle x\rangle_\phi, 1\big\}$$

Even though the **single-point quadrature rule** successfully approximates the integral of the product of scaling function with a given function, the quadrature rule works only when the support of the considered scaling function does not contain a boundary of the problem.

When the problem boundaries are within the support of the scaling function we will have to resort to a **multi-point quadrature rule.** In such cases the quadrature points are taken to be those used in Gauss-Legendre quadrature. The weights for these quadrature points however are calculated by setting up a set of linear equations using the known partial moments (calculated using techniques discussed in the Appendix B).

Consider a case where the lower limit of the integral is 0 and the scaling function considered has 0 in its support

$$\langle x^m\rangle_{\phi_{0k}} = \int_0^\infty x^m \phi_{0k}(x)\,dx = \sum_{i=1}^{N} x'^m_i w'_i$$

The $x_i$'s are those of Gauss-Legendre quadrature in the interval $[0, k + 2K - 1]$, i.e. 0 and the upper bound of the support of the scaling function in question. In order to find the appropriate weights, the following linear system, the number of points, $N$ is decided by the wavelet used, where $N = K + 1$. This set of equations is written in the matrix form

$$x = Aw$$

where $A$ has the form

$$A = \begin{pmatrix} x_1^0 & x_2^0 & \cdots & x_N^0 \\ x_1^1 & x_2^1 & \cdots & x_N^1 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^N & x_2^N & \cdots & x_N^N \end{pmatrix}_{N\times N}$$

$x$ has the form

$$x = \begin{pmatrix} \langle x^0 \rangle_\phi \\ \langle x^1 \rangle_\phi \\ \vdots \\ \langle x^N \rangle_\phi \end{pmatrix}_{N\times 1}$$

and $w$ has the form

$$w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}_{N\times 1}$$

The same technique holds when an upper integral limit overlaps the scaling function support. This technique returns the **multi-point quadrature rule** for when the scaling function overlaps the upper or lower limits of the integral and the single point quadrature discussed previously returns the **single-point quadrature rule** for when the scaling function is well within the limits of the integral.

Both quadrature rules discussed must be generalized to integrate the product of any scaling function ($\phi_{jn}$) and a polynomial. Consider a general quadrature rule with the point and weight $\{x_i, w_i\}$

$$\int x^m \phi(x)\, dx \approx \sum_i x_i^m w_i \tag{91}$$

for a scaling function of any scale and translation

$$\int x^m \phi_{jn}(x)\, dx = \int x^m T^n D^j \phi(x)\, dx$$

using (108)

$$\begin{aligned} \int x^m \phi_{jn}(x)\, dx &= \int \left(T^{-n} D^{-j} x^m\right) \phi(x)\, dx \\ &= \int \left(T^{-n} 2^{\frac{j}{2}} \left(2^j x\right)^m\right) \phi(x)\, dx \\ &= 2^{j\left(\frac{1}{2}+m\right)} \int (x+n)^m \phi(x)\, dx \end{aligned}$$

From (91)

$$\int x^m \phi_{jn}(x)\, dx \approx 2^{j\left(\frac{1}{2}+m\right)} \sum_i (x_i+n)^m w_i = \sum_i \left(2^j (x_i+n)\right)^m 2^{\frac{j}{2}} w_i$$

The modified quadrature point and weight $\{x_i', w_i'\}$ are

$$x_i' = 2^j (x_i + n) \qquad\qquad w_i' = 2^{\frac{j}{2}} w_i \tag{92}$$

For the **single-point quadrature rule** with the point and weight $\{x_1, w_1\}$ the modified quadrature point and weight (for any $\phi_{jn}$) $\{x_1', w_1'\}$ are

$$x_1^{'} = 2^j(x_1 + n) = 2^j\left(\langle x\rangle_\phi + n\right) \qquad\qquad w_1^{'} = 2^{\frac{j}{2}} w_1 = 2^{\frac{j}{2}}$$

These quadrature rules yield exact results for polynomials of degrees less than 3 within the support of the considered wavelet. Therefore, for any function that can be piecewise approximated by polynomials of degrees less than 3, this quadrature rule will give fairly accurate results.

The advantage of using the Daubachies bases is that by definition (14) lower order moments $(0 \leq k \leq K-1)$ of Daubachies wavelets are zero. As consequence $d_{kl}$ of (19) will be zero. Once the problem is discretized using these quadrature techniques, and the $c_l$'s of a function $f(x)$ are known, like in (19), the wavelet transform will map them to a set of $d_l$'s and $d_{kl}$'s, where the $d_{kl}$'s will be significantly small compared to $d_l$'s. Thus filtering $d_{kl}$'s on the basis of magnitude will still result in a good approximation of $f(x)$.

# Chapter 05 - Results

The numerical method is tested with different sets of mass parameters. The eigenvalue problem (82) is first represented in the scaling function basis at different scales (resolutions) and then solved. The masses selected to test the numerical method are chosen to test different aspects of the method.

***Test 1***

First set of masses used was selected to look at the convergence of the results with the analytical results. The Pauli-Villars fermion mass was selected to be twice that of the boson, i.e. $m_1 = 2\mu_1$ but both masses are kept at low values compared to the boson mass $\mu_0$. The masses selected are $\mu_1 = 10\mu_0$, $m_1 = 20\mu_0$ and $m_0 = -5\mu_0$. Figure 5 and Figure 6 show the real parts of the eigenvalues for various resolutions. However the DB2 case did not return any complex eigenvalues while the DB3 case returned complex eigenvalues at lower resolutions.

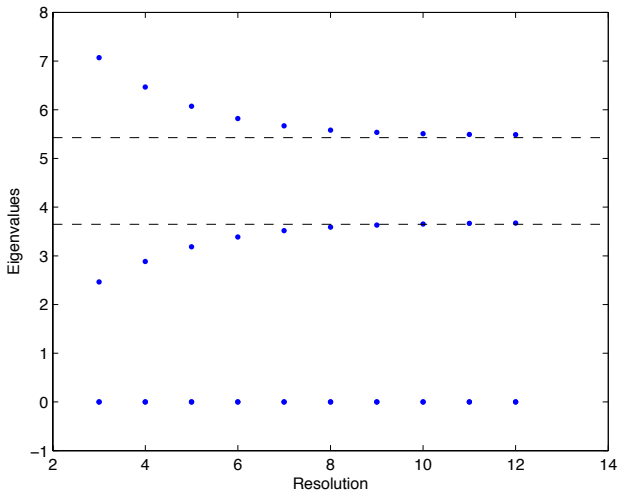



**Figure 5 - Eigenvalues of test 1 in the DB2 scaling function basis at different scales.**
The dashed lines show the expected values, obtained using the analytical results. The mass parameters used are $M = \mu_0$, $\mu_1 = 10\mu_0$, $m_1 = 20\mu_0$, $m_0 = -5\mu_0$.

Figure 5 and Figure 6 show a good convergence of the two eigenvalues, for the lower masses. The actual coupling parameter, $g$, is calculated from the eigenvalues and compared with the analytical result. In order to extrapolate the result to a finer scale, the calculated coupling values need to be fitted to a curve. It was determined that the quantity that shows a power law variation

with the inverse of the resolution is the eigenvalue, $\beta$, rather than the coupling, $g$, therefore the following model was devised using (83).

$$\beta(x) = ax^b + c \Rightarrow g(x) = \frac{4\pi}{\sqrt{ax^b + c}} \tag{93}$$

where $x$ is the inverse of the resolution. Some of the lower resolutions eigenvalues carry lower weights in the fit calculation since they are inaccurate than the higher resolution ones. The results are plotted in Figure 7. Once the fit is obtained it can be extrapolated to $x \to 0$ as follows

$$\lim_{x\to 0} \frac{4\pi}{\sqrt{ax^b + c}} \to \frac{4\pi}{\sqrt{c}} \tag{94}$$

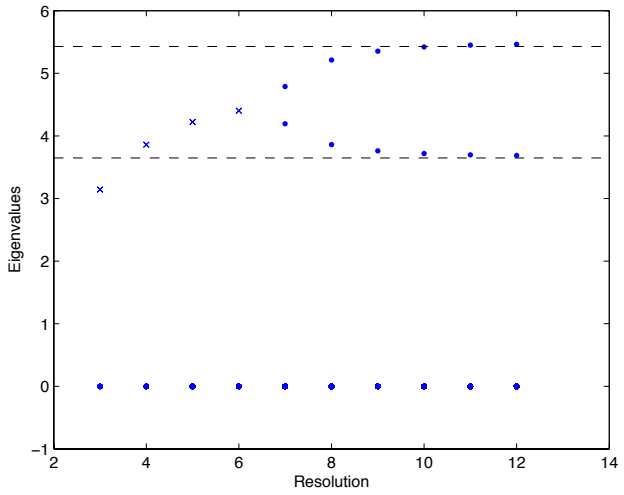



**Figure 6 - Same as Figure 5 but for DB3 scaling function bases.**
The mass parameters are same as Figure 5. The ×'s show the real part of the complex eigenvalues.

**Table 2 - 4 Fit outputs for test 1.**
The fit results for the form (93) is shown and the two values in parenthesis next to each coefficient show the upper and the lower 95% confidence intervals for each coefficient. Goodness of the fits are also shown.

| Eigenvalue | DB2 fit | DB3 fit |
|---|---|---|
| 1<br>(g=6.579) | a = -15.45 (-19.47, -11.43)<br>b = 2.04 (1.773, 2.308)<br>c = 3.792 (3.715, 3.869)<br>Goodness of fit:<br>SSE: 0.006804<br>R-square: 0.9966<br>Adjusted R-square: 0.9957<br>RMSE: 0.03118 | a = 1.373e+05 (-8.557e+05, 1.13e+06)<br>b = 6.504 (3.123, 9.885)<br>c = 3.675 (3.655, 3.695)<br>Goodness of fit:<br>SSE: 2.032e-07<br>R-square: 0.9999<br>Adjusted R-square: 0.9997<br>RMSE: 0.0004508 |
| 2<br>(g=5.394) | a = 26.88 (18.28, 35.47)<br>b = 2.305 (2.017, 2.593)<br>c = 5.379 (5.311, 5.447)<br>Goodness of fit:<br>SSE: 0.001509<br>R-square: 0.9964<br>Adjusted R-square: 0.9954<br>RMSE: 0.01468 | a = -8.213e+04 (-3.646e+05, 2.004e+05)<br>b = 6.055 (4.259, 7.852)<br>c = 5.49 (5.432, 5.548)<br>Goodness of fit:<br>SSE: 0.0002013<br>R-square: 0.998<br>Adjusted R-square: 0.9966<br>RMSE: 0.008192 |

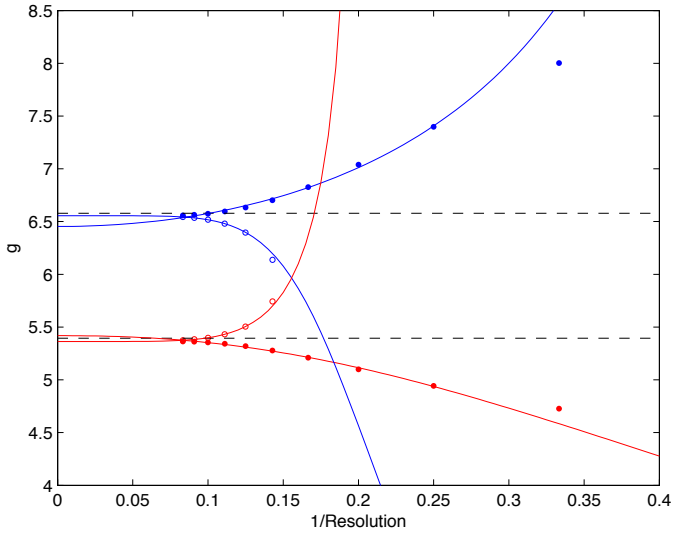



**Figure 7 - Calculated coupling of test 1 at different scales.**
The filled symbols show the DB2 scaling function basis results and open symbols show the DB3 scaling function basis results. The dashed lines show the expected value, obtained using the analytical results. The fits that correspond to each set are also shown.

At these masses the kernel will contain structure on a quite low scale. Even though the masses were small, in the case of DB3 the eigenvalues at lower resolutions [3,6] are complex. It can safely be assumed that at these resolutions the DB3 scaling functions fail to resolve the details of the kernel. In the DB2 case, we do not encounter this problem since the DB2 scaling function itself has a smaller support, 2 units less than that of DB3 when they are not scaled. The fit outputs are shown in Table 2. By looking at the $R^2$ values of the fits, it is clear that the fits explain the data quite well. The results from these fits are compared in Table 3.

**Table 3 - Comparison of the numerical and analytical coupling values for test 1.**
The 95% confidence interval of fit coefficient, $c$, is used in the calculation of the uncertainty in the result.

| Eigenvalue | Analytical Result | DB2 Extrapolated numerical result | Percentage difference | DB3 Extrapolated numerical result | Percentage difference |
|---|---|---|---|---|---|
| 1 | 6.579 | $6.45 \pm 0.07$ | 1.96% | $6.56 \pm 0.02$ | 0.29% |
| 2 | 5.394 | $5.42 \pm 0.03$ | 0.49% | $5.36 \pm 0.03$ | 0.63% |

At this point it may be useful to look at the computational time required for this calculation at different resolutions to get an idea of the efficiency of this numerical technique. Table 4 shows the computing time required for the construction of the kernel matrix in the DB2 basis followed by the time required for the eigenvalue problem for the dense kernel. This does not include the time required to generate the quadrature points and weights. The computing time is a rough

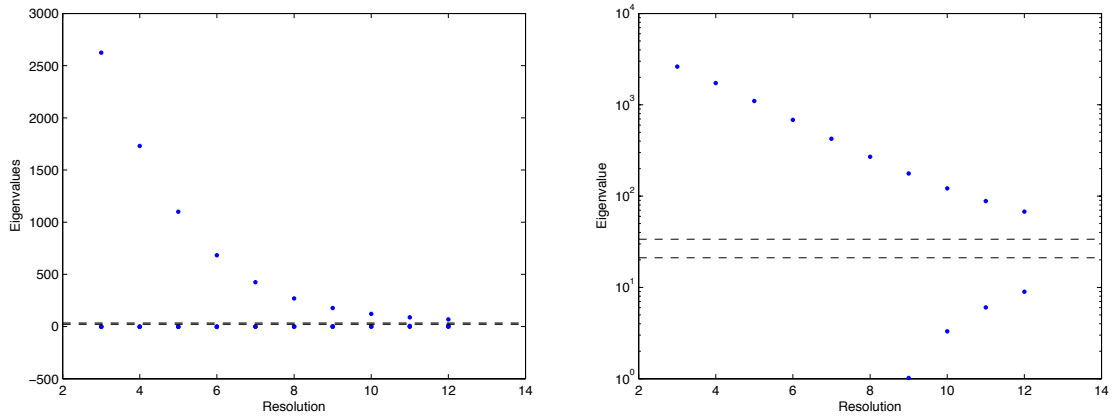
measurement, and there is also some computational overhead in retrieving the quadrature points from memory and storing the calculated kernel in memory, that obviously changes with the size of the matrix in question. In the first few resolutions the eigenvalue problem clearly takes less time compared to the function evaluations but, as the kernel size increases the eigenvalue problem takes more time than the function evaluations. One objective of the filtering phase of the study is decreasing the eigenvalue problem time by taking advantage of the sparsity of the kernel after filtering.

**Table 4 - The Computing time for the DB2 calculation at different scales.**

| Scaling function scale | Number of function evaluations | Time required for the function evaluations (s) | Kernel matrix size | Time required for the eigenvalue problem (s) |
|---|---|---|---|---|
| $-3$ | 000000099 | 00000.25 | $10\times10$ | 00000.00 |
| $-4$ | 000000323 | 00000.43 | $18\times18$ | 00000.00 |
| $-5$ | 000001155 | 00001.13 | $34\times34$ | 00000.02 |
| $-6$ | 000004355 | 00003.56 | $66\times66$ | 00000.07 |
| $-7$ | 000016899 | 00012.62 | $130\times130$ | 00000.52 |
| $-8$ | 000066563 | 00046.99 | $258\times258$ | 00005.26 |
| $-9$ | 000264195 | 00181.47 | $514\times514$ | 00040.03 |
| $-10$ | 001052675 | 00712.49 | $1026\times1026$ | 00406.82 |
| $-11$ | 004202499 | 02825.62 | $2050\times2050$ | 03222.65 |
| $-12$ | 016793603 | 11248.34 | $4098\times4098$ | 36175.77 |

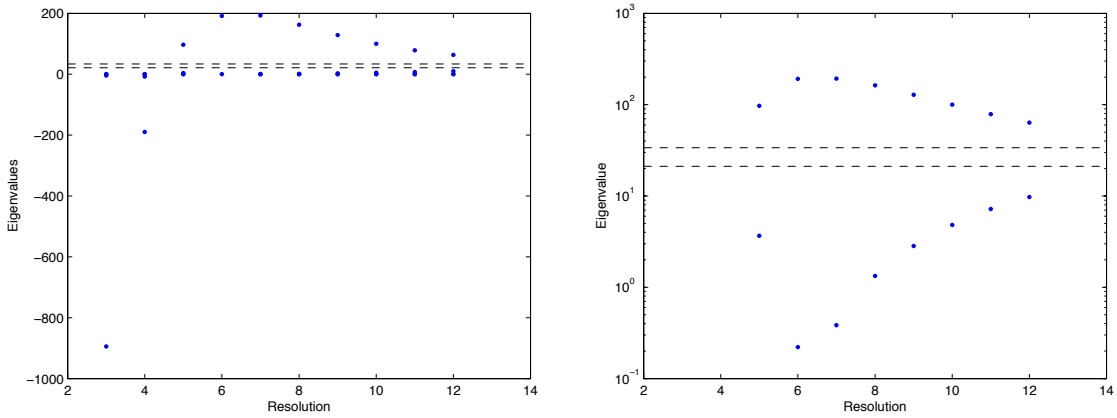
***Test 2***

The second set of masses used was selected so that the Pauli-Villars fermion mass is ten times that of the boson, i.e. $m_1 = 10\mu_1$ i.e. $\mu_1 = 200\mu_0$ and $m_1 = 2000\mu_0$ while $m_0 = -5\mu_0$. Figure 8 and Figure 9 show the convergence of the eigenvalues to the expected values. It is apparent that at these masses, the numerical results show a good convergence but the convergence is not as rapid as in test case 1. The eigenvalues are used to calculate the coupling parameter, $g$, just as in test case 1. Once coupling parameters are calculated an extrapolation is done using the same model as test 1. Figure 10 shows the results of these extrapolations.




a b

**Figure 8 - Real part of eigenvalues of test 2 in the DB2 scaling function basis at different scales.**
a) Same as Figure 5 but for mass parameters $M = \mu_0$, $\mu_1 = 200\mu_0$, $m_1 = 2000\mu_0$, $m_0 = -5\mu_0$.
b) Eigenvalues displayed on a semi-log scale for clarity (zero and negative eigenvalues are removed).




a b

**Figure 9 - Same as Figure 8 but for DB3 scaling function bases.**
a) The mass parameters are same as Figure 8.
b) Same as Figure 8 b), eigenvalues on semi-log scale.

