Galerkin Wavelet Method for Global Waves in 1D

JORDI BESORA

Master’s Degree Project
Stockholm, Sweden 2004

TRITA-NA-E04091
Master's Thesis in Numerical Analysis (20 credits)
at the Scientific Computing International Master Program,
Royal Institute of Technology year 2004
Supervisor at Nada was André Jaun
Examiner was Axel Ruhe
Abstract.

The Galerkin wavelet method for solving partial differential equations has been tested for two particular cases. The results of the first case, a harmonic wave equation, corroborate the results of other authors, indicating that the method indeed works. The results of the second case, a biharmonic wave equation, show that a more accurate method for the solution of wavelet overlap integrals, involving more than two basis functions, is required.

Sammanfatning.

Galerkin wavelet metoden för lösning av partiella differentiella ekvationer har testats i två olika fall. Det ena, en harmonisk vågsekvation, har bekräftat andras resultat i och med att metoden fungerar. Det andra, en biharmonisk vågsekvation, har visat behovet av noggrannare metoder till lösning av integraler av wavelets och dess derivata när mer än två basfunktioner är inblandade.

Abstracte.

S’ha provat el mètode Galerkin wavelet per la resolució d’equacions diferencials en derivades parcials en dos casos particulars. El primer, una equació d’ona harmònica, ha corroborat resultats d’altres autors en el sentit que el mètode funciona. El segon cas, una equació d’ona biharmonica, ha posat de manifest la necessitat de mètodes més precisos per a la solució d’integrals de superposició de wavelets amb més de dues funcions base.
## Contents

1 Introduction. ................................................. 7

2 Galerkin Method and Wavelet Basis Functions .......... 8
   2.1 Galerkin Method. .................................... 8
   2.2 Wavelets. ........................................... 8
   2.3 Galerkin Wavelet Method. ............................ 11
      2.3.1 2-term Connection Coefficients .............. 12
      2.3.2 3-term Connection Coefficients .............. 14

3 Test Problems. ............................................. 15
   3.1 Harmonic Wave Equation. ............................. 15
      3.1.1 Dispersion Analysis. .......................... 17
   3.2 Biharmonic Wave Equation. ........................... 17
      3.2.1 Dispersion Analysis. .......................... 18

4 Implementation. ............................................ 20
   4.1 Connection Coefficients. ............................ 20
      4.1.1 2-term Connection Coefficients .............. 20
      4.1.2 3-term Connection Coefficients .............. 21
   4.2 Linear Systems. ...................................... 22
      4.2.1 Harmonic Wave Equation. ........................ 22
      4.2.2 Biharmonic Wave Equation. ........................ 22

5 Results ...................................................... 24
   5.1 Harmonic Wave Equation. ............................. 24
   5.2 Biharmonic Wave Equation. ............................ 28

6 Conclusions and Future Work. ............................ 30

7 Acknowledgements. ......................................... 30

A Exact Solution of the Harmonic Wave Equation. ....... 31

B Code. ....................................................... 32
   B.1 concoeff_2tuple.m .................................. 32
   B.2 moments.m .......................................... 33
   B.3 daubechies.m ...................................... 34
   B.4 testcase.m .......................................... 35
   B.5 concoeff_3tuple.m .................................. 37
   B.6 multiscale.m ....................................... 37
   B.7 a.m .................................................. 41
   B.8 testcoeff.m .......................................... 43
   B.9 coeffchecksum.m .................................... 44
   B.10 D6p9020.m .......................................... 44
1 Introduction.

The so-called mode-conversion phenomenon, which has been known for decades in plasma physics, consists of a linear coupling through which energy is transferred between two waves traveling with the same phase velocity somewhere inside a plasma. This phenomenon plays a central role in the heating and current generation with waves, but the scale between fast (long wavelength) and slow (short wavelength) waves makes it difficult to account for in anything but the simplest 1D configurations. Today, more powerful computers and sophisticated mathematical methods generally deal with the 2D geometry of a tokamak, either by a ray-tracing or with a global approach. Extending the global approach, this work aims at testing a Galerkin wavelet method to solve model equations first in 1-D, in the hope later to achieve dramatic gains in computational speed when the wave equations are solved in full 2D tokamak geometry.