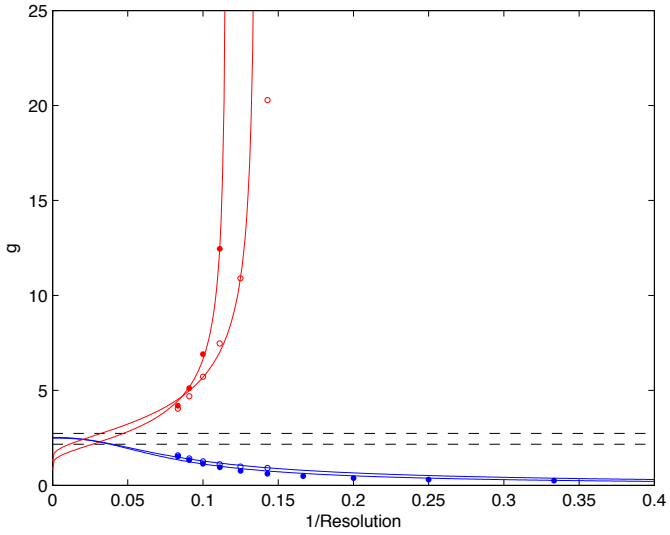



**Figure 10 - Calculated coupling of test 2 at different scales.**
As in Figure 7, the filled and open symbols show the DB2 and DB3 scaling function basis results.

**Table 5 - Fit output for test 2.**
The information shown is the same as Table 2 but for test 2.

| Eigenvalue | DB2 fit | DB3 fit |
|---|---|---|
| 1<br>(g=2.733) | a = 3.96e+04 (-3.053e+05, 3.845e+05)<br>b = 2.601 (-2.378, 7.58)<br>c = 25.58 (-235.2, 286.3)<br>Goodness of fit:<br>SSE: 0.1191<br>R-square: 0.9003<br>Adjusted R-square: 0.8604<br>RMSE: 0.1543 | a = 1.318e+04 (-1.585e+05, 1.849e+05)<br>b = 2.267 (-5.019, 9.554)<br>c = 24.83 (-230.7, 280.4)<br>Goodness of fit:<br>SSE: 0.01904<br>R-square: 0.9427<br>Adjusted R-square: 0.9046<br>RMSE: 0.07967 |
| 2<br>(g=2.165) | a = -296.4 (-7.642e+04, 7.583e+04)<br>b = 0.1038 (-34.87, 35.08)<br>c = 236.9 (-7.849e+04, 7.897e+04)<br>Goodness of fit:<br>SSE: 0.1439<br>R-square: 0.9965<br>Adjusted R-square: 0.9895<br>RMSE: 0.3793 | a = -141.4 (-6321, 6038)<br>b = 0.1603 (-10.69, 11.01)<br>c = 102.6 (-6611, 6816)<br>Goodness of fit:<br>SSE: 0.5491<br>R-square: 0.9819<br>Adjusted R-square: 0.9637<br>RMSE: 0.524 |

The fits do show significant uncertainties, due to the limited number of data points available for the extrapolation. Detailed fit parameters are given in Table 5. By the $R^2$ values of the fits, it is clear that the two curves that were suggested to approach the first eigenvalue were not good, for both DB2 and DB3 bases, hence the quite large uncertainties. Results from the fits are compared in Table 6. The uncertainties of the fits are so high that a comparison with the expected values will be meaningless. Using a scaling function basis of a very fine scale, i.e. higher resolution, results in a very large dense kernel matrix. Unless the kernel matrix is filtered, it will increase the resource requirement of the eigenvalue problem even more.

**Table 6 - Comparison of the numerical and analytical coupling values for test 2.**

| Eigenvalue | Analytical Result | DB2 Extrapolated numerical result | DB3 Extrapolated numerical result |
|---|---|---|---|
| 1 | 2.733 | $2.48 \pm 13$ | $2.52 \pm 13$ |
| 2 | 2.165 | $0.82 \pm 136$ | $1.24 \pm 41$ |

***Test 3***

The third set of masses used are same as test 2 except for the change of the fermion mass, $m_0 = -50\mu_0$. In test 2, at lower resolutions $J = [3,8]$, the eigenvalues were negative, but in this case, resolutions of only $J = [3,4]$ give negative eigenvalues, giving us more data points for the extrapolation, ultimately decreasing the extrapolations uncertainties given in Table 8.

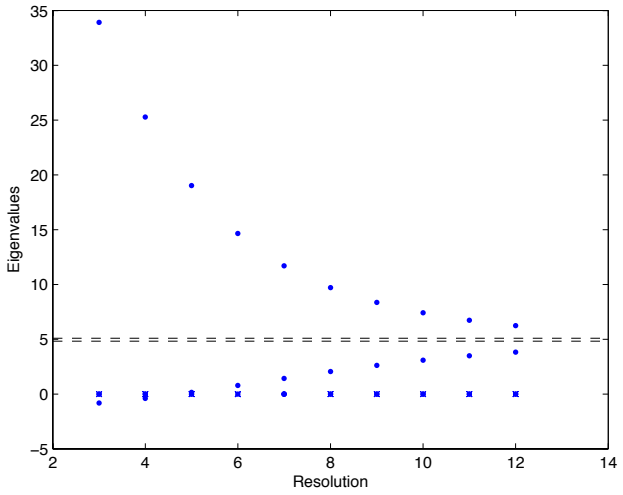



**Figure 11 - Real part of eigenvalues of test 3 in the DB2 scaling function basis at different scales.**
Same as Figure 5 but for mass parameters $M = \mu_0$, $\mu_1 = 200\mu_0$, $m_1 = 2000\mu_0$, $m_0 = -50\mu_0$.

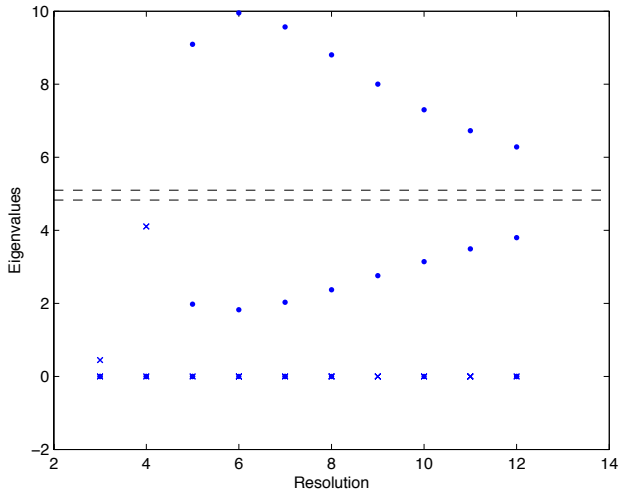



**Figure 12 - Same as Figure 11 but for DB3 scaling function bases.**
The masses used are same as Figure 11. The ×'s show the complex eigenvalues.

Once the eigenvalues are calculated, the coupling is calculated as usual and fitted to model 1. The fit output parameters are shown in Table 7 while the results of these fits are shown in Figure 13. Convergence of this test case proved to be better than that of case 2 giving us an idea of how $m_0$ would effect the calculation. Also the fits have less uncertainty than that of test 2 due to the availability of more data points to the extrapolation. Apparently the higher $m_0$ reduces the scale of the features of the kernel to levels that can be approximated by the scaling functions of considered scales, giving us results that are much more accurate than the smaller $m_0$ case test 2

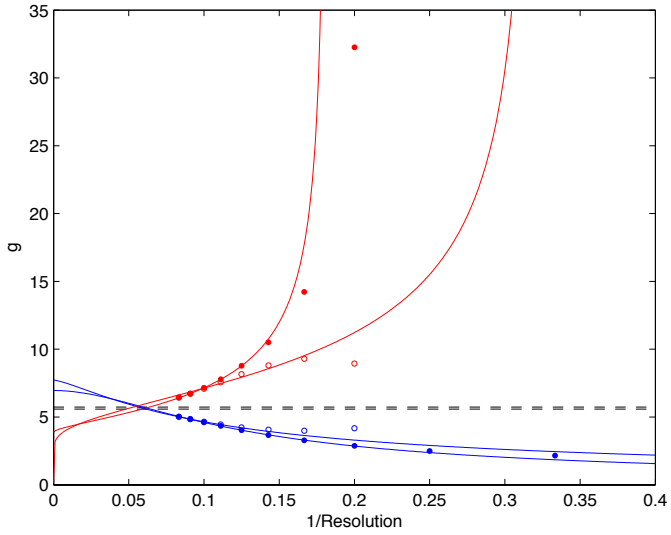



**Figure 13 - Calculated coupling of test 3 at different scales.**
As in Figure 7, the filled and open symbols show the DB2 and DB3 scaling function basis results.

**Table 7 - Fit output for test 3.**
The information shown is the same as Table 2 but for test 3.

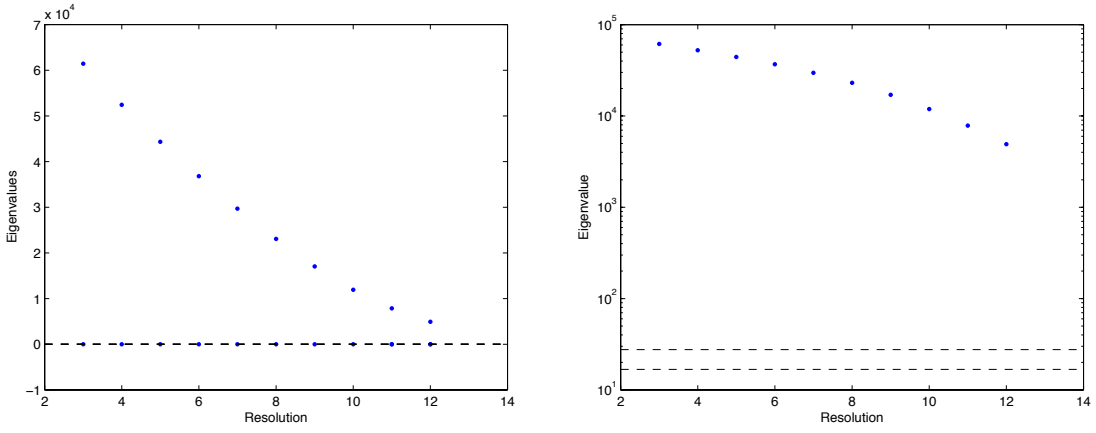
| Eigenvalue | DB2 fit | DB3 fit |
|---|---|---|
| 1<br>(g=5.734) | a = 357.1 (260, 454.3)<br>b = 1.93 (1.746, 2.114)<br>c = 3.263 (2.616, 3.91)<br>Goodness of fit:<br>SSE: 0.004081<br>R-square: 0.9994<br>Adjusted R-square: 0.9991<br>RMSE: 0.02608 | a = 104.2 (-261.3, 469.7)<br>b = 1.351 (-0.8428, 3.544)<br>c = 2.646 (-4.481, 9.773)<br>Goodness of fit:<br>SSE: 9.931e-06<br>R-square: 0.9999<br>Adjusted R-square: 0.9998<br>RMSE: 0.003151 |
| 2<br>(g=5.580) | a = -28.53 (-35.97, -21.1)<br>b = 0.5876 (0.1077, 1.068)<br>c = 10.46 (4.234, 16.68)<br>Goodness of fit:<br>SSE: 0.0002742<br>R-square: 0.9999<br>Adjusted R-square: 0.9998<br>RMSE: 0.01171 | a = -6264 (-3.616e+04, 2.363e+04)<br>b = 0.000427 (-0.002093, 0.002947)<br>c = 6261 (-2.364e+04, 3.616e+04)<br>Goodness of fit:<br>SSE: 1.246<br>R-square: 0.8482<br>Adjusted R-square: 0.7874<br>RMSE: 0.4992 |

**Table 8 - Comparison of the numerical and analytical coupling values for test 3.**

| Eigenvalue | Analytical Result | DB2 Extrapolated numerical result | Percentage difference | DB3 Extrapolated numerical result | Percentage difference |
|---|---|---|---|---|---|
| 1 | 5.734 | $6.96 \pm 0.69$ | 21% | $7.73 \pm 10$ | 35% |
| 2 | 5.580 | $3.89 \pm 1.1$ | 30% | $0.16 \pm 0.38$ | 97% |

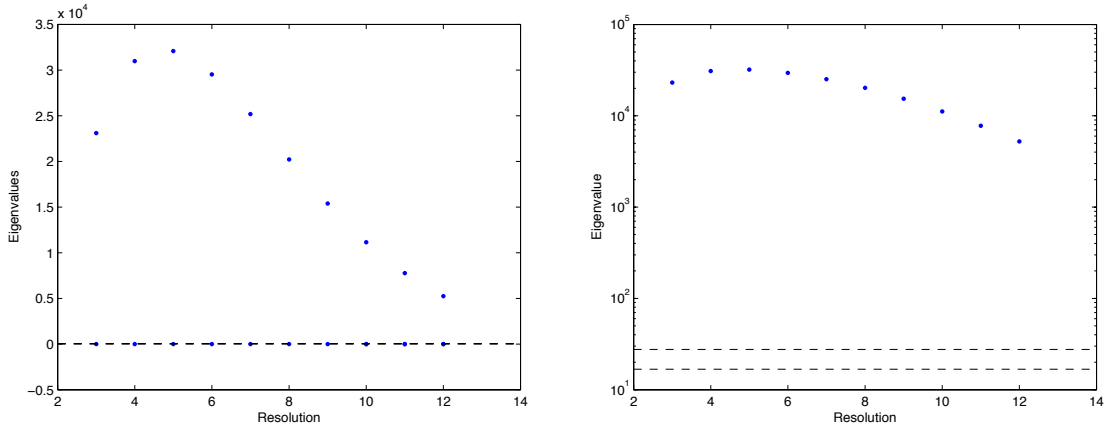
***Test 4***

The final set of masses used was selected so that the Pauli-Villars fermion mass is one hundred times that of the boson, i.e. $m_1 = 100\mu_1$ $\mu_1 = 500\mu_0$, $m_1 = 50000\mu_0$ while $m_0 = -5\mu_0$. Figure 14 and Figure 15 show the results for these mass parameters.




a b

**Figure 14 - Real part of eigenvalues of test 3 in the DB2 scaling function basis at different scales.**
a) Same as Figure 8 but for masses $M = \mu_0, \mu_1 = 500\mu_0, m_1 = 50000\mu_0, m_0 = -5\mu_0$
b) Eigenvalues displayed on a semi-log scale for clarity (zero and negative eigenvalues are removed).




a b

**Figure 15 - Same as Figure 14 but for DB3 scaling function bases.**
a) The masses used are same as Figure 14
b) Same as Figure 14 b), eigenvalues on semi-log scale.

The convergence of the results is slower than the previous test cases. In these calculations our numerical method did not return 2 positive eigenvalues. This suggests that at the given mass parameters the kernel has structure on a fine scale that the used scaling functions could not resolve i.e. the kernel matrix is severely under-sampled. Again the actual coupling parameter $g$ is calculated from the eigenvalues. The result is shown in Figure 16. The attempt to extrapolate the numerical results did not give us results that converge due to the limited number of resolutions used.

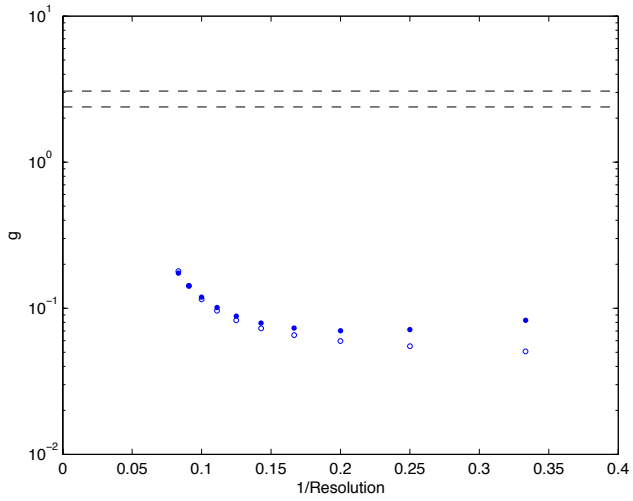



**Figure 16 - Calculated coupling of test 3 at different scales.**
As in Figure 7 the filled and open symbols show the DB2 and DB3 scaling function basis results.

**Filtering**

Once the dense kernel matrix in the scaling function basis at a specified scale was obtained, the wavelet transformation process followed by a filtering makes it considerably sparse. Figure 17 shows one of the nine parts of the kernel unfiltered and filtered at 3 different thresholds. The white (zero) bands in Figure 17 are due to the zero padding of the original matrix. The zero padding was done because the wavelet transform requires that the input matrix to be a square matrix with the length of one dimension is a power of 2. Once filtered the resulting matrix is again diagonalized and the eigenvalues are used to calculate the coupling, $g$. These values are then compared with the original dense matrix results. We expect these results to improve the efficiency of the eigenvalue problem with the cost of accuracy; therefore it makes little sense to compare the filtered results to the analytical results.

Table 9 shows the new matrix dimensions with the typical time requirement for the filtering at different scales. The filtering process discussed in Chapter 2 is somewhat inefficient because in order to find the threshold value for the matrix, it has to be sorted. For larger matrices this can be a considerable overhead. Also we do not expect these timings to change for different thresholds.

**Table 9 - Time requirement for filtering the DB2 kernel (by 95%) at different scales.**

| Scaling function scale | Kernel matrix size | Time required for the filtering (s) |
|---|---|---|
| $-3$ | $16\times16$ | 0000.02 |
| $-4$ | $32\times32$ | 0000.04 |
| $-5$ | $64\times64$ | 0000.07 |
| $-6$ | $128\times128$ | 0000.20 |
| $-7$ | $256\times256$ | 0000.65 |
| $-8$ | $512\times512$ | 0002.73 |
| $-9$ | $1024\times1024$ | 0012.85 |
| $-10$ | $2048\times2048$ | 0096.36 |
| $-11$ | $4096\times4096$ | 0784.22 |

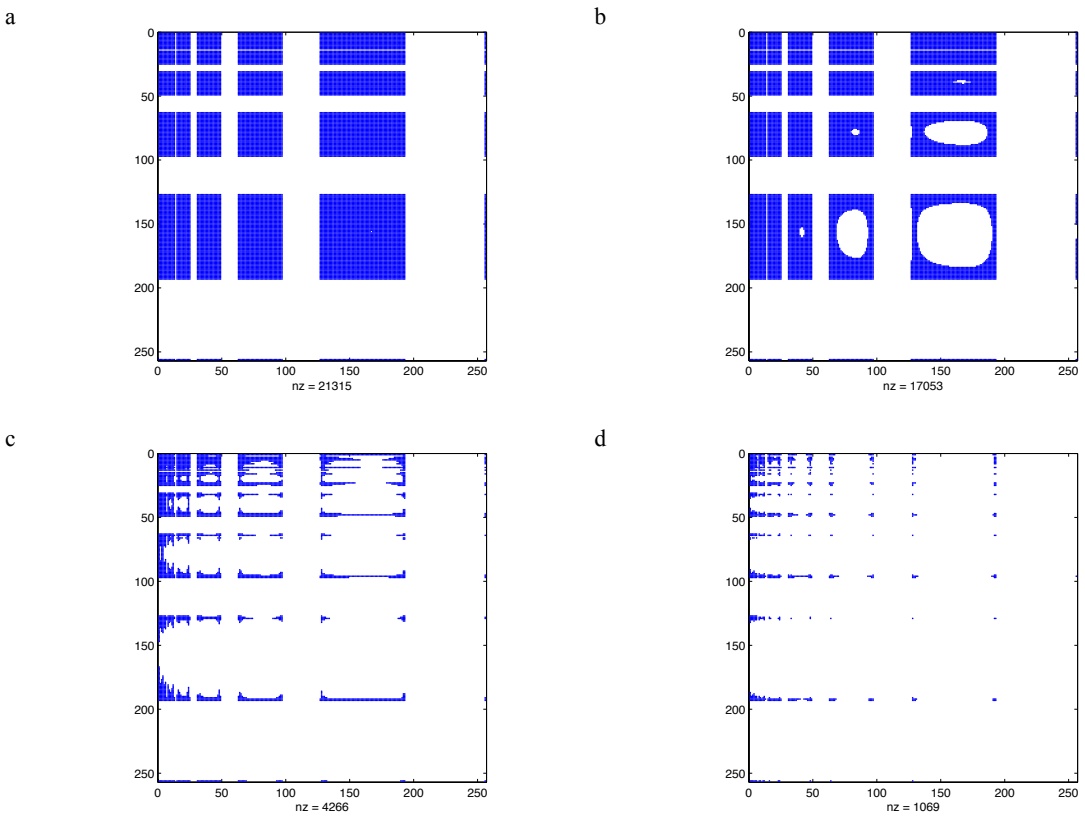



**Figure 17 - One of the nine parts of the DB2 kernel at scale $J = 5$, transformed and filtered.**
a) The transformed but unfiltered matrix. b) 20% is filtered out b) 80% is filtered out b) 95% is filtered out. The number of nonzero elements in each case is also shown below each plot.

***Test 1***

From Figure 18 and Figure 19 it is evident that for higher resolutions (in the test case 1, $J \geq 7$) even at 95% of the non-zero elements of the transformed kernel filtered out the eigenvalue problem returns reliable results.

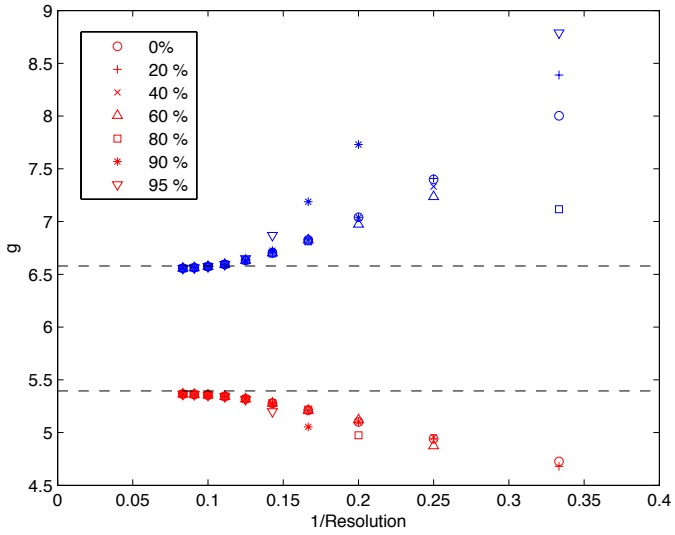



**Figure 18 - Coupling calculated using eigenvalues from kernel functions filtered at different levels for the DB2 basis for mass parameters of test case 1.**

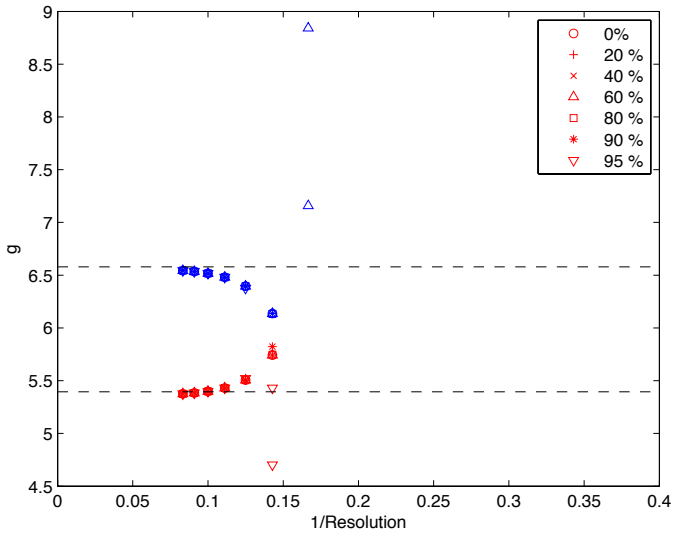



**Figure 19 - Same as Figure 18 for the DB3 basis at different scales.**

***Test 2***

Even though the test case 2 did not give us a rapid convergence as the case 1, we tested the effects of filtering on the eigenvalues. The Figure 20 and Figure 21 shows the filtering results for the test 2 kernel. Just as in test case 1 for higher resolutions (in the test case 2, $J \geq 10$) even at 95% of the non-zero elements of the transformed kernel filtered out the eigenvalue problem returned reliable results.

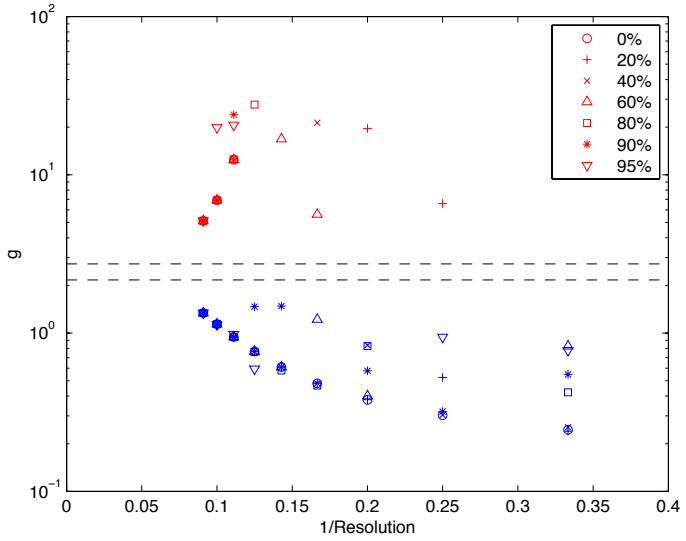



**Figure 20 - Same as Figure 18 coupling is calculated for mass parameters of test case 2.**

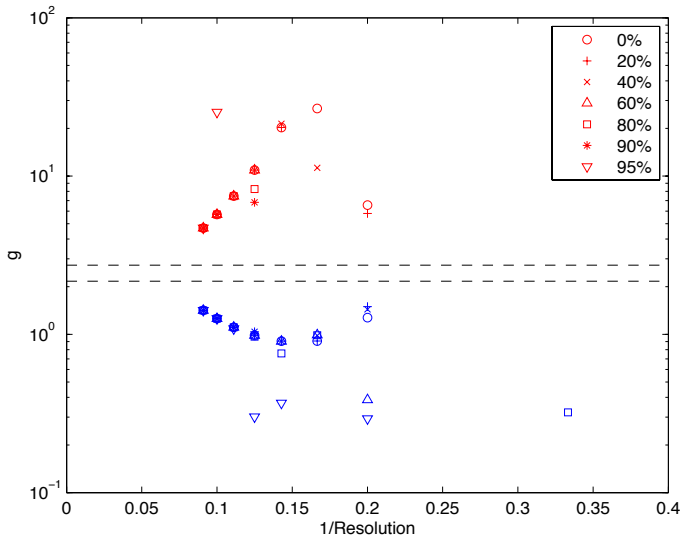



**Figure 21 - Same as Figure 20 for the DB3 basis at different scales.**

***Test 3***

Test case 3 was also tested to observe the effects of filtering. Figure 24 and Figure 25 shows the same behavior as the previous test cases; for finer scales (higher resolutions) the eigenvalues seem stable for higher percentages of filtering.

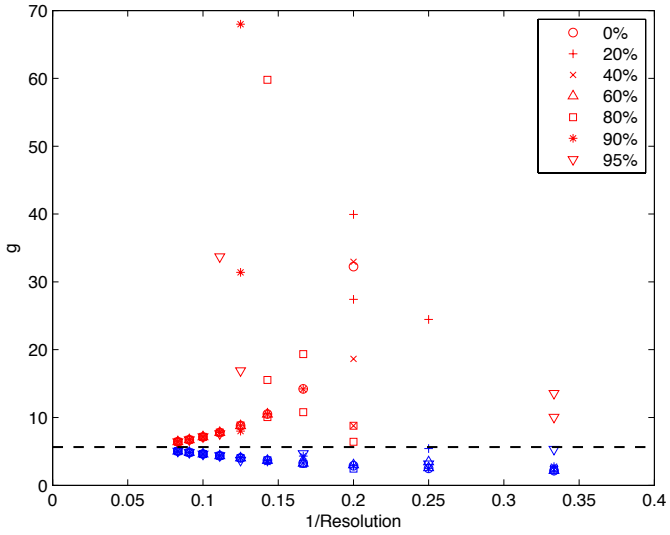



**Figure 22 - Same as Figure 18, Coupling is calculated for mass parameters of test case 2.**

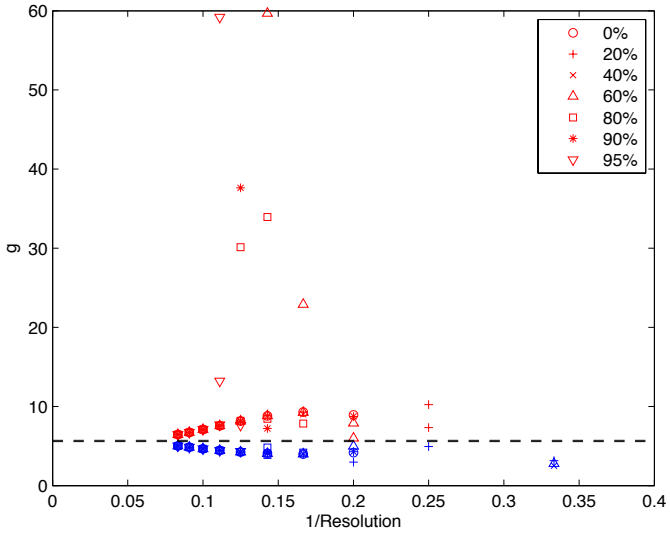



**Figure 23 - Same as Figure 20 for the DB3 basis at different scales.**

***Test 4***

Test case 4 was also tested to observe the effects of filtering. Figure 24 and Figure 25 shows the same behavior as the previous test cases; for finer scales (higher resolutions) the eigenvalues seem stable for higher percentages of filtering.

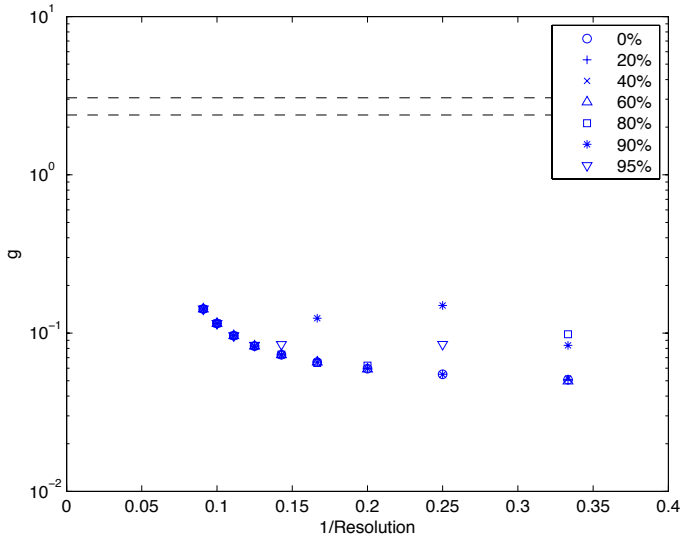



**Figure 24 - Same as Figure 18 but for mass parameters of test case 3.**

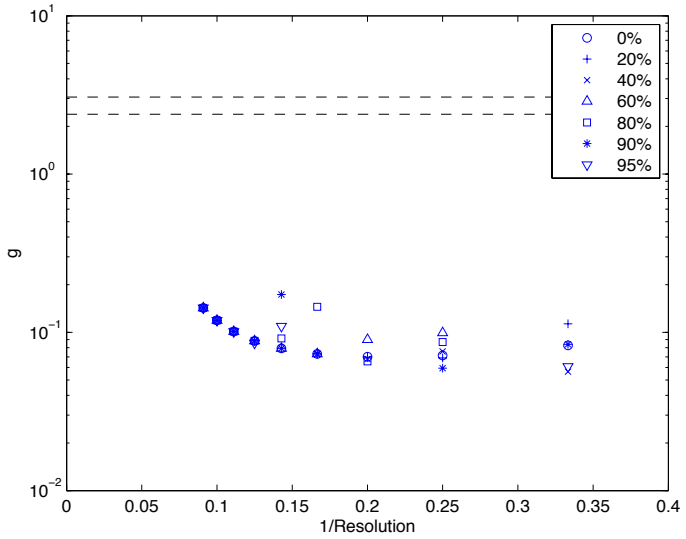



**Figure 25 – Same as Figure 24 but for DB3 basis at different scales.**

## Chapter 06 - Conclusion

From our tests, it was apparent that the wavelet basis was capable of discretizing the kernel of the integral equation accurately, but the scale of the Daubachies scaling functions used for the discretization is critical in getting good results. The higher the mass parameters, the finer the basis has to be to pick up the small-scale details of the kernel. This obviously increases the number of function evaluations; the actual kernel matrix dimension increases as $\approx 2^J$, where $J$ is the scale of the basis used.

By the filtered results of test cases 1, 2 and 3, we can see that if the kernel is originally discretized using a scaling function basis with a high enough resolution, even a 95% filtering will still produce reliable results. It is clear from the results section that the scale that qualifies as 'fine enough' changes with the mass parameters in question, since they will change the scale of the structure of the kernel. But the ability to force 95% of the matrix to be sparse while retaining the original structure is a fascinating property of the wavelet transform.

The original problem consists of two primary operations, discretizing the kernel, followed by solving the matrix eigenvalue problem. Our results suggest that in this first operation, discretizing at a high enough resolution (fine enough scale) is essential. Using a fine-scale basis comes with the cost of a higher number of function evaluations and larger kernel matrices, and a large kernel matrix gives an expensive eigenvalue problem. If the eigenvalue problem is preceded by the wavelet transform and a filtering step, it can improve the efficiency of the calculation by a significant amount. In conclusion we can suggest that the wavelet transform is a very powerful numerical analysis tool in solving integral equations with kernels of various structure.

## Bibliography

[1]. *Scattering Calculations with Wavelets.* **Kessler, B M, Payne, G L and Polyzou, W N.** 2003, Few Body Systems, Vol. 33, pp. 1-26.

[2]. *The mass renormalization of nonperturbative light-front Hamiltonian theory: An illustration using truncated, Pauli-Villars-regulated Yukawa interactions.* **Brodsky, Stanley J, Hiller, John R and McCartor, Gary.** 2003, Annals of Physics, Vol. 305, pp. 266-285.

[3]. *Quantum chromodynamics and other field theories on the light cone.* **Brodsky, Stanley J, Pauli, Hans -Christian and Pinsky, Stephen S.** 4–6, August 1998, Physics Reports, Vol. Volume 301, pp. 299–486.

[4]. **Daubechies, Ingrid.** *Ten Lectures on Wavelets.* Philadelphia : Society for Industrial and Applied Mathematics, 2004.

[5]. **Press, William H, et al.** *Numerical recipes : the art of scientific computing.* 3. New York : Cambridge University Press, 2007.

[6]. *Two-boson truncation of Pauli-Villars-regulated Yukawa theory.* **Brodsky, Stanley J, Hiller, John R and McCartor , Gary.** 2006, Annals of Physics, Vol. 321, pp. 1240-1264.

[7]. *Quadrature Formulae and Asymptotic Error Expansions for wavelet approximations of smooth functions.* **Sweldens, Wim and Piessens, Robert.** 1994, SIAM Journal on Numerical Analysis, Vol. 31, pp. 1240–1264.

[8]. **Kessler, B M, Payne, G L and Polyzou, W N.** *Wavelet Notes.* Iowa City : s.n., 2008.

[9]. *Application of Pauli–Villars regularization and discretized light-cone quantization to a single-fermion truncation of Yukawa theory.* **Brodsky, Stanley J, Hiller, John R and McCartor, Gary .** 2001, Physical Review D, Vol. 64, p. 114023.

[10]. *Matrices and Quadrature Rules for Wavelets.* **Shann, W C and Yen, C C.** 4, 1998, Taiwanese Journal of Mathematics, Vol. 2, pp. 435-446.

# Appendix A

**Unit translation operator**

The two primary operators that generate the scaling function and wavelet bases are the discrete translation operator, $T$ and dyadic scale operator, $D$. The properties of these functions are discussed in great detail in [4] and [8]. The properties and relationships that are used in this study are as follows.

The unit translation operator $T$ is defined by

$$Tf(x) = f(x-1) \tag{95}$$

Figure 26 demonstrates the translation operator $T$ translating the function $f(x)$ to the right by one unit.