The document is organized as follows. Section 2 introduces the Galerkin method, as well as wavelets, and then goes on to combine them together in the Galerkin wavelet method. Section 3 shows how to discretize a differential equation and obtain a linear system of equations in order to compute a Galerkin wavelet approximation of the solution. Harmonic and biharmonic wave equations are used as test cases. Section 4 outlines the construction of the matrices. Section 5 presents results for the harmonic equation test together with some interesting observations. Section 6 draws some conclusions.
2 Galerkin Method and Wavelet Basis Functions

2.1 Galerkin Method.

One way to discretize a differential equation

\[ Au = f \]  \hspace{1cm} (1)

is to choose a finite number of functions, and to approximate the exact solution by a combination of those trial functions. Methods based on this approach are called projection methods since the approximate solution is nothing but a projection of the exact solution onto the subspace spanned by the trial functions.

In projection methods, the approximate solution \( \tilde{u}(x) \) is written in terms of the base functions \( \phi_j \) as shown in equation 2.

\[ \tilde{u}(x) = \sum_{j=1}^{n} c_j \phi_j \]  \hspace{1cm} (2)

In 1917, the Russian engineer V.I. Galerkin\(^1\) proposed a projection method based on the weak form, or equation of virtual work. To derive the weak form one chooses a set of test functions against which to test the residual, and imposes the condition that the residual be orthogonal to the test functions. So, given an inner product \( \langle \cdot, \cdot \rangle \) the weak form can be written as

\[ \langle v, (A\tilde{u} - f) \rangle = 0 \]  \hspace{1cm} (3)

Galerkin suggested that equation 3 should be satisfied for all \( v \) of the same form as \( \tilde{u} \). That is

\[ v(x) = \sum_{i=1}^{n} a_i \phi_i \]

Since the only way to ensure this is to enforce equation 3 for each \( \phi_i \) separately, one chooses the test functions to be the same as the trial functions, and equation 3 can be written as

\[ \sum_{j=1}^{n} c_j \langle \phi_i, A\phi_j \rangle = \langle \phi_i, f \rangle \quad i = 1, \ldots, n \]  \hspace{1cm} (4)

In matrix form, this is \( Sc = f \) where \( S \) is an n-by-n matrix with elements \( s_{i,j} = \langle \phi_i, A\phi_j \rangle \), \( c = (c_1, \ldots, c_n)^T \) and \( f = (\langle \phi_1, f \rangle, \ldots, \langle \phi_n, f \rangle)^T \).

2.2 Wavelets.

Wavelets are an orthonormal basis for functions in \( L^2(\mathbb{R}) \) which have compact support or exponentially decaying support, have continuity properties that can easily be increased, have a complete basis that can easily be generated by simple recurrence relation, have very good convergence properties in the context of projection methods and their space is broken down into a family of subspaces which enable multi-resolution analysis.

\(^1\)Should actually be transcribed as Galyorkin.
Only the so-called Daubechies compactly supported wavelets are used in this paper. They were introduced by Ingrid Daubechies in 1988 using a finite set of nonzero scaling coefficients, with
\[
\sum_{k=0}^{N-1} a_k = 2
\]
where \(N\) denotes the order, or genus, of the Daubechies wavelet. The functions generated with these coefficients will have \(\text{supp}(\varphi) = [0, N - 1]\) and \((N/2 - 1)\) vanishing wavelets moments.

A “wavelet system” consists of a mother scaling function \(\varphi(x)\) and a mother wavelet function \(\psi(x)\). The scaling relation is defined as
\[
\varphi(x) := \sum_{k=0}^{N-1} a_k \varphi(2x - k) = \sum_{k=0}^{N-1} a_k \varphi_k(2x)
\]
The scaling function will also hold for \(\varphi(2x)\), and, by induction for all \(\varphi(2^j x)\), so we can write all the dilations and translations of \(\varphi\) as
\[
\varphi_{j,k}(x) = 2^j \varphi(2^j x - k)
\]
The wavelet function is defined in terms of the scaling function as
\[
\psi(x) := \sum_{k=1}^{N-2} (-1)^k a_{k+1} \varphi(2x + k)
\]
Alternatively, the scaling functions are sometimes defined as
\[
\varphi(x) := \sqrt{2} \sum_{k=0}^{N-1} h_k \varphi(2x - k)
\]
This simply means that \(a_k = \sqrt{2} h_k\), with the condition
\[
\sum_{k=0}^{N-1} h_k = \sqrt{2}
\]

Wavelets have a number of properties that are quickly summarized in the rest of this section. Denote by \(V\) the set of all scaling functions \(\{\varphi(x)\}\) and by \(W\) the set of all wavelet functions \(\{\psi(x)\}\).