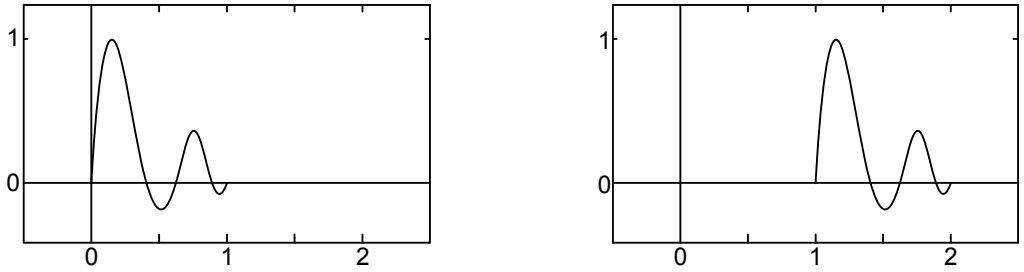



**Figure 26 - Discrete translation operator.**

Therefore denote

$$T^n f(x) = f(x-n) \tag{96}$$

The unit translation operator has the property:

$$\big(Tf(x), Tg(x)\big) = \int_{-\infty}^{\infty} f^*(x-1)g(x-1)\,dx$$

By taking $y = x - 1$

$$\big(Tf(x), Tg(x)\big) = \int_{-\infty}^{\infty} f^*(y)g(y)\,dy = (f,g)$$

This means that the unit translation operator preserves the scalar product

$$\big(Tf(x), Tg(x)\big) = (f,g) \tag{97}$$

If $A$ is a linear operator its adjoint $A^\dagger$ is defined by the relation

$$\big(Af(x), g(x)\big) = \big(f(x), A^\dagger g(x)\big)$$

It follows that

$$\big(Tf(x), g(x)\big) = \Big(f(x), T^{\dagger}g(x)\Big) = \int_{-\infty}^{\infty} f^{*}(x-1)g(x)dx \tag{98}$$

By taking $y = x - 1$

$$\Big(f(x), T^{\dagger}g(x)\Big) = \int_{-\infty}^{\infty} f^{*}(y)g(y+1)dy$$

or

$$T^{\dagger}g(x) = g(x+1)$$

which is a left shift by one unit. Since

$$\big(f(x), g(x)\big) = \big(Tf(x), Tg(x)\big) = \Big(f(x), T^{\dagger}Tg(x)\Big)$$

it follows that $T^{\dagger} = T^{-1}$. Therefore $T$ is unitary

**Discrete scale operator**

The linear operator $D$, corresponding to discrete scale transformations, is defined by:

$$Df(x) = \frac{1}{\sqrt{2}} f\left(\frac{x}{2}\right) \tag{99}$$

Figure 27 demonstrates the discrete scale operator $D$ stretching the function twice as wide as the original $f(x)$ and the magnitude decreased by a factor of $\sqrt{2}$.

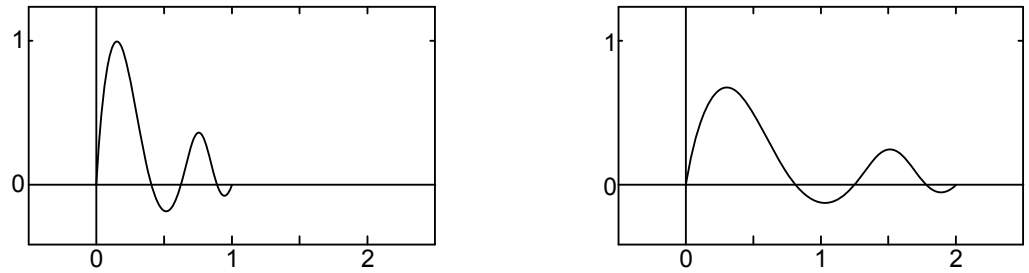



**Figure 27 - Discrete scale operator.**

Therefore denote

$$D^{m}f(x) = \frac{1}{\sqrt{2}^{m}} f\left(\frac{x}{2^{m}}\right) = 2^{-\frac{m}{2}} f(2^{-m}x)$$

Note that the normalization ensures

$$\big(Df(x), Dg(x)\big) = \int_{-\infty}^{\infty} \frac{1}{2} f^{*}\left(\frac{x}{2}\right) g\left(\frac{x}{2}\right) dx$$

By taking $y = \frac{x}{2}$

$$\big(Df(x), Dg(x)\big) = \int_{-\infty}^{\infty} \frac{1}{2} f^{*}(y)g(y)\, 2dy = \int_{-\infty}^{\infty} f^{*}(y)g(y)\, dy = (f, g)$$

This means that the unit scale operator preserves the scalar product

$$\big(Df(x), Dg(x)\big) = (f, g) \tag{100}$$

The adjoint of $D$ is determined by the definition

$$\big(Df(x), g(x)\big) = \Big(f(x), D^{\dagger}g(x)\Big) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2}} f^{*}\left(\frac{x}{2}\right) g(x)dx \tag{101}$$

By taking $y = \frac{x}{2}$

$$\Big(f(x), D^{\dagger}g(x)\Big) = \int_{-\infty}^{\infty} f^{*}(y)\sqrt{2}g(2y)dx$$

or

$$D^{\dagger}g(x) = \sqrt{2}g(2x)$$

which shows that $D^{\dagger} = D^{-1}$. Therefore $D$ is also unitary. Additional relationships involving the translation $T$ and dilation $D$ operators are useful for further computations

$$DTf(x) = Df(x-1) = \frac{1}{\sqrt{2}} f\left(\frac{x}{2} - 1\right) = \frac{1}{\sqrt{2}} f\left(\frac{x-2}{2}\right) = T^{2}Df(x)$$

$$DT = T^{2}D \tag{102}$$

Another important relationship

$$TD^{-1}f(x) = T\sqrt{2}f(2x) = \sqrt{2}f(2x-2) = D^{-1}T^{2}f(x)$$

$$TD^{-1} = D^{-1}T^{2} \tag{103}$$

used on

$$\phi_{jk}(x) = D^{j}T^{k}\phi(x) = 2^{-\frac{j}{2}} f\left(2^{-j}(x-k)\right) \tag{104}$$

gives the following

$$\phi_{jk}(x) = D^{j}T^{k}D^{-1}\sum_{r=0}^{2K-1} h_r T^{r}\phi(x) = \sum_{r=0}^{2K-1} h_r D^{j}T^{k}D^{-1}T^{r}\phi(x)$$

and using (103)

$$\phi_{jk}(x) = \sum_{r=0}^{2K-1} h_r D^{j-1}T^{2k}T^{r}\phi(x) = \sum_{r=0}^{2K-1} h_r D^{j-1}T^{2k+r}\phi(x)$$

$$\phi_{jk}(x) = \sum_{r=0}^{2K-1} h_r \phi_{(j-1)(2k+r)}(x) \tag{105}$$

For the case $j = 0$ and $k = 0$

$$\phi(x) = \sum_{r=0}^{2K-1} h_r \phi_{(-1)(2+r)}(x) = \sum_{r=0}^{2K-1} \sqrt{2}h_r \phi(2x-r) \tag{106}$$

The relationship (105) can be written for the wavelet

$$\psi_{jk}(x) = \sum_{r=0}^{2K-1} h_r \psi_{(j-1)(2k+r)}(x) \tag{107}$$

Another important relationship uses the adjoint property of the $D$ and $T$ operators

$$\left(f(x), D^j T^k g(x)\right) = \left(T^{-k} D^{-j} f(x), g(x)\right) \tag{108}$$

# Appendix B

**Moments and Quadrature Rules**

For the wavelet numerical analysis methods the following values are required

- Full moments of scaling functions
  - Any order $m$ for (scale-0 translation-0)

$$\int x^m \phi_{00}(x)dx = \langle x^m \rangle_{\phi} \tag{109}$$

  - Any order $m$ for (any scale-$k$ and any translation-$l$):

$$\int x^m \phi_{kl}(x)dx = \langle x^m \rangle_{\phi_{kl}} \tag{110}$$

- Partial moments of scaling functions
  - Zero order ($l \in \mathbb{Z}$, $0 < l < 2n-1$):

$$\int_0^l \phi_{00}(x)dx = \langle x^0 \rangle_{\phi[0,l]} \tag{111}$$

  - Zero order ($l \in \mathbb{Z}$, $0 < l < 2n-1$): $\langle x^0 \rangle_{\phi[l,2n-l]}$ (Complementary partial moment)

$$\int_l^{2n-l} \phi_{00}(x)dx = \langle x^0 \rangle_{\phi[l,2n-l]} \tag{112}$$

  - Any order $m$ ($l \in \mathbb{Z}$, $0 < l < 2n-1$): $\langle x^m \rangle_{\phi[0,l]}$

$$\int_0^l x^m \phi_{00}(x)dx = \langle x^{\mathrm{m}} \rangle_{\phi[0,l]} \tag{113}$$

  - Any order $m$ ($l \in \mathbb{Z}$, $0 < l < 2n-1$): $\langle x^m \rangle_{\phi[n,2n-1]}$ (Complementary partial moment)

$$\int_0^l x^m \phi_{00}(x)dx = \langle x^0 \rangle_{\phi[n,2n-1]} \tag{114}$$

  - Any order $m$ (scale-0 translation-$k$ and $k \in \mathbb{Z}$, $-(2n-1) < k < 0$): $\langle x^m \rangle_{\phi_{0k}[0,\infty)}$

$$\int_0^{\infty} x^m \phi_{0k}(x)dx = \langle x^m \rangle_{\phi_{0k}[0,\infty)} \tag{115}$$

  - Any order $m$ (scale-0 translation-$k$ and $k \in \mathbb{Z}$, $-(2n-1) < k < 0$): $\langle x^m \rangle_{\phi_{0k}(-\infty,0]}$ (Complementary partial moment)

$$\int_{-\infty}^0 x^m \phi_{0k}(x)dx = \langle x^m \rangle_{\phi_{0k}(-\infty,0]} \tag{116}$$

- Any order $m$ (scale-$j$ translation-$k$ and $k \in \mathbb{Z}$, $-(2n-1) < k < 0$): $\langle x^m \rangle_{\phi_{jk}[0,\infty)}$

$$\int_0^\infty x^m \phi_{jk}(x) dx = \langle x^m \rangle_{\phi_{jk}[0,\infty)} \tag{117}$$

- Any order $m$ (scale-$j$ translation-$k$ and $k \in \mathbb{Z}$, $-(2n-1) < k < 0$): $\langle x^m \rangle_{\phi_{jk}(-\infty,0]}$ (Complementary partial moment)

$$\int_{-\infty}^0 x^m \phi_{jk}(x) dx = \langle x^m \rangle_{\phi_{jk}(-\infty,0]} \tag{118}$$

- Any order $m$ (scale-$j$ translation-$k$ and $l$ may not be an integer): $\langle x^m \rangle_{\phi_{jk}[l,\infty)}$

$$\int_l^\infty x^m \phi_{jk}(x) dx = \langle x^m \rangle_{\phi_{jk}[l,\infty)} \tag{119}$$

- Any order $m$ (scale-$j$ translation-$k$ and $l$ may not be an integer): $\langle x^m \rangle_{\phi_{jk}(-\infty,l]}$ (Complementary partial moment)

$$\int_{-\infty}^l x^m \phi_{jk}(x) dx = \langle x^m \rangle_{\phi_{jk}(-\infty,l]} \tag{120}$$

- Singular moments
  - Order $-1$ for (scale-0 translation-$k$ and $k \in \mathbb{Z}$, $-(2n-1) < k < 0$): $\langle x^{-1} \rangle_{\phi_{0k}}$

$$\int x^{-1} \phi_{0k}(x) dx = \langle x^m \rangle_{\phi_{0k}} \tag{121}$$

  - Order $-1$ for (scale-$j$ translation-$k$ and $k \in \mathbb{Z}$, $-(2n-1) < k < 0$): $\langle x^{-1} \rangle_{\phi_{jk}}$

$$\int x^{-1} \phi_{0k}(x) dx = \langle x^m \rangle_{\phi_{jk}} \tag{122}$$

Calculating the scaling function, $\phi$ (the basis function) itself is computationally intensive due to its fractal nature. But their moments are calculated using the scaling coefficients and manipulating the scaling equation. The scaling function is never calculated explicitly, making this technique highly efficient.

**Moments**

***Full-moment of any order of un-scaled and un-translated scaling functions***

$$\langle x^m \rangle_\phi = \big(x^m, \phi(x)\big) = \int_{-\infty}^{\infty} x^m \phi(x)\, dx$$

The moments can be constructed from the normalization condition

$$\langle x^0 \rangle_\phi = \big(x^0, \phi(x)\big) = \int_{-\infty}^{\infty} \phi(x)\, dx = 1$$

In order to calculate moments of any order $\langle x^m \rangle_\phi$ we write

$$\langle x^m \rangle_\phi = \big(x^m, \phi(x)\big) = \big(Dx^m, D\phi(x)\big)$$

Using the scaling equations (107) and (99)

$$\langle x^m \rangle_\phi = \left( \frac{1}{\sqrt{2}} \left( \frac{x}{2} \right)^m, \sum_{p=0}^{2K-1} h_p T^p \phi(x) \right) = \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{p=0}^{2K-1} h_p \left( x^m, T^p \phi(x) \right) \tag{123}$$

where the adjoint property (109) of the operator $T$ is used to write

$$\langle x^m \rangle_\phi = \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{p=0}^{2K-1} h_p \left( T^{-p} x^m, \phi(x) \right) = \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{p=0}^{2K-1} h_p \left( (x+p)^m, \phi(x) \right) \tag{124}$$

The expansion of $(x+p)^m$ gives

$$\langle x^m \rangle_\phi = \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{p=0}^{2K-1} h_p \left( \sum_{k=0}^{m} \binom{m}{k} p^{m-k} x^k, \phi(x) \right) \tag{125}$$

where the $x^m$ term of the sum is brought over to the left side to get

$$\langle x^m \rangle_\phi - \frac{1}{2^m} \langle x^m \rangle_\phi = \frac{1}{\sqrt{2}} \frac{1}{2^m} \sum_{p=0}^{2K-1} h_p \sum_{q=0}^{m-1} \binom{m}{q} p^{m-q} \langle x^q \rangle_\phi \tag{126}$$

To get the following final relationship

$$\langle x^m \rangle_\phi = \left( \frac{1}{2^m - 1} \right) \frac{1}{\sqrt{2}} \sum_{p=0}^{2K-1} h_p \sum_{q=0}^{m-1} \binom{m}{q} p^{m-q} \langle x^q \rangle_\phi \tag{127}$$

Starting with $\langle x^0 \rangle_\phi = 1$, this recursive relation can be used to calculate $m$-order moment of the scaling function $\phi$, $\langle x^m \rangle_\phi$.

*Application*

---

*Example 1*

For the case $m = 1$

$$\langle x \rangle_\phi = \left( \frac{1}{2-1} \right) \frac{1}{\sqrt{2}} \sum_{p=0}^{2K-1} h_p \sum_{q=0}^{0} \binom{0}{q} p^{-q} \langle x^q \rangle_\phi = \frac{1}{\sqrt{2}} \sum_{p=0}^{2K-1} p h_p$$

The first order moments are used to calculate the higher order moments.

---

***Full-moment of any order of scaled and translated scaling functions***

The $m$-order moment of the unscaled and untranslated scaling function $\phi$, $\langle x^m \rangle_\phi$ can be generalized to give moments of scaling function $\phi_{jk}$ as follows

$$\langle x^m \rangle_{\phi_{jk}} = \left( x^m, D^j T^k \phi(x) \right) = \left( D^{-j} x^m, T^k \phi(x) \right) = \left( T^{-k} D^{-j} x^m, \phi(x) \right) \tag{128}$$

Once the operators act on $x^m$

$$\langle x^m \rangle_{\phi_{jk}} = \left( \frac{1}{\sqrt{2}^{-j}} \left( \frac{x+k}{2^{-j}} \right)^m, \phi(x) \right) = \frac{1}{2^{-\frac{j}{2}}} \frac{1}{2^{-jm}} \left( (x+k)^m, \phi(x) \right) \tag{129}$$

The expansion of the power gives

$$\langle x^m \rangle_{\phi_{jk}} = \frac{1}{2^{-j\left(m+\frac{1}{2}\right)}} \sum_{p=0}^{m} \binom{m}{p} k^{m-p} \langle x^p \rangle_{\phi} \tag{130}$$

Using this recursive relation $m$-order moment of the scaling function $\phi_{jk}$, $\langle x^m \rangle_{\phi_{jk}}$ can be calculated using the scaling function moments of order 0 to $m$ ($\langle x^0 \rangle_{\phi}$ to $\langle x^m \rangle_{\phi}$).

## Partial Moments

***Zeroth order partial moments of un-scaled and un-translated scaling functions***

A wavelet with a support $[0,2K-1]$ will have $2K-2$ partial moments, and these, $\langle x^0 \rangle_{\phi[0,a]}$ where *($a \in \mathbb{Z}$, $0 < a < 2K-1$)* calculated as follows.

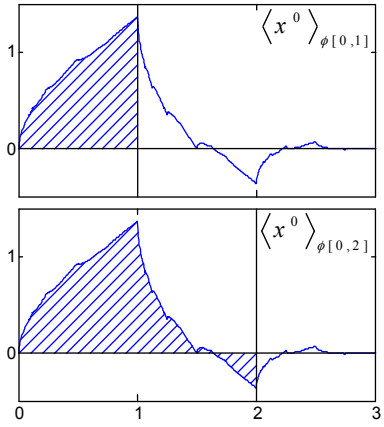



**Figure 28 - The DB2 Scaling functions parts that yield the 2 Partial moments.**
The shaded region indicates the part that contributes to the integrals, $\langle x^m \rangle_{\phi[0,2]}$ and $\langle x^m \rangle_{\phi[0,1]}$.

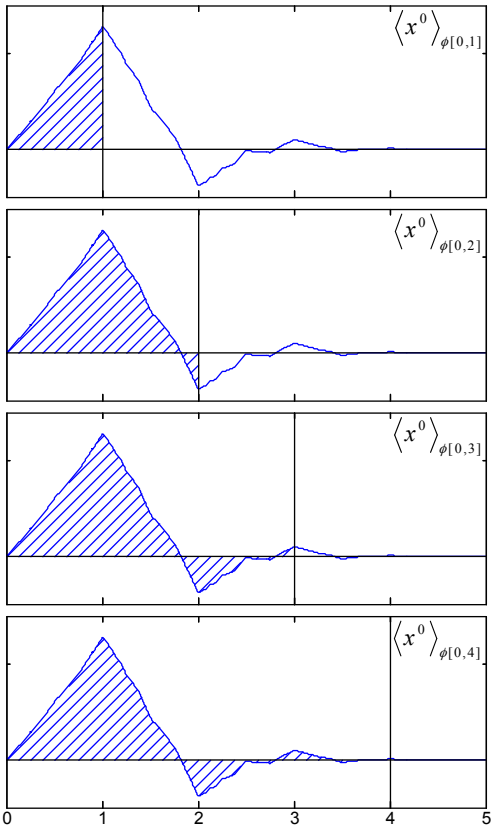



**Figure 29 - The DB3 Scaling functions parts that yield the 4 Partial moments.**
Same as Figure 28 but for the DB3 scaling function.

The partial moments for DB2 (Figure 28) and DB3 (Figure 29) wavelets

$$\langle x^m \rangle_{\phi[0,l]} = \int_0^l x^m \phi(x)\, dx \tag{131}$$

where $l \in \{1, \dots, 2K-2\}$. First consider the order 0 partial moments. Using the relationship (107)

$$\langle x^0 \rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \sqrt{2} h_p \int_0^l \phi(2x-p) dx$$

By taking $y = 2x - 1$

$$\langle x^0 \rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \frac{h_l}{\sqrt{2}} \int_{-p}^{2l-p} \phi(y) dy = \sum_{p=0}^{2K-1} \frac{h_p}{\sqrt{2}} \langle x^0 \rangle_{\phi[-p,2l-p]}$$

Since the support of the scaling function is $[0,2K-1]$ the negative lower limit can be taken as 0.

$$\langle x^0 \rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \frac{h_p}{\sqrt{2}} \langle x^0 \rangle_{\phi[0,2l-p]}$$

By taking $\frac{h_p}{\sqrt{2}} = c_p$ and $2l - p = q$

$$\langle x^0 \rangle_{\phi[0,l]} = \sum_{q=2l-2K+1}^{2l} c_{2l-q} \langle x^0 \rangle_{\phi[0,q]}$$

For a given $l$ the equation can be expanded

$$\langle x^0 \rangle_{\phi[0,l]} = c_{2K-1} \langle x^0 \rangle_{\phi[0,2l-2K+1]} + c_{2K-2} \langle x^0 \rangle_{\phi[0,2l-2K+2]} + \cdots + c_l \langle x^0 \rangle_{\phi[0,l]} + \cdots + c_1 \langle x^0 \rangle_{\phi[0,2l-1]} + c_0 \langle x^0 \rangle_{\phi[0,2l]}$$

when the $c_l \langle x^0 \rangle_{\phi[0,l]}$ term is brought to the left hand side

$$(1 - c_l) \langle x^0 \rangle_{\phi[0,l]} = c_{2K-1} \langle x^0 \rangle_{\phi[0,2l-2K+1]} + c_{2K-2} \langle x^0 \rangle_{\phi[0,2l-2K+2]} + \cdots + c_1 \langle x^0 \rangle_{\phi[0,2l-1]} + c_0 \langle x^0 \rangle_{\phi[0,2l]}$$

Once written for all $l$ these equations are a linear system of the non-trivial partial moments and take the form

$$M_{lq} m_q = v_l$$

where $l$ and $q \in [1,2K-2]$, $m_q$ is the partial moments vector and $M_{lq}$ is of the following form

$$M_{lq} = \delta_{lq} - C_{lq}$$

with

$$C_{lq} = c_{2l-q}, \quad l < K-1, q = 1, \dots, 2l$$
$$C_{lq} = 0, \quad l < K-1, q = 2l+1, \dots, 2K-2$$
$$C_{lq} = c_{2l-q}, \quad l = K-1, K$$
$$C_{lq} = c_{2l-q}, \quad l > K, q = 2l-2K+1, \dots, 2K-2$$
$$C_{lq} = 0, \quad l > K, q = 1, \dots, 2l-2K$$

The vector $v_l$ is of the following form

$$v_l = 0, \quad n < K$$
$$v_l = \sum_{p=0}^{2(q-K)+1} c_p, \quad K \le n \le 2K-2$$

The matrix $M_{lq} = I - C_{lq}$ can be written as follows

$$I - C_{lq} = \begin{pmatrix} 1-c_1 & c_0 & 0 & 0 & \cdots & 0 \\ -c_3 & 1-c_2 & -c_1 & -c_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ -c_{2K-3} & -c_{2K-4} & \cdots & \cdots & \ddots & -c_0 \\ -c_{2K-1} & -c_{2K-2} & \cdots & \cdots & \cdots & -c_2 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & -c_{2K-1} & 1-c_{2K-2} \end{pmatrix}_{(2K-2)\times(2K-2)} \tag{132}$$

The vector $m_q$ of partial moments can be written as follows

$$m_q = \begin{pmatrix} \langle x^0 \rangle_{\phi[0,1]} \\ \langle x^0 \rangle_{\phi[0,2]} \\ \vdots \\ \langle x^0 \rangle_{\phi[0,K-1]} \\ \langle x^0 \rangle_{\phi[0,K]} \\ \vdots \\ \langle x^0 \rangle_{\phi[0,2K-2]} \end{pmatrix}_{(2K-2)\times 1}$$

and the vector $v_l$ can be written as follows

$$v_l = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ c_0 + c_1 \\ \vdots \\ c_0 + \dots + c_{2K-3} \end{pmatrix}_{(2K-2)\times 1}$$

The matrix $M_{lq}$ is inverted to obtain the partial moments of order zero.

*Case 1*

For the case $K = 2$ (Daubachies 2 scaling function)

$$\begin{pmatrix} 1-c_1 & -c_0 \\ -c_3 & 1-c_2 \end{pmatrix}\begin{pmatrix} \langle x^0\rangle_{\phi[0,1]} \\ \langle x^0\rangle_{\phi[0,2]} \end{pmatrix} = \begin{pmatrix} 0 \\ c_0 + c_1 \end{pmatrix}$$

*Case 2*

For the case $K = 3$ (Daubachies 3 scaling function)

$$\begin{pmatrix} 1-c_1 & -c_0 & 0 & 0 \\ -c_3 & 1-c_2 & -c_1 & -c_0 \\ -c_5 & -c_4 & 1-c_3 & -c_2 \\ 0 & 0 & -c_5 & 1-c_4 \end{pmatrix}\begin{pmatrix} \langle x^0\rangle_{\phi[0,1]} \\ \langle x^0\rangle_{\phi[0,2]} \\ \langle x^0\rangle_{\phi[0,3]} \\ \langle x^0\rangle_{\phi[0,4]} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c_0 + c_1 \\ c_0 + c_1 + c_2 + c_3 \end{pmatrix}$$

***Complementary zeroth order partial moments of un-scaled and un-translated scaling functions***

Complementary partial moments, $\langle x^0\rangle_{\phi[l,2n-l]}$ where *($l \in \mathbb{Z},\ 0 < l < 2n-1$)*, can be calculated by

$$\langle x^0\rangle_{\phi[l,2K-1]} = \langle x^0\rangle_{\phi} - \langle x^0\rangle_{\phi[0,l]} = 1 - \langle x^0\rangle_{\phi[0,l]} \tag{133}$$

***Partial moments of any order of un-scaled and un-translated scaling functions***

Higher order partial moments, $\langle x^m\rangle_{\phi[0,l]}$ where *($l \in \mathbb{Z},\ 0 < l < 2n-1$),* can be constructed similarly

$$\langle x^m\rangle_{\phi[0,l]} = \int_0^l x^m \phi(x)\,dx = \sum_{p=0}^{2K-1} \sqrt{2}h_p \int_0^l \phi(2x-p)x^m dx$$

By taking $y = 2x - p$

$$\langle x^m\rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \sqrt{2}h_p \int_{-p}^{2l-p} \phi(y)\left(\frac{(y+p)}{2}\right)^m \frac{dy}{2}$$

$$= \sum_{p=0}^{2K-1} \frac{h_p}{2^m\sqrt{2}} \int_{-p}^{2l-p} \phi(y)(y+p)^m dy$$

Expanding $(y+p)^m$

$$\langle x^m\rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \frac{h_p}{2^m\sqrt{2}} \int_{-p}^{2l-p} \phi(y) \sum_{q=0}^{m} \binom{m}{q} p^{m-q} y^q\, dy$$

Since the support of the scaling function is greater than 0 negative lower limit can be taken as 0.

$$\langle x^m \rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \frac{h_p}{2^m\sqrt{2}} \sum_{q=0}^{m} \binom{m}{q} p^{m-q} \int_0^{2l-p} \phi(y) y^q dy$$

$$= \sum_{p=0}^{2K-1} \frac{h_p}{2^m\sqrt{2}} \sum_{q=0}^{m} \binom{m}{q} p^{m-q} \langle x^q \rangle_{\phi[0,2l-p]}$$

Using $\frac{h_p}{\sqrt{2}} = c_p$

$$\langle x^m \rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \frac{c_p}{2^m} \sum_{q=0}^{m} \binom{m}{q} p^{m-q} \langle x^q \rangle_{\phi[0,2l-p]}$$

Rearranging the terms

$$\langle x^m \rangle_{\phi[0,l]} = \sum_{p=0}^{2K-1} \frac{c_p}{2^m} \sum_{q=0}^{m-1} \binom{m}{q} p^{m-q} \langle x^q \rangle_{\phi[0,2l-p]} + \sum_{p=0}^{2K-1} \frac{c_p}{2^m} \langle x^m \rangle_{\phi[0,2l-p]}$$

Use $r = 2l - p$ in the second term

$$\langle x^m \rangle_{\phi[0,n]} = \sum_{p=0}^{2K-1} \frac{c_p}{2^m} \sum_{q=0}^{m-1} \binom{m}{q} p^{m-q} \langle x^q \rangle_{\phi[0,2l-p]} + \sum_{r=2l-2K+1}^{2n} \frac{c_{2l-r}}{2^m} \langle x^m \rangle_{\phi[0,r]}$$

Moving the partial moment instances of $\langle x^m \rangle_{\phi[0,s]}$ to the left, a set of equations is constructed.

$$\sum_{q=1}^{2K-2} \left( \delta_{lq} - \frac{C_{lq}}{2^m} \right) \langle x^m \rangle_{\phi[0,q]}$$

$$= \delta_{(l \geq K)} \sum_{r=2l-2k+1}^{2l} \frac{c_{2l-r}}{2^m} \langle x^m \rangle_{\phi}$$

$$+ \sum_{p=0}^{2K-1} \frac{c_p}{2^m} \sum_{q=0}^{m-1} \binom{m}{q} p^{m-q} \langle x^q \rangle_{\phi[0,2l-p]}$$

written as follows

$$\sum_{q=1}^{2K-2} \left( \delta_{lq} - \frac{C_{lq}}{2^m} \right) \langle x^m \rangle_{\phi[0,q]} = w_l$$

where $w_p$

$$w_l = \delta_{(l \geq K)} \sum_{r=2l-2k+1}^{2l} \frac{c_{2l-r}}{2^m} \langle x^m \rangle_{\phi} + \sum_{p=0}^{2K-1} \frac{c_p}{2^m} \sum_{q=0}^{m-1} \binom{m}{q} p^{m-q} \langle x^q \rangle_{\phi[0,2l-p]}$$

which can be expressed in terms of the full moments of order $k$ and partial moments of order less than $k$. The desired order-$k$ partial moments are obtained by solving the system.

$$\langle x^m \rangle_{\phi[0,n]} = \sum_{q=1}^{2K-2} \left( \delta_{lq} - \frac{C_{lq}}{2^m} \right)^{-1} w_q$$

Note that the $C_{lq}$ matrix is identical to the $C_{lq}$ matrix that appears in equation for the 0-order partial moments (132).

***Complementary partial moments of any order of un-scaled and un-translated scaling functions***

Complementary higher order partial moments, $\langle x^m \rangle_{\phi[n,2n-1]}$ where *($l \in \mathbb{Z}$, $0 < l < 2n-1$),* are given like equation (133)

$$\langle x^m \rangle_{\phi[l,2K-1]} = \langle x^m \rangle_{\phi} - \langle x^m \rangle_{\phi[0,l]}$$

***Partial moments of any order of un-scaled but translated scaling functions overlapping zero***

By having the lower limit of the integral to be zero the partial moments, $\langle x^m \rangle_{\phi_{0k}[0,\infty)}$ where $k \in \mathbb{Z}$, $-(2n-1) < k < 0$*),* can be calculated as follows.

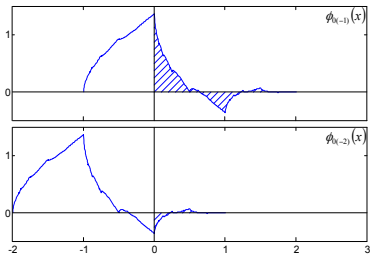


**Figure 30 - The DB2 Scaling functions parts that yield the 2 Partial moments**.
The shaded region indicates the part part that contributes to the integrals, $\langle x^m \rangle_{\phi_{0(-1)}[0,2]}$ and $\langle x^m \rangle_{\phi_{0(-2)}[0,2]}$.

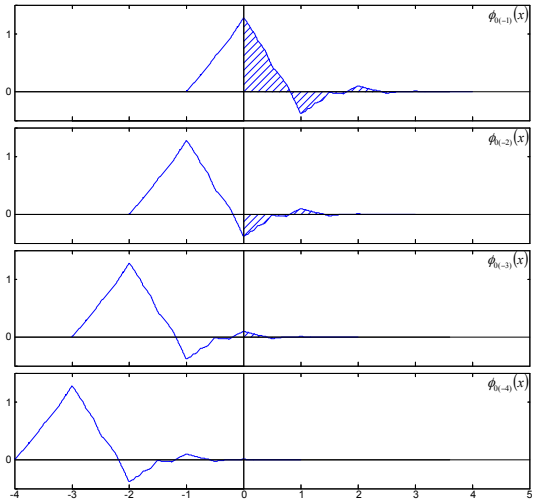


**Figure 31 - The DB3 Scaling functions part that yield the 4 Partial moments.**
Same as Figure 30 but for the DB3 scaling function.

Partial moment integrals can be constructed as follows

$$\langle x^m \rangle_{\phi_{0k}[0,\infty)} = \int_0^{\infty} x^m \phi(x-k)\, dx$$

If $k \leq -(2K-1)$ then $\langle x^m \rangle_{\phi_{0k}[0,\infty)} = 0$, If $k \geq 0$ then $\langle x^m \rangle_{\phi_{0k}[0,\infty)} = \langle x^m \rangle_{\phi_{0k}}$. Partial moments are calculated for $-(2K-1) < k < 0$. Using (107)

$$\langle x^m \rangle_{\phi_{0k}[0,\infty)} = \int_0^{\infty} x^m \sum_{p=0}^{2K-1} \sqrt{2} h_p \phi(2x - 2k - p)\, dx$$

By taking $2x = y$

$$\langle x^m \rangle_{\phi_{0k}[0,\infty)} = \int_0^{\infty} \frac{y^m}{2^m} \sum_{p=0}^{2K-1} \sqrt{2} h_p \phi\big(y - (2k + p)\big) \frac{dy}{2}$$

By taking $y = x$ and using $\frac{h_p}{\sqrt{2}} = c_p$

$$\langle x^m \rangle_{\phi_{0k}[0,\infty)} = \frac{1}{2^m} \sum_{p=0}^{2K-1} c_p \int_0^{\infty} x^m \phi\big(x - (2k + p)\big) dx$$

By writing $\langle x^m \rangle_{\phi_{0k}[0,\infty)} = N_k$

$$N_k = \frac{1}{2^m} \sum_{p=0}^{2K-1} c_p N_{2k+p} \tag{134}$$

When $k \geq 0$; $N_k = M_k$ where $M_k$ is a full moment

***Complementary partial moments of any order of un-scaled but translated scaling functions overlapping zero***

Complementary higher order partial moments $\langle x^m \rangle_{\phi_{0k}(-\infty,0]}$ where $k \in \mathbb{Z}$, $-(2n-1) < k < 0$) are given by

$$\langle x^m \rangle_{\phi_{ok}(-\infty,0]} = \langle x^m \rangle_{\phi_{ok}} - \langle x^m \rangle_{\phi_{ok}[0,\infty)}$$

***Partial moments of any order of scaled and translated scaling functions overlapping zero***

Partial moment integrals of any scale, $\langle x^m \rangle_{\phi_{jk}[0,\infty)}$ where $k \in \mathbb{Z}$, $-(2n-1) < k < 0$), can be constructed as follows

$$\langle x^m \rangle_{\phi_{jk}[0,\infty)} = \int_0^{\infty} x^m \phi_{j0}(x - k)\, dx$$

If $k \leq -(2K-1)$ then $\langle x^m \rangle_{\phi_{jk}[0,\infty)} = 0$, If $k \geq 0$ then $\langle x^m \rangle_{\phi_{jk}[0,\infty)} = \langle x^m \rangle_{\phi_{jk}}$. Partial moments are calculated for $-(2K-1) < k < 0$. Using (107)

$$\langle x^m \rangle_{\phi_{jk}[0,\infty)} = \int_0^{\infty} x^m \sum_{p=0}^{2K-1} \sqrt{2} h_p \phi_{j0}(2x - 2k - p)\, dx$$

By taking $2x = y$

$$\langle x^m \rangle_{\phi_{jk}[0,\infty)} = \int_0^{\infty} \frac{y^m}{2^m} \sum_{p=0}^{2K-1} \sqrt{2} h_p \phi_{j0}\big(y - (2k + p)\big) \frac{dy}{2}$$

By taking $y = x$ and using $\frac{h_p}{\sqrt{2}} = c_p$

$$\langle x^m \rangle_{\phi_{jk}[0,\infty)} = \frac{1}{2^m} \sum_{p=0}^{2K-1} c_p \int_0^{\infty} x^m \phi_{j0}\big(x - (2k + p)\big) dx$$

By writing $\langle x^m \rangle_{\phi_{jk}[0,\infty)} = N_k$

$$N_k = \frac{1}{2^m} \sum_{p=0}^{2K-1} c_p N_{2k+p} \tag{135}$$

When $k \geq 0$; $N_k = M_k$ where $M_k$ is a full moments of scale $j$. The process is the same as was used to obtain (134) but using full moments of scale $j$.

***Complementary partial moments of any order of scaled and translated scaling functions overlapping zero***

For complementary higher order partial moments of any scale and translation, $\langle x^m \rangle_{\phi_{jk}(-\infty,0]}$ where $k \in \mathbb{Z}$, $-(2n-1) < k < 0$) can be calculated as follows

$$\langle x^m \rangle_{\phi_{jk}(-\infty,0]} = \langle x^m \rangle_{\phi_{jk}} - \langle x^m \rangle_{\phi_{jk}[0,\infty)}$$

***Partial moments of any order of scaled and translated scaling functions***

Partial moment integrals where the bound is not essentially zero, $\langle x^m \rangle_{\phi_{jk}[l,\infty)}$ where ($l$ may not be an integer), be constructed as follows

$$\langle x^m \rangle_{\phi_{jk}[l,\infty)} = \int_l^{\infty} x^m \phi_{j0}(x - k)\, dx$$

Using the shift $y = x - l$ and the expansion

$$x^m = (y + l)^m = \sum_{p=0}^{m} \binom{m}{p} l^{m-p} y^p$$

we have

$$\langle x^m \rangle_{\phi_{jk}[l,\infty)} = \int_0^{\infty} \sum_{p=0}^{m} \binom{m}{p} l^{m-p} y^p\, \phi_{j0}(y + l - k)\, dy$$

$$= \sum_{p=0}^{m} \binom{m}{p} l^{m-p} \langle y^m \rangle_{\phi_{j(k-l)}[0,\infty)}$$

which can be solved since $\langle y^m \rangle_{\phi_{j(k-l)}[0,\infty)}$ is already calculated.