1. \(\{\varphi_{j,k}\}_{j \geq 0, k \in \mathbb{Z}}\) is an orthonormal basis for \(L^2(\mathbb{R})\).
2. \(V_{j+1} = V_j \bigoplus W_j\).
3. \(L^2(\mathbb{R}) = \text{clos}_{L^2} \left( V_0 \bigoplus_{j=0}^\infty W_j \right)\).
4. \(\{\varphi_{0,k}, \psi_{j,k}\}_{j \geq 0, k \in \mathbb{Z}}\) is an orthonormal basis for \(L^2(\mathbb{R})\).
5. \(\int_{-\infty}^{\infty} \varphi(x) \, dx = 1\).
2.2 Wavelets

6. $\sum_{k \in \mathbb{Z}} \varphi_{0,k} = 1.$
7. $\int_{-\infty}^{\infty} \psi(x)x^k dx = 0 : k = 0, \ldots, N/2 - 1.$
8. $\{x^k\}_{k=0}^{N/2-1} \in V_N$

The multi-resolution analysis mentioned before is based on property no. 2. A multi-resolution analysis is a nested sequence $V_0 \subset V_1 \subset \cdots \subset L^2(\mathbb{R})$ satisfying the following properties:

1. $\bigcap_{j \in \mathbb{Z}} V_j = 0.$
2. $\text{clos}_{L^2} \left( \bigcup_{j \in \mathbb{Z}} V_j \right) = L^2(\mathbb{R}).$
3. $f(x) \in V_j \iff f(2x) \in V_{j+1}$
4. $\exists \varphi \in V_0$ such that $\{\varphi_{0,k}(x) = \varphi(x - k)\}_k \in \mathbb{Z}$ forms a Riesz basis for $V_0$

The wavelet expansion of a function $f(x) \in L^2(\mathbb{R})$ is of the form

$$f(x) = \sum_{l \in \mathbb{Z}} c_{0l} \varphi_{0l}(x) + \sum_{j=0}^{\infty} \sum_{k \in \mathbb{Z}} c_{jk} \psi_{jk}(x)$$

in other words

$$f(x) \in \left\{ V_0 \oplus W_0 \oplus W_1 \oplus W_2 \cdots \oplus W_\infty \right\}_{L^2(\mathbb{R})}$$

Note that since $V_{j+1} = V_j \oplus W_j$ one can also write

$$f(x) \in \left\{ V_j \oplus W_j \oplus W_{j+1} \oplus W_{j+2} \cdots \oplus W_\infty \right\}_{L^2(\mathbb{R})}$$

and still have an exact representation of $f(x)$. If this representation is truncated at level $j$, what is left is an approximation of $f(x)$ at resolution $j$ (see Mallat [6] for details)

$$\tilde{f}(x) = \sum_{l \in \mathbb{Z}} c_{jl} \varphi_{jl}(x) \quad (5)$$

To solve PDEs we will need an expansion for $f(x)$, but also for the derivatives of $f(x)$. Daubechies showed that $\exists \lambda > 0$ such that a wavelet of genus $N$ has $\lambda(N/2 - 1)$ continuous derivatives; for small $N$, $\lambda \geq 0.55$. Define the following notation for derivation of a function:

$$\varphi^d_l := \frac{\partial^d \varphi_l}{\partial x^d}$$

Taking $d$ derivatives of equation 5 gives

$$f^d(x) = \sum_{l \in \mathbb{Z}} c_{l} \varphi^d_l(x)$$
Approximate $\varphi^d_l(x)$ as

$$\varphi^d_l(x) = \sum_m \lambda_m \varphi_m(x)$$

where

$$\lambda_m = \int_{-\infty}^{\infty} \varphi^d_l(x) \varphi_m(x) dx$$

which is a so called 2-term connection coefficient, a functional that will be introduced immediately, when wavelet functions are used as basis functions for the Galerkin method.