***Complementary partial moments of any order of scaled and translated scaling functions***

Complementary higher order partial moment integrals of any scale and translation with a boundary other than zero, $\langle x^m \rangle_{\phi_{jk}(-\infty,l]}$ where ($l$ may not be an integer), can be constructed as follows

$$\langle x^m \rangle_{\phi_{jk}(-\infty,l]} = \langle x^m \rangle_{\phi_{jk}} - \langle x^m \rangle_{\phi_{jk}[l,\infty)}$$

**Singular Moments**

***Singular moment for un-scaled but translated scaling function overlapping zero.***

Moment of order $-1$, $\langle x^{-1} \rangle_{\phi_{jk}}$where $k \in \mathbb{Z}$, $-(2n-1) < k < 0$), is singular and can be calculated as follows

$$I_{jk} = \int \frac{\phi_{jk}(x)}{x} dx = \int \frac{D^j T^k \phi(x)}{x} dx$$

Using the property (100)

$$\begin{aligned} I_{jk} &= \int D\left(D^j T^k \phi(x)\right) D\left(\frac{1}{x}\right) dx = \int \left(D^{j+1} T^k \phi(x)\right) \left(\frac{1}{x/2}\frac{1}{\sqrt{2}}\right) dx \\ &= \sqrt{2} \int \frac{D^j \left(D T^k \phi(x)\right)}{x} dx \end{aligned} \tag{136}$$

Using the property (102)

$$I_{jk} = \sqrt{2} \int \frac{D^j \left(T^{2k} D \phi(x)\right)}{x} dx$$

Since $D\phi(x) = \sum_{p=0}^{2K-1} h_p T^p \phi(x)$

$$I_{jk} = \sqrt{2} \int \frac{D^j \left(T^{2k} \sum_{p=0}^{2K-1} h_p T^p \phi(x)\right)}{x} dx = \sum_{p=0}^{2K-1} \sqrt{2} h_p \int \frac{D^j T^{2k+p} \phi(x)}{x} dx$$

$$I_{jk} = \sum_{p=0}^{2K-1} \sqrt{2} h_p I_{j(2k+p)} \tag{137}$$

Gives the linear system for the scaling functions of scale - 0

$$I_{0k} = \sum_{p=0}^{2K-1} \sqrt{2} h_p I_{0(2k+p)} \tag{138}$$

Starting off by

$$I_{jk} = \int \frac{\phi_{jk}(x)}{x} dx = \int \frac{D^j T^k \phi(x)}{x} dx = \int \frac{\phi(x)}{T^{-k} D^{-j} x} dx = 2^{-\frac{j}{2}} \int \frac{\phi(x)}{x+k} dx$$

Using the series expansion $\frac{1}{x+k} = \frac{1}{k} \sum_{p=0}^{\infty} \left(\frac{-x}{n}\right)^p$

$$I_{jk} = 2^{-\frac{j}{2}} \int \phi(x) \frac{1}{k} \sum_{p=0}^{\infty} \left(\frac{-x}{k}\right)^p dx = \frac{2^{-\frac{j}{2}}}{k} \sum_{p=0}^{\infty} \left(\frac{-1}{k}\right)^p \langle x^p \rangle_{\phi_{00}}$$

Ending up with the linear relationship where $\langle x^p \rangle_{\phi_{00}}$ are known from the previous moment calculations. For the scale - 0

$$I_{0k} = \frac{1}{k} \sum_{p=0}^{\infty} \left(\frac{-1}{k}\right)^p \langle x^p \rangle_{\phi_{00}} \tag{139}$$

Finally using the standard integral

$$\int_{-a}^{a} \frac{1}{x \pm 0i} dx = \pm \pi i \tag{140}$$

Since $\sum_{p=-\infty}^{\infty} \phi_{0p}(x) = 1$

$$\int_{-a}^{a} \frac{\sum_{p=-b-2K-1}^{b} \phi_{0p}(x)}{x} dx = \pm \pi i$$

Ending up with the linear relationship

$$\sum_{p=-b-2K-1}^{b} I_{0p} = \pm \pi i \tag{141}$$

The linear system (5) can be solved with constraints (139) and (141) to obtain singular moments for the scale – 0

*Application*

For the case $K = 2$ (DB2 scaling function). There are two singular moments for $k = -1$ & $-2$ and these are shown in Figure 32.

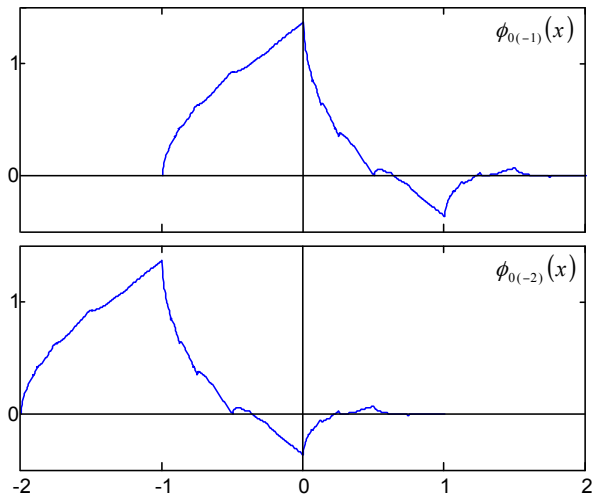



**Figure 32 - DB2 scaling functions have 2 singular moments.**

Using (5) for $k = -1$ & $-2$

$$I_{0(-1)} = \sum_{p=0}^{2K-1} \sqrt{2} h_p I_{0(-2+p)} \qquad I_{0(-2)} = \sum_{p=0}^{2K-1} \sqrt{2} h_p I_{0(-4+p)}$$

The first subscript (scale) can be omitted since all terms are in the same scale.

$$I_{-1} = \sqrt{2}(h_0 I_{-2} + h_1 I_{-1} + h_2 I_0 + h_3 I_1)$$

$$I_{-2} = \sqrt{2}(h_0 I_{-4} + h_1 I_{-3} + h_2 I_{-2} + h_3 I_{-1})$$

$$\begin{bmatrix} I_{-1} \\ I_{-2} \end{bmatrix} = \sqrt{2} \begin{bmatrix} h_1 & h_0 \\ h_3 & h_2 \end{bmatrix} \begin{bmatrix} I_{-1} \\ I_{-2} \end{bmatrix} + \sqrt{2} \begin{bmatrix} h_2 I_0 + h_3 I_1 \\ h_0 I_{-4} + h_1 I_{-3} \end{bmatrix} \tag{142}$$

for $k = 0$

$$I_0 = \sqrt{2}(h_0 I_0 + h_1 I_1 + h_2 I_2 + h_3 I_3)$$

$$I_0 = \frac{\sqrt{2}}{1 - \sqrt{2} h_0}(h_1 I_1 + h_2 I_2 + h_3 I_3)$$

for $k = -3$

$$I_{-3} = \sqrt{2}(h_0 I_{-6} + h_1 I_{-5} + h_2 I_{-4} + h_3 I_{-3})$$

$$I_{-3} = \frac{\sqrt{2}}{1 - \sqrt{2} h_3}(h_0 I_{-6} + h_1 I_{-5} + h_2 I_{-4})$$

In order to use (141), two bounds are imposed. The terms that relate to the 2 singular moments are shown in Figure 33.

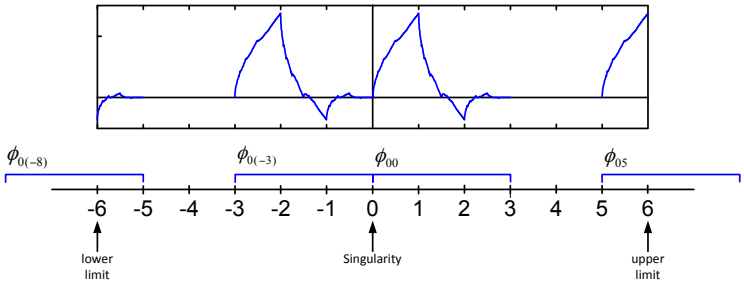



**Figure 33 - The non-singular moments around the singularity that are related to the 2 singular moments.**

When the support of the function contains the bound the partial moment will be considered. Starting at (140)

$$\int_{-6}^{6} \frac{1}{x \pm 0i} dx = \mp \pi i$$

$$\int_{-6}^{6} \frac{\sum_{p=-8}^{5} \phi_{0p}(x)}{x} dx = \mp \pi i$$

Equation (141) becomes (scale subscript is omitted)

$$\sum_{p=-8}^{5} I_p = \mp\pi i$$

$$I_{-8} + I_{-7} + I_{-6} + I_{-5} + I_{-4} + I_{-3} + I_{-2} + I_{-1} + I_0 + I_1 + I_2 + I_3 + I_4 + I_5 = \mp\pi i$$

When the limits are $-6$ and 6 the terms $I_{-8}$ and $I_{-7}$ are partial moments of order $-1$ since the support of $\phi_{0(-8)}$ and $\phi_{0(-7)}$ include the lower bound $-6$. The terms $I_{-6}$, $I_{-5}$, $I_{-4}$, $I_{-3}$, $I_0$, $I_1$, $I_2$, $I_3$ are full moments of order $-1$. The terms $I_4$ and $I_5$ are also partial moments of order $-1$ since the support of $\phi_{05}$ and $\phi_{06}$ include the upper bound 6. In order to combine with equation (142) this is represented in the matrix form

$$\begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}\begin{bmatrix}I_{-1}\\I_{-2}\end{bmatrix} + \begin{bmatrix}I_{-8} + I_{-7} + I_{-6} + I_{-5} + I_{-4} + I_{-3} + I_0 + I_1 + I_2 + I_3 + I_4 + I_5 \mp \pi i\\0\end{bmatrix} \tag{143}$$

By (142)+(143)

$$\begin{bmatrix}I_{-1}\\I_{-2}\end{bmatrix} = \left[\sqrt{2}\begin{bmatrix}h_1 & h_0\\h_3 & h_2\end{bmatrix} + \begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}\right]\begin{bmatrix}I_{-1}\\I_{-2}\end{bmatrix} + \begin{bmatrix}\sqrt{2}h_2I_0 + \sqrt{2}h_3I_1 + I_{-8} + I_{-7} + I_{-6} + +I_4 + I_5 \ldots\\ +I_{-5} + I_{-4} + I_{-3} + I_0 + I_1 + I_2 + I_3 \mp \pi i\\ \sqrt{2}h_0I_{-4} + \sqrt{2}h_1I_{-3}\end{bmatrix} \tag{144}$$

This system can be solved for $I_{-1}$ and $I_{-2}$

***Singular moment for scaled but translated scaling function overlapping zero.***

Singular moment of order $-1$, $\langle x^{-1}\rangle_{\phi_{jk}}$ where $k \in \mathbb{Z}$, $-(2n-1) < k < 0$) can be calculated using un-scale singular moments of the same order, $I_{0n}$

$$I_{jk} = 2^{j/2} I_{0k} \tag{145}$$

# Appendix C

## Quadrature Points and Weights Code Listings

The following Matlab script generates the quadrature points and weights at different scales. The generated points and weights are saved to a data file that can be used in the next step of the calculation. The functions required by the `generator` script are listed below

```
generator.m
  ↳initialize.m
  ↳initfunc.m
  ↳moment00.m
  ↳moment.m
  ↳partialmoment.m
  ↳subpartialmoment.m
  ↳support.m
  ↳isinsup.m
  ↳gaulegf.m
  ↳makeNmn.m
```

The functions are listed below.

### *generator.m*

Generates Quadrature points and weights

```
%generator Generates Points and Weights and saves it in a structure called
%   data. The structure is saved in a 'J=%J a=%a N=%N.mat' file
%   The data structure contains the following
%
%   data.N - Number of wavelets
%   data.J - The Scale of the wavelets
%   data.a - Lower Limit of the problem
%   data.b - Upper Limit of the problem
%   data.K - Wavelet Number, for Daubachies2 : K = 2
%                            for Daubachies3 : K = 3
%   data.dim - unscaled wavelet length
%   data.h - the scaling coefficients
%   data.g - the wavelet coefficients
%   data.c - the scaling coefficients/sqrt(2)
%   data.points - Quadrature points
%   data.weights - weights
%   data.Nmn - The wavelet Ideneitiy matrix
%
%   Uses globals:
%       K - Wavelet Number
%       N - Number of wavelets
%       J - The Scale of the wavelets
%       a - Lower Limit of the problem
%       b - Upper Limit of the problem
%       dim - unscaled wavelet length

clear all
clc

format long

global K N J a b dim h

initialize('db2')

j = 10
initfunc(-j,0,(2^j)+2)
```

```
Sn = -a/(2^J) - dim + 1
En = Sn + N - 1
a
b

points = NaN(K+1,N);
weights = zeros(K+1,N);
imnvals = zeros(1,N);

for kl = Sn:En

    disp(sprintf('kl = %d',kl))

    [LB UB] = support(J,kl);

    if (isinsup(J,kl,-a))
        [x,remw] = gaulegf(-a,UB,K+1);

        for tl = 0:K
            A(tl+1,:) = x.^tl;
            pmom = partialmoment(J,tl,-a);
            pmomx(tl+1) = pmom(dim-(-Sn+kl+1));
        end

        w = A\pmomx';

        points(:,-Sn+kl+1) = x';
        weights(:,-Sn+kl+1) = w;

    elseif (~(isinsup(J,kl,-a)) && ~(isinsup(J,kl,b))) && ~(isinsup(J,kl,0))

        points(1,-Sn+kl+1) = (2^J)*(moment(0,0,1)+kl);
        weights(1,-Sn+kl+1) = 2^(J/2);

    elseif (isinsup(J,kl,0))

        points(1,-Sn+kl+1) = (2^J)*(moment(0,0,1)+kl);
        weights(1,-Sn+kl+1) = 2^(J/2);

    elseif (isinsup(J,kl,b))
        [x,remw] = gaulegf(LB,b,K+1);

        for tl = 0:K
            A(tl+1,:) = x.^tl;
            pmom = moment(J,kl,tl) - partialmoment(J,tl,b);
            pmomx(tl+1) = pmom(N - (dim-1) + (dim-(-Sn+kl+1)));
        end

        w = (A\pmomx');

        points(:,-Sn+kl+1) = x';
        weights(:,-Sn+kl+1) = w;

    end


end


data.J = J;
data.K = K;
data.N = N;
data.a = a;
data.b = b;
data.dim = dim;
data.h = h;
data.points = points;
data.weights = weights;
data.Nmn = makeNmn(data.N);

filename = sprintf('J=%d a=%d N=%d',J,a,N)

save(filename,'data','-V7.3')
```

***initialize.m***

Initializes the scaling coefficients, $h_l$, as given in Table 1. The wavelets coefficients $g_l$, are given by (13).

```
function initialize(dbn)
% initialize(dbn) Initializes the coefficients for the corresponding
%   wavelets
%
%   Input:
%       dbn = 'db2' or 'db3'
%
%   Output:
%       defines globals  K, dim, h, g, c
%       K - Wavelet Number, for Daubachies2 : K = 2
%                           for Daubachies3 : K = 3
%       dim - unscaled wavelet length
%       h - the scaling coefficients
%       g - the wavelet coefficients
%       c - the scaling coefficients/sqrt(2)

global K dim h g c

if (strcmp(dbn, 'db1'))
    K = 1;
    h = [1/2
         1/2];
    g = [1/2
        -1/2];
elseif (strcmp(dbn, 'db2'))
    K = 2;
    h = [(1 + sqrt(3)) / (4 * sqrt(2))
         (3 + sqrt(3)) / (4 * sqrt(2))
         (3 - sqrt(3)) / (4 * sqrt(2))
         (1 - sqrt(3)) / (4 * sqrt(2))];
    g = [(1 - sqrt(3)) / (4 * sqrt(2))
        -(3 - sqrt(3)) / (4 * sqrt(2))
         (3 + sqrt(3)) / (4 * sqrt(2))
        -(1 + sqrt(3)) / (4 * sqrt(2))];
elseif (strcmp(dbn, 'db3'))
    K = 3;
    h = [(1  +     sqrt(10) +     sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (5  +     sqrt(10) + 3 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (10 - 2 * sqrt(10) + 2 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (10 - 2 * sqrt(10) - 2 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (5  +     sqrt(10) - 3 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (1  +     sqrt(10) -     sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))];
    g = [(1  +     sqrt(10) -     sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
        -(5  +     sqrt(10) - 3 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (10 - 2 * sqrt(10) - 2 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
        -(10 - 2 * sqrt(10) + 2 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
         (5  +     sqrt(10) + 3 * sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))
        -(1  +     sqrt(10) +     sqrt(5 + 2 * sqrt(10))) / (16 * sqrt(2))];
elseif (strcmp(dbn, 'db4'))
    error('Simulink:actionNotTaken', 'error!, wavelet id not implemented %s\n', dbn);
else
    error('Simulink:actionNotTaken', 'error!, invalid wavelet id %s\n', dbn);
end

dim = 2*K-1;
c = h / sqrt(2);

end
```

***initfunc.m***

Initializes scale $J$ and upper and lower bounds, $a$ and $b$.

```
function initfunc(scale,aval,Nval)
%initfunc(scale,aval,Nval) Initializes the limits and scale and number of
%   wavelets.
%
%   Input:
%       scale - The Scale of the wavelets
%       aval - Lower Limit of the problem
%       Nval - Number of wavelets
%   Uses globals:
%       K - Wavelet Number
%   Output:
%       Global Variables N J a b are defined and initalized
%       N - Number of wavelets
%       J - The Scale of the wavelets
%       a - Lower Limit of the problem
%       b - Upper Limit of the problem

global K N J a b

N = Nval;
J = scale;
a = -aval;
b  = -a+(N-2*K+2)*(2^J);

end
```

***moment00.m***

Full moment of any order for unscaled, untranslated scaling functions as given by equation (127).

```
function fmom = moment00(order)
% moment(scale, trans, order) Returns Any order full moments for scaling
%   functions with
%   Translation = 0
%   Scale = 0
%
%   Input:
%       order - Order of the moment (nth moment)
%   Uses globals:
%       h - the scaling coefficients
%       K - Wavelet Number
%       dim - unscaled wavelet length
%   Output:
%       mom = <x^n>phi00

m = order;

global h K dim

if (m == 0)
    fmom = 1;
elseif (m == 1)
    temp0 = 0;
    for il = 0:dim
        temp0 = temp0 + il * h(il+1);
    end
    fmom = temp0 / sqrt(2);
else
    temp1 = 0;
    for kl = 0:(m - 1)
        temp2 = 0.0;
        for ll = 0:dim
            temp2 = temp2 + h(ll+1) * ll^(m - kl);
        end
        temp1 = temp1 + nchoosek(m, kl) * temp2 * moment00(kl);
    end
    fmom = temp1 / (sqrt(2)*((2^m) - 1));
end

end
```

***moment.m***

Full moment of any order for scaled, translated scaling functions as in equation (130).

```
function mom = moment(scale, trans, order)
% moment(scale, trans, order) Returns Any order full moments for scaling
%   functions with
%   Translation = any translation
%   Scale = any scale
%
%   Input:
%       scale - scale of the scaling functions
%       trans - translation of the scaling functions
%       order - Order of the moment (nth moment)
%   Uses globals:
%       h - the scaling coefficients
%       K - Wavelet Number
%       dim - unscaled wavelet length
%
%   Output:
%       mom = <x^n>phijk

k = scale;
l = trans;
m = order;

global h K dim

temp0 = 0;
for nl = 0:m
    temp0 = temp0 + nchoosek(m, nl) * (l^(m - nl)) * moment00(nl);
end
mom = (2^(k * (m + 0.5))) * temp0;

end
```

***subpartialmoment.m***

Partial moments calculated using equation (135).

```
function pmom = subpartialmoment(scale,order)
% subpartialmoment(scale,order) Returns Any order partial moments
%   in the boundry [0:inf] for scaling functions with
%   Translation = -1, -2, ... , -(2K-2)
%   Scale = any scale
%
%   Input:
%       scale - scale of the scaling functions
%       order - Order of the moment (nth moment)
%   Uses globals:
%       c - the scaling coefficients/sqrt(2)
%       K - Wavelet Number
%       dim - unscaled wavelet length
%
%   Output:
%       pmom = [ <x^n>phij(-1)[0:inf],...
%                <x^n>phij(-2)[0:inf],...
%                ...,
%                <x^n>phij(-(2k-2))[0:inf] ]

k = order;
j = scale;

global c K dim

for m = 1:dim-1
    for n = 1:dim-1
        if (((2*m-n) < 0) || ((2*m-n) > dim))
            C(m,n) = 0;
        else
            C(m,n) = c(2*m-n+1)/(2^k);
        end
        if (m == n)
            C(m,n) = 1 - C(m,n);
        else
            C(m,n) = -C(m,n);
        end
```

```
    end
end

CI = inv(C);

for n = 1:dim-1;
    v(n) = 0;
    if ((n >= K) && (n <= 2*K-2))
        v(n) = 0;
    else
        for kl = 0:dim-2*n;
            v(n) = v(n) + c(2*n+kl+1)*moment(j, kl, k)/(2^k);
        end
    end
end


for m = 1:dim-1;
        pmom(m) = 0;
end

for m = 1:dim-1;
    for n = 1:dim-1;
        pmom(m) = pmom(m) + CI(m,n) * v(n);
    end
end


end
```

## *partialmoment.m*

```
function pmom = partialmoment(scale,order,lim)
% partialmoment(scale,order,lim) Returns Any order partial moments
%   in the boundry [lim:inf] for scaling functions with
%   Translation = lim-1, lim-2, ... , lim-(2K-2)
%   Scale = any scale
%
%   Input:
%       scale - scale of the scaling functions
%       order - Order of the moment (nth moment)
%       lim - Limit
%   Uses globals:
%       c - the scaling coefficients/sqrt(2)
%       K - Wavelet Number
%       dim - unscaled wavelet length
%
%   Output:
%       pmom = [ <x^n>phij(lim-1)[lim:inf],...
%                <x^n>phij(lim-2)[lim:inf],...
%                ...,
%                <x^n>phij(lim-(2k-2))[lim:inf] ]

j = scale;
m = order;
a = lim;

global c K dim

for kl = 1:dim-1
    pmom(kl) = 0;
    for r = 0:m
        y = subpartialmoment(j,r);
        pmom(kl) = pmom(kl) + nchoosek(m,r)*(a^(m-r))*y(kl);
    end
end


end
```

***support.m***

Gives the support of a scaling function.

```
function [LB UB] = support(scale, trans)
% support(scale, trans) Returns lower and upper bounds of a scaling function
%   with a given scale and translation
%
%   Input:
%       scale - scale of the wavelet
%       trans - translation of the wavelet
%   Uses globals:
%       dim - unscaled wavelet length
%
%   Output:
%       LB - lower bound of the support of the scaling function phi(scale,trans)
%       UB - upper bound of the support of the scaling function phi(scale,trans)

j = scale;
i = trans;

global dim

LB = i/2^-j;
UB = (i + dim)/2^-j;

end
```

***isinsup.m***

Checks if a given value is in the support of a scaling function specified by a scale and translation.

```
function retval = isinsup(scale,trans,val)
% isinsup(scale,trans,val) Checks if a given value is in the support of a given
%   scaling function
%
%   Input:
%       scale - scale of the wavelet
%       trans - translation of the wavelet
%       val - translation of the wavelet
%   Uses globals:
%       dim - unscaled wavelet length
%
%   Output:
%       retval - Boolean for LB < val && val < UB

j = scale;
i = trans;

global dim

LB = i/2^-j;
UB = (i + dim)/2^-j;

retval = LB < val && val < UB;

end
```

***gaulegf.m***

Generates n number of Gauss-Legendre Quadrature points and weights

```
function [x,w] = gaulegf(x1,x2,n)
% gaulegf(x1,x2,n) Generates n number of Gauss-Legendre Quadrature points and
%   weights (w) in the interval [x1 x2]
%
%   Input:
%       x1 - Lower limit
%       x2 - Lower limit
%       n - Number of points
%
%   Output:
%       x - Quadrature points
%       w - weights
```

```
eps = 3.0E-14;

m = (n+1)/2;
xm = 0.5*(x2+x1);
xl = 0.5*(x2-x1);
for i = 1:m
    z = cos(3.141592654*(i-0.25)/(n+0.5));
    while (true)
        p1 = 1.0;
        p2 = 0.0;
        for j=1:n
            p3 = p2;
            p2 = p1;
            p1 = ((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
        end
        pp = n*(z*p1-p2)/(z*z-1.0);
        z1 = z;
        z = z1 - p1/pp;
        if(abs(z-z1) <= eps);break;end
    end
    x(i) = xm - xl*z;
    x(n+1-i) = xm + xl*z;
    w(i) = 2.0*xl/((1.0-z*z)*pp*pp);
    w(n+1-i) = w(i);
end


end
```

### *makeNmn.m*

Generates $N_{mn}$ Matrix using the system of equations given by (85) and (86)

```
function Nmn = makeNmn(N)
% makeNmn(scale, trans) Returns the wavelet identity matrix of a given size
%   that takes in to account the orthonrmality of wavelets with the edge
%   effects
%
%   Input:
%       N - The size of the identity
%   Uses globals:
%       h - the scaling coefficients
%       K - Wavelet Number
%
%   Output:
%       N - an NxN wavelet identity matrix

global h K

Nmn = eye(N);

if (K == 2)

    ANmn =[h(2)*h(3) h(2)*h(4) h(1)*h(3) h(1)*h(4);
           h(2)*h(1) h(2)*h(2) h(1)*h(1) h(1)*h(2);
           h(4)*h(3) h(4)*h(4) h(3)*h(3) h(3)*h(4);
           h(4)*h(1) h(4)*h(2) h(3)*h(1) h(3)*h(2)];

    nmnvals = (eye(4,4)-ANmn)\[0;(h(3)^2 + h(4)^2);0;0];

    Nmn(1,2) = nmnvals(1);
    Nmn(2,2) = nmnvals(2);
    Nmn(1,1) = nmnvals(3);
    Nmn(2,1) = nmnvals(4);

    nmnvals = (eye(4,4)-ANmn)\[0;0;(h(1)^2 + h(2)^2);0];

    Nmn(end-1,end)   = nmnvals(1);
    Nmn(end,end)     = nmnvals(2);
    Nmn(end-1,end-1) = nmnvals(3);
    Nmn(end,end-1)   = nmnvals(4);

elseif (K == 3)
    ANmn = [h(2)*h(5) h(2)*h(6)         0         0 h(1)*h(5) h(1)*h(6)         0         0         0
0         0         0         0         0         0;
          h(2)*h(3) h(2)*h(4) h(2)*h(5) h(2)*h(6) h(1)*h(3) h(1)*h(4) h(1)*h(5) h(1)*h(6)         0
0         0         0         0         0         0;
```

```
            h(2)*h(1) h(2)*h(2) h(2)*h(3) h(2)*h(4) h(1)*h(1) h(1)*h(2) h(1)*h(3) h(1)*h(4)         0
0         0         0         0         0         0         0;
                    0         0 h(2)*h(1) h(2)*h(2)         0 h(1)*h(1) h(1)*h(2)         0         0
0         0         0         0         0         0         0;
            h(4)*h(5) h(4)*h(6)         0         0 h(3)*h(5) h(3)*h(6)         0         0 h(2)*h(5)
h(2)*h(6)         0         0 h(1)*h(5) h(1)*h(6)         0         0;
            h(4)*h(3) h(4)*h(4) h(4)*h(5) h(4)*h(6) h(3)*h(3) h(3)*h(4) h(3)*h(5) h(3)*h(6) h(2)*h(3)
h(2)*h(4) h(2)*h(5) h(2)*h(6) h(1)*h(3) h(1)*h(4) h(1)*h(5) h(1)*h(6);
            h(4)*h(1) h(4)*h(2) h(4)*h(3) h(4)*h(4) h(3)*h(1) h(3)*h(2) h(3)*h(3) h(3)*h(4) h(2)*h(1)
h(2)*h(2) h(2)*h(3) h(2)*h(4) h(1)*h(1) h(1)*h(2) h(1)*h(3) h(1)*h(4);
                    0         0 h(4)*h(1) h(4)*h(2)         0         0 h(3)*h(1) h(3)*h(2)         0
0 h(2)*h(1) h(2)*h(2)         0         0 h(1)*h(1) h(1)*h(2);
            h(6)*h(5) h(6)*h(6)         0         0 h(5)*h(5) h(5)*h(6)         0         0 h(4)*h(5)
h(4)*h(6)         0         0 h(3)*h(5) h(3)*h(6)         0         0;
            h(6)*h(3) h(6)*h(4) h(6)*h(5) h(6)*h(6) h(5)*h(3) h(5)*h(4) h(5)*h(5) h(5)*h(6) h(4)*h(3)
h(4)*h(4) h(4)*h(5) h(4)*h(6) h(3)*h(3) h(3)*h(4) h(3)*h(5) h(3)*h(6);
            h(6)*h(1) h(6)*h(2) h(6)*h(3) h(6)*h(4) h(5)*h(1) h(5)*h(2) h(5)*h(3) h(5)*h(4) h(4)*h(1)
h(4)*h(2) h(4)*h(3) h(4)*h(4) h(3)*h(1) h(3)*h(2) h(3)*h(3) h(3)*h(4);
                    0         0 h(6)*h(1) h(6)*h(2)         0         0 h(5)*h(1) h(5)*h(2)         0
0 h(4)*h(1) h(4)*h(2)         0         0 h(3)*h(1) h(3)*h(2);
                    0         0         0         0         0         0         0         0 h(6)*h(5)
h(6)*h(6)         0         0 h(5)*h(5) h(5)*h(6)         0         0;
                    0         0         0         0         0         0         0         0 h(6)*h(3)
h(6)*h(4) h(6)*h(5) h(6)*h(6) h(5)*h(3) h(5)*h(4) h(5)*h(5) h(5)*h(6);
                    0         0         0         0         0         0         0         0 h(6)*h(1)
h(6)*h(2) h(6)*h(3) h(6)*h(4) h(5)*h(1) h(5)*h(2) h(5)*h(3) h(5)*h(4);
                    0         0         0         0         0         0         0         0         0
0 h(6)*h(1) h(6)*h(2)         0         0 h(5)*h(1) h(5)*h(2)];

    nmnvals = (eye(16,16)-
ANmn)\[0;0;(h(3)*h(5)+h(4)*h(6));(h(3)^2+h(4)^2+h(5)^2+h(6)^2);0;0;(h(5)^2+h(6)^2);(h(3)*h(5)+h(4)*h(6
));0;0;0;0;0;0;0;0];

    Nmn(1,1) = nmnvals(13);
    Nmn(2,1) = nmnvals(14);
    Nmn(3,1) = nmnvals(15);
    Nmn(4,1) = nmnvals(16);
    Nmn(1,2) = nmnvals(9);
    Nmn(2,2) = nmnvals(10);
    Nmn(3,2) = nmnvals(11);
    Nmn(4,2) = nmnvals(12);
    Nmn(1,3) = nmnvals(5);
    Nmn(2,3) = nmnvals(6);
    Nmn(3,3) = nmnvals(7);
    Nmn(4,3) = nmnvals(8);
    Nmn(1,4) = nmnvals(1);
    Nmn(2,4) = nmnvals(2);
    Nmn(3,4) = nmnvals(3);
    Nmn(4,4) = nmnvals(4);

    nmnvals = (eye(16,16)-
ANmn)\[0;0;0;0;0;0;0;0;(h(1)*h(3)+h(2)*h(4));(h(1)^2+h(2)^2);0;0;(h(1)^2+h(2)^2+h(3)^2+h(4)^2);(h(1)*h
(3)+h(2)*h(4));0;0];

    Nmn(end-3,end-3) = nmnvals(13);
    Nmn(end-2,end-3) = nmnvals(14);
    Nmn(end-1,end-3) = nmnvals(15);
    Nmn(end  ,end-3) = nmnvals(16);
    Nmn(end-3,end-2) = nmnvals(9);
    Nmn(end-2,end-2) = nmnvals(10);
    Nmn(end-1,end-2) = nmnvals(11);
    Nmn(end  ,end-2) = nmnvals(12);
    Nmn(end-3,end-1) = nmnvals(5);
    Nmn(end-2,end-1) = nmnvals(6);
    Nmn(end-1,end-1) = nmnvals(7);
    Nmn(end  ,end-1) = nmnvals(8);
    Nmn(end-3,end)   = nmnvals(1);
    Nmn(end-2,end)   = nmnvals(2);
    Nmn(end-1,end)   = nmnvals(3);
    Nmn(end  ,end)   = nmnvals(4);
end

end
```

**Eigenvalue Problem Code Listings**

The following Matlab script uses the quadrature points and weights file generated from the generate function and calculates the kernel matrix and then solves the matrix eigenvalue problem at the different scales. Apart from solving the eigenvalue problem this script also saves the calculated kernel matrix in to a file that corresponds with the scale, The functions required by the calculator script are listed below

```
calculator.m
  ↳Fun_J_mp.m
  ↳Fun_J_pp.m
  ↳Fun_J_mm.m
  ↳Fun_J_pm.m
  ↳Fun_P.m
  ↳Fun_Q.m
  ↳lambda2g.m
  ↳g2lambda.m
  ↳Fun_g.m
    ↳Fun_In.m
    ↳Fun_Lnib.m
```

The functions are listed below.

***calculator.m***

Calculates the kernel matrices and then eigenvalues

```
%calculator Calculates the kernal using the Points and Weights calculated
%   by the generator and saves it
%   in a structure called batch. the kernal is saved in parts
%   [batch.K11{l} batch.K12{l} batch.K13{l};
%    batch.K21{l} batch.K22{l} batch.K23{l};
%    batch.K31{l} batch.K32{l} batch.K33{l}]
%   not as the combined whole
%   Where l is different scales
%
%   Defines globals
%       M2
%       mu = [mu(1) mu(2)];
%       m = [m(1) m(2)];
%
%   batch.M2 = M2;
%   batch.mu = mu;
%   batch.m = m;
%   Saves in batch.mat
%
%   Also solves the kernal eigenvalue problem at that scale and eigenvalue
%   are stored in result.vals_numerical{l}
%   result.resolution{l}
%   result.loop = batch.loop;
%   result.M2 = M2;
%   result.mu = mu;
%   result.m = m;
%   result.lambda contains the analytical method eigenvalues.
%   Saves in result.mat
%
%   tempbatch.mat
%   tempresult.mat
%   saves the above in each iteration so in the event of an interuption the
%   temporary files may be used
%
```

```
clear
clc

curdir = pwd;

dbn = 'db3'
Calcn = 'Calc5'

rdir = [curdir(1:max(strfind(curdir,'/'))) ['Generator/' dbn '/']];

file = {[rdir 'J=-3 a=0 N=10']
        [rdir 'J=-4 a=0 N=18']
        [rdir 'J=-5 a=0 N=34']
        [rdir 'J=-6 a=0 N=66']
        [rdir 'J=-7 a=0 N=130']
        [rdir 'J=-8 a=0 N=258']
        [rdir 'J=-9 a=0 N=514']
        [rdir 'J=-10 a=0 N=1026']
        [rdir 'J=-11 a=0 N=2050']
        [rdir 'J=-12 a=0 N=4098']
        [rdir 'J=-13 a=0 N=8194']
        [rdir 'J=-14 a=0 N=16386']
        [rdir 'J=-15 a=0 N=32770']};

global m mu M2
M2 = 1
mu = [1 500]
m = [-2000 50000]

result.M2 = M2;
result.mu = mu;
result.m = m;

result.loop = [1 2 3 4 5]

T_total = tic;

for l = result.loop

    load(file{l});

    batch.N = data.N;

    [batch.K11 batch.K12 batch.K13 batch.K21 batch.K22 batch.K23 batch.K31 batch.K32 batch.K33] =
deal(zeros(data.N));

    T_eval = tic;

    fprintf('l = %d & J = %d : ',l,data.J);

    r = 0;

    for q = 1:data.N
        for p = 1:data.N

            for ll = 1:data.K+1
                for kl = 1:data.K+1
                    if (~isnan(data.points(ll,q)) && ~isnan(data.points(kl,p)))
                        batch.K11(q,p) = batch.K11(q,p) +
Fun_J_pp(0,0,data.points(ll,q),data.points(kl,p))*Fun_Q(0,data.points(kl,p)) * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K12(q,p) = batch.K12(q,p) +
Fun_J_pp(0,1,data.points(ll,q),data.points(kl,p))*Fun_Q(1,data.points(kl,p)) * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K13(q,p) = batch.K13(q,p) +
Fun_J_pm(0,data.points(ll,q),data.points(kl,p))*Fun_P(data.points(kl,p))  * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K21(q,p) = batch.K21(q,p) +
Fun_J_pp(1,0,data.points(ll,q),data.points(kl,p))*Fun_Q(0,data.points(kl,p)) * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K22(q,p) = batch.K22(q,p) +
Fun_J_pp(1,1,data.points(ll,q),data.points(kl,p))*Fun_Q(1,data.points(kl,p)) * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K23(q,p) = batch.K23(q,p) +
Fun_J_pm(1,data.points(ll,q),data.points(kl,p))*Fun_P(data.points(kl,p))  * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K31(q,p) = batch.K31(q,p) +
Fun_J_mp(0,data.points(ll,q),data.points(kl,p))*Fun_Q(0,data.points(kl,p)) * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K32(q,p) = batch.K32(q,p) +
```

```
Fun_J_mp(1,data.points(ll,q),data.points(kl,p))*Fun_Q(1,data.points(kl,p)) * data.weights(ll,q) *
data.weights(kl,p);
                        batch.K33(q,p) = batch.K33(q,p) +
Fun_J_mm(data.points(ll,q),data.points(kl,p))*Fun_P(data.points(kl,p))  * data.weights(ll,q) *
data.weights(kl,p);
                    end
                end
            end
            r = r+1;
        end
    end

    fprintf('%09d function evaluations in t = %08.2f',r,toc(T_eval))

    batch.Omn = zeros(data.N);
    batch.Nmn = data.Nmn;

    B = [batch.Nmn batch.Omn batch.Omn;
         batch.Omn batch.Nmn batch.Omn;
         batch.Omn batch.Omn batch.Nmn];

    A = [batch.K11 batch.K12 batch.K13;
         batch.K21 batch.K22 batch.K23;
         batch.K31 batch.K32 batch.K33];

    T_eigen = tic;

    [vectors values] = eigs(A,B);
    result.vals_numerical{l} = diag(values);
    result.resolution{l} = -data.J;
    result.N(l) = data.N;

    fprintf(' eigenvalue problem in t = %08.2f\n',toc(T_eigen))

    save('tempbatch','batch','-V6')
    save('tempresult','result','-V6')

    save([Calcn '/' dbn '/' sprintf('batch{%d}',l)],'batch','-V6')
    clearvars 'batch'

end

fprintf('total time elapsed t = %011.4f\n',toc(T_total))

save([Calcn '/' dbn '/' 'result'],'result','-V6')

%%

figure
for t = result.loop
    plot(result.resolution{t}*ones(size(result.vals_numerical{t})),(result.vals_numerical{t}),'.')
    hold on
end

plot([2 12],[g2lambda(Fun_g(1)) g2lambda(Fun_g(1))],'--k')
plot([2 12],[g2lambda(Fun_g(2)) g2lambda(Fun_g(2))],'--k')


xlabel('Resolution')
ylabel('eigenvalues')
```