### 2.3 Galerkin Wavelet Method.

When the base functions in Galerkin’s method are piecewise polynomials, the method is called finite element method. By analogy, it is called Galerkin wavelet method when the base functions are wavelets.

Approximate the exact solution with a finite number of wavelet functions $\varphi$, as in equation 5

$$\bar{u}(x) = \sum_{l=1}^{N_x} u_l \varphi_l$$

Define the inner product $\langle \varphi_k, \varphi_l \rangle$ as

$$\langle \varphi_k, \varphi_l \rangle = \int_{-\infty}^{\infty} \varphi_k \varphi_l dx$$

and take the $\varphi$ functions to be wavelets $\varphi$. The system $Sc = f$ obtained by applying Galerkin’s method to equation 1, will have elements $s_{k,l} = \langle \varphi_k, A \varphi_l \rangle$ and $f = (\langle \varphi_1, f \rangle, ..., \langle \varphi_{N_x}, f \rangle)^T$.

If the operator $A$ is linear with constant coefficients, the elements $s_{k,l}$ will contain terms of the form

$$\Omega^{d_1,d_2}_{k,:} = \int_{-\infty}^{\infty} \varphi^d_k \varphi^d_l dx$$

which are called 2-term connection coefficients. The super-indices $d_1$ and $d_2$ denote derivatives of the wavelet functions $\varphi$.

If the operator $A$ either is nonlinear or has variable coefficients, or both, then the connection coefficients involve a larger number of terms, depending on the characteristics of $A$. In its general form, an n-term connection coefficient is defined as

$$\Omega^{d_1,...,d_n}_{k_1,...,k_n} := \int_{-\infty}^{\infty} \varphi^d_{k_1} \cdots \varphi^d_{k_n} dx = \int_{-\infty}^{\infty} \prod_{i=1}^{n} \varphi^d_{k_i} dx$$

This paper will use at most 3-term connection coefficients

$$\Omega^{d_1,d_2,d_3}_{k,j,m} = \int_{-\infty}^{\infty} \varphi^d_k \varphi^d_j \varphi^d_m dx$$

Note that, since translations of a wavelet $\varphi_k(x)$ are orthonormal, we have

$$\Omega^{0,0}_{k-l} = \int \varphi_k \varphi_l dx = \delta_{l-k}$$
which is 1 only when \( k = l \) and 0 otherwise, and

\[
\Omega^{0,0}_{l-k,m-k} = \int \varphi_k \varphi_l \varphi_m \, dx = \delta_{l-k,m-k}
\]  

(10) which is 1 only when \( k = l = m \) and 0 otherwise.

By a change of variables, any 2-term connection coefficient \( \Omega^{d_1,d_2}_{k,l} \) can be turned into \( \Omega^{d_1,d_2}_{0,l-k} := \Omega^{d_1,d_2}_{l',m'} \), and any \( \Omega^{d_1,d_2,d_3}_{k,l,m} \) can be turned into \( \Omega^{d_1,d_2,d_3}_{0,0,0} \), where \( l' = l - k \) and \( m' = m - k \). For simplicity, \( \Omega^{d_1,d_2}_{l,m} \) and \( \Omega^{d_1,d_2,d_3}_{l,m} \) will be used throughout this document.

The Daubechies wavelet functions \( \varphi \) cannot be represented in closed form for \( N > 2 \) so analytic calculation of the integrals is not an option, and numerical quadrature is often inaccurate due to the highly oscillating nature of the wavelet basis functions.

Latto et al. [1] provide an alternative method for computing wavelet connection coefficients, but it is designed for unbounded intervals. In order to deal with bounded intervals, Romine et al. [5] and Lu et al. [3] provide two different methods. The approach Romine et al. take in their paper is to modify the connection coefficients obtained by Latto et al. at the boundaries, while Lu et al. modify the interval instead, and use the connection coefficients as obtained by Latto et al. The method used in this document is the one described by Lu et al. [3] which basically consists on expanding the domain by adding \( N - 1 \) points to the left and \( N - 1 \) points to the right, so that now the original boundaries are actually inside the new domain, sufficiently away from the new boundaries, or “Fictitious Boundaries” as Lu et al. refer to them, to remain unaffected by the inaccuracy introduced by using connection coefficients that were computed for unbounded domains.