### *Fun_J_mp.m*

Kernel function, $J_{-,a+}$ given by equation (61).

```
function Ret_J_mp = Fun_J_mp(a,y,yprime)
% Fun_J_mp(a,y,yprime) Analytical J(-,a+) kernel function

global m M2

Ret_J_mp = (((-1)^(a))/sqrt(y*yprime))*(1/(M2-(m(1)^2))) * (1/(1-y)) * ((m(a+1)/(1-yprime))+m(1))+...
           (((-1)^(a+1))/sqrt(y*yprime))*(1/(M2-(m(2)^2))) * (1/(1-y)) * ((m(a+1)/(1-yprime))+m(2));

end
```

### *Fun_J_pp.m*

Kernel function, $J_{i+,a+}$ given by equation (56).

```
function Ret_J_pp = Fun_J_pp(i,a,y,yprime)
% Fun_J_pp(i,a,y,yprime) Analytical J(i+,a+) kernel function

global m M2

Ret_J_pp = (((-1)^(a  ))/sqrt(y*yprime)) * (1/(M2-(m(1)^2))) * ((m(i+1)/(1-y))+m(1)) * ((m(a+1)/(1-
yprime))+m(1)) +...
           (((-1)^(a+1))/sqrt(y*yprime)) * (1/(M2-(m(2)^2))) * ((m(i+1)/(1-y))+m(2)) * ((m(a+1)/(1-
yprime))+m(2));

end
```

### *Fun_J_mm.m*

Kernel function, $J_{-,-}$ given by equation (64).

```
function Ret_J_mm = Fun_J_mm(y,yprime)
% Fun_J_mm(y,yprime) Analytical J(-,-) kernel function

global m M2

Ret_J_mm = ( 1/sqrt(y*yprime)) * (1/(M2-(m(1)^2))) * (1/(1-y)) * (1/(1-yprime)) +...
           (-1/sqrt(y*yprime)) * (1/(M2-(m(2)^2))) * (1/(1-y)) * (1/(1-yprime)) ;

end
```

### *Fun_J_pm.m*

Kernel function, $J_{i+,-}$ given by equation (60).

```
function Ret_J_pm = Fun_J_pm(i,y,yprime)
% Fun_J_pm(i,y,yprime) Analytical J(i+,-) kernel function

global m M2

Ret_J_pm = ( 1/sqrt(y*yprime)) * (1/(M2-(m(1)^2))) * ((m(i+1)/(1-y))+m(1)) * (1/(1-yprime)) +...
           (-1/sqrt(y*yprime)) * (1/(M2-(m(2)^2))) * ((m(i+1)/(1-y))+m(2)) * (1/(1-yprime));

end
```

### *Fun_P.m*

Kernel function, $P$ given by equation (59).

```
function Ret_P = Fun_P(Arg_yprime)
% Fun_P(Arg_yprime) Analytically obtained P(y) function

global m mu M2

Ret_P = (-1+Arg_yprime)*Arg_yprime*( (Arg_yprime*(m(1)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime -
(mu(1)^2)*(-1+Arg_yprime)) * log( (Arg_yprime*(m(1)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime + (mu(1)^2) -
Arg_yprime*(mu(1)^2)) ) - ...
                                  (Arg_yprime*(m(2)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime -
(mu(1)^2)*(-1+Arg_yprime)) * log( (Arg_yprime*(m(2)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime + (mu(1)^2) -
Arg_yprime*(mu(1)^2)) ) - ...
                                  (Arg_yprime*(m(1)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime -
(mu(2)^2)*(-1+Arg_yprime)) * log( (Arg_yprime*(m(1)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime + (mu(2)^2) -
Arg_yprime*(mu(2)^2)) ) + ...
                                  (Arg_yprime*(m(2)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime -
(mu(2)^2)*(-1+Arg_yprime)) * log( (Arg_yprime*(m(2)^2) + (M2)*(-1+Arg_yprime)*Arg_yprime + (mu(2)^2) -
Arg_yprime*(mu(2)^2)) ) );

end
```

### *Fun_Q.m*

Kernel function, $Q_a$ given by equation (55).

```
function Ret_Q = Fun_Q(Arg_a,Arg_yprime)
% Fun_Q(Arg_a,Arg_yprime) Analytically obtained Q_a(y) function

global m mu M2

Ret_Q = -(-1 + Arg_yprime)*Arg_yprime*( log( ((m(Arg_a+1)^2)*Arg_yprime + (M2)*(-1 +
Arg_yprime)*Arg_yprime + (mu(1)^2) - Arg_yprime*(mu(1)^2)) ) - ...
                                        log( ((m(Arg_a+1)^2)*Arg_yprime + (M2)*(-1 +
Arg_yprime)*Arg_yprime + (mu(2)^2) - Arg_yprime*(mu(2)^2)) ) );

end
```

### *lambda2g.m*

Converts the eigenvalue to Coupling $g$ equation (83).

```
function g = lambda2g(lambda)
% Converts the eigenvalue to coupling g eqn (83)

g = 4*pi./sqrt(lambda);

end
```

### *g2lambda.m*

Converts the Coupling $g$ to eigenvalue equation (83).

```
function lambda = g2lambda(g)
% converts coupling g to eigenvalue eqn(83)

lambda = (4*pi/g).^2;

end
```

### *Fun_g.m*

Calculates the coupling $g$ analytically using equation (43)

```
function g_analytical = Fun_g(n,d)

% Calculates the 2 analytical coupling values

global M2 mu m

if (nargin == 1), d = 1e-4;, end

g2_analytical(1) =  -( (sqrt(M2)-m(1))*(sqrt(M2)-m(2)) )/( (m(2)-m(1))*(mu(1)*Fun_In(1,d) +
sqrt(M2)*Fun_In(0,d)) );

g2_analytical(2) =  -( (sqrt(M2)+m(1))*(sqrt(M2)+m(2)) )/( (m(2)-m(1))*(mu(1)*Fun_In(1,d) -
sqrt(M2)*Fun_In(0,d)) );

if (n==1)
    g_analytical = sqrt(g2_analytical(1));
elseif (n==2)
    g_analytical = sqrt(g2_analytical(2));
end



end
```

### *Fun_In.m*

Functions (44) and (45) required for analytical $g$ calculation

```
function Ret_In = Fun_In(n,d)
% Fun_In(n) I(n) function for the analytical solution

global m mu

if (nargin == 1), d = 1; end

switch n
    case 0
        temp = 0;
        for i = 0:1
            for b = 0:1
                temp = temp + ((-1)^(i+b))*(Fun_Lnib(0,i,b,d) - Fun_Lnib(1,i,b,d));
            end
        end
    case 1
        temp = 0;
        for i = 0:1
            for b = 0:1
                temp = temp + ((-1)^(i+b))*(m(i+1)/mu(0+1))*Fun_Lnib(0,i,b,d);
            end
        end
end

Ret_In = temp/(16*(pi^2));

end
```

### *Fun_Lnib.m*

Function (46) required for analytical $g$ calculation

```
function Ret_Lnib = Fun_Lnib(n,i,b,d)
% Fun_Lnib(n,i,b) L(n,i,b) function for the analytical solution
% The integral uses Adaptive simpsons quadrature

global m mu M2

func = @(z,n,i,b) (z.^n).*log( z.*(m(i+1)^2)/M2 + (1-z).*(mu(b+1)^2)/M2 + z.*(1-z) );

funz = @(z) func(z,n,i,b);

Ret_Lnib = quad(funz,0,1,d);


end
```

**Filtering Code Listings**

The following Matlab script uses the calculated kernel matrix in the previous step and runs the wavelet transform on the matrix. The transformed matrix is then filtered using specified thresholds and the filtered matrix eigenvalue problem is again solved. The `TransNFilterNEigen` script will save the filtered eigenvalues to a file

```
TransNFilterNEigen
   ↳dwt2Nfilter
     ↳getthreshold
     ↳threshold
   ↳dwt2N
     ↳padmatrix
     ↳makewtn
```

***TransNFilterNEigen.m***

Transforms, filters and solves eigenvalue problem

```
function TransNFilterNEigen(th,Calcn,dbn,loop)

% transforms and Filteres the original eiegnvalue problem and then solves
% the resulting filtered eigenvalue problem
% th - Percentage filtering threshold
% Calcn - Specifies the folder with the Original kernel files
% dbn - Specifies DB2 or DB3
% loop - array with scales of the Original kernel files
% once done Will save the solved eigenvalues to a file 'filterres.mat'

curdir = pwd;

rdir = [curdir(1:max(strfind(curdir,'/'))) 'Calculator/' Calcn '/' dbn '/'];

addpath([curdir(1:max(strfind(curdir,'/'))) 'Calculator/'])

addpath([curdir(1:max(strfind(curdir,'/'))) 'Generator/'])

initialize(dbn)

N = 100;

T_total = tic;

for resn = loop

    filterbatch.resolution = resn+2;

    fprintf('resn = %02d & J = %d : ',resn,filterbatch.resolution);

    file = [rdir sprintf('batch{%d}',resn)];
    load(file);

    filterbatch.thres = th;

    T_filt = tic;

    [filterbatch.A11 filterbatch.N11 filterbatch.PN11] = dwt2Nfilter(batch.K11,N,th); [filterbatch.A12
filterbatch.N12 filterbatch.PN12] = dwt2Nfilter(batch.K12,N,th); [filterbatch.A13 filterbatch.N13
filterbatch.PN13] = dwt2Nfilter(batch.K13,N,th);
    [filterbatch.A21 filterbatch.N21 filterbatch.PN21] = dwt2Nfilter(batch.K21,N,th); [filterbatch.A22
filterbatch.N22 filterbatch.PN22] = dwt2Nfilter(batch.K22,N,th); [filterbatch.A23 filterbatch.N23
filterbatch.PN23] = dwt2Nfilter(batch.K23,N,th);
    [filterbatch.A31 filterbatch.N31 filterbatch.PN31] = dwt2Nfilter(batch.K31,N,th); [filterbatch.A32
filterbatch.N32 filterbatch.PN32] = dwt2Nfilter(batch.K32,N,th); [filterbatch.A33 filterbatch.N33
filterbatch.PN33] = dwt2Nfilter(batch.K33,N,th);
```

```
    filterbatch.PN = [filterbatch.PN11 filterbatch.PN12 filterbatch.PN13;
                      filterbatch.PN21 filterbatch.PN22 filterbatch.PN23;
                      filterbatch.PN31 filterbatch.PN32 filterbatch.PN33];

    filterbatch.percentageOfZeros = sum(filterbatch.PN(:))/9;
    filterres.percentageOfZeros = filterbatch.percentageOfZeros;

    fprintf('%2.1f%% filtered in t = %07.2f ',filterres.percentageOfZeros*100,toc(T_filt))

    B = [dwt2N(batch.Nmn,N,1) dwt2N(batch.Omn,N,0) dwt2N(batch.Omn,N,0);
         dwt2N(batch.Omn,N,0) dwt2N(batch.Nmn,N,1) dwt2N(batch.Omn,N,0);
         dwt2N(batch.Omn,N,0) dwt2N(batch.Omn,N,0) dwt2N(batch.Nmn,N,1)];

    clearvars batch

    A = [filterbatch.A11 filterbatch.A12 filterbatch.A13;
         filterbatch.A21 filterbatch.A22 filterbatch.A23;
         filterbatch.A31 filterbatch.A32 filterbatch.A33];

    clearvars filterbatch

    T_eigen = tic;

    [vectors values] = eigs(A,B);
    filterres.vals_numerical{resn} = (diag(values));
    filterres.resolution{resn} = resn+2;

    fprintf('eigenvalue problem in t = %08.2f\n',toc(T_eigen))

    save('tempfilterres','filterres','-V6')

%     save([Calcn '/' dbn '/'  strrep(sprintf('thresh(%3.2f)/',th),'.','')
sprintf('filterbatch(%d)',resn)],'filterbatch','-V6')

%     clearvars filterbatch

end

save([Calcn '/' dbn '/' strrep(sprintf('thresh(%3.2f)',th),'.','') '/filterres1'],'filterres','-V6')

end
```

## *getthreshold.m*

```
function thresholdValue = getthreshold(imageMatrix,precent)

% Given a % threshold this function returns the thresholdValue needed to 0
% out the specified percentage of elements

temp1 = abs(imageMatrix(:));
temp2 = temp1(temp1~=0);
imageArray = sort(temp2);
thresholdValue = imageArray(ceil(length(imageArray)*precent));

end
```

## *threshold.m*

```
function [filteredM num0] = threshold(originalM,thres)

% 0s all elements below or equal to a thres
% also returns number of Zeroed elemnts (avoids elements already zero)

[s1 s2] = size(originalM);
Originalvals = originalM(:);
vals = abs(Originalvals);
num0 = 0;

for loop = 1:length(vals)
    if ( (vals(loop)<=thres) && (vals(loop)~=0) ), Originalvals(loop) = 0; num0 = num0+1; end
end

filteredM = reshape(Originalvals,s1,s2);

end
```

### *dwt2N.m*

```
function Fx = dwt2N(Xin,N,pad)
% Fx = dwt2N(Xin,N,pad)
% Fx is the discrete wavelet transformation in 2-dimensions of Matrix Xin
% The number of iterations (or levels) is specified by N But N will be
% limited at the lowest level.
% The global variable K has to be defined

global K
[l w] = size(Xin);
s = l;

if (l ~= w)
    error('Simulink:actionNotTaken', 'error!, input not square, cannot be transformed');
else

    X = padmatrix(Xin,pad);
    [l w] = size(X);
    s = l;
    if (N > nextpow2(s)-1), N = nextpow2(s)-1; end

    Fx = X;

    if (N == 1)
        trM = makewtn(s);
        Fx = (trM)*X*(trM');
    elseif (N>1)
        trM = makewtn(s);
        Fx = (trM)*X*(trM');
        for i = 2:N
            if (s <= 2*K), break, end
            trM = eye(l);
            s = s/2;
            tempM = makewtn(s);
            trM(1:s,1:s) = tempM(1:s,1:s);
            Fx = (trM)*Fx*(trM');
        end
    end
end


end
```

### *padmatrix.m*

```
function Fx = padmatrix(X,a)
% Pads the Matrix X to the next power of 2 size
% Can be padded with zeros (a=0)
%                    ones (a=1)
%                    NaNs (a=NaN)

[l w] = size(X);

if (l == w)
    n = (2^nextpow2(l));
    if (a==0)
        Fx = zeros(n);
        Fx(1:w,1:w) = X(1:w,1:w);
    elseif (a==1)
        Fx = eye(n);
        Fx(1:w,1:w) = X(1:w,1:w);
    elseif (isnan(a))
        Fx = NaN(n);
        Fx(1:w,1:w) = X(1:w,1:w);
    end
else
    error('Simulink:actionNotTaken', 'error!, input not square, cannot be transformed');
end

end
```

***makewtn.m***

Creates wavelet transformation matrices given in equations (23) and (24).

```
function C = makewtn(N)
% Makes wavelet transformation matrix of a given size N

global h g dim

C = zeros(N);

for m = 1:N/2
    for n = 1:N
        if ( ((2*m-n) < -dim+1) || ((2*m-n) > 1) )
        else
            C(m,n) = h(n-2*(m-1));
        end
        if ((2*m-n) >= N-dim+1)
            C(m,n) = h(N-2*(m-1)+n);
        end
    end
end

for m = 1:N/2
    for n = 1:N
        if ( ((2*m-n) < -dim+1) || ((2*m-n) > 1) )
        else
            C(m+N/2,n) = g(n-2*(m-1));
        end
        if ((2*m-n) >= N-dim+1)
            C(m+N/2,n) = g(N-2*(m-1)+n);
        end
    end
end


end
```