Let’s now take a look at how to compute 2-term and 3-term connection coefficients.

### 2.3.1 2-term Connection Coefficients

Let \( A^{d_1,d_2} \) be a column vector which 2N - 3 components are the connection coefficients defined as

\[
\Omega^{d_1,d_2}_{l,m} = \int_{-\infty}^{\infty} \varphi^{d_1}_{l} \varphi^{d_2}_{m} \, dx
\]

Taking derivatives of the scaling function \( \varphi(x) \), assuming it is \( d \) times differentiable, gives

\[
\varphi^{d}(x) = 2^{d} \sum_{k=0}^{N-1} a_k \varphi_{k}(2x)
\]

Substituting it into the definition of the 2-term connection coefficient and changing variables, Latto et al. [1] arrive at

\[
\Omega^{d_1,d_2}_{l,m} = 2^{d_1+d_2-1} \sum_{p,q=0}^{2^{d_1}+2^{d_2}} \int_{-\infty}^{\infty} \varphi^{d_1}_{p}(x)\varphi^{d_2}_{q}(x) dx
\]

which, after simplification and considering it for all \( \Omega^{d_1,d_2}_{l,m} \), gives a system of linear equations with \( A^{d_1,d_2} \) as unknown vector:

\[
TA^{d_1,d_2} = \frac{1}{2^{(d-1)}} A^{d_1,d_2}
\]

where

\[
d := d_1 + d_2
\]
and

\[ T_{lq} = \sum_p a_p a_{q-2l+p} \]

These are the so-called scaling equations. But this system is homogeneous, and thus does not have a unique nonzero solution. In order to make the system inhomogeneous, one equation is added which is derived from the moment equations of the scaling function \( \varphi \). This is the normalization equation

\[ d! = (-1)^d \sum_l M_l^d A_l^{0,d} \]

Coefficients of the form \( A_l^{0,d} \) are easily obtained from \( A_l^{d_1,d_2} \) by integrating by parts \( d_1 \) times.

\[ A_l^{d_1,d_2} = \int_{-\infty}^{\infty} \varphi_{l_1}^{d_1} \varphi_{l_2}^{d_2} dx \]

\[ = \left[ \varphi_{l_1}^{d_1-1} \varphi_{l_2}^{d_2} \right]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \varphi_{l_1}^{d_1-1} \varphi_{l_2}^{d_2+1} dx \]

Because of the compact support wavelet basis functions exhibit, this becomes

\[ A_l^{d_1,d_2} = - \int_{-\infty}^{\infty} \varphi_{l_1}^{d_1-1} \varphi_{l_2}^{d_2+1} dx \]

and after \( d_1 \) integrations

\[ A_l^{d_1,d_2} = (-1)^{d_1} \int_{-\infty}^{\infty} \varphi_{l_1}^{d_2+d_3} dx = (-1)^{d_1} A_l^{0,d} \]

The moments \( M_k^l \) of \( \varphi_l \) are defined as

\[ M_k^l = \int_{-\infty}^{\infty} x^k \varphi_l(x) dx \]

The scaling function \( \varphi(x) \) is normalized by definition so that \( M_0^0 = 1 \). By induction on \( k \), Latto et al. derive an explicit formula to compute the moments:

\[ M_l^j = \frac{1}{2(2^j - 1)} \sum_{k=0}^{j} \binom{j}{k} (-1)^{j-k} \sum_{l=0}^{k-1} \binom{k}{l} M_l^0 \left( \sum_{i=0}^{N-1} a_i k^{-l} \right) \]

where the \( a_i \) are the Daubechies wavelet coefficients.

Finally, the system will be

\[ \left( T - \frac{1}{M^d} I \right) A^{d_1,d_2} = \begin{pmatrix} 0 \\ d! \end{pmatrix} \]

where \( M^d \) is a row vector with all the \( M_l^d \). This is an over-determined \((2N-3)+1\)-by-\((2N-3)\) system which has as its unique solution the exact solution of the connection coefficients \( \Omega_l^{d_1,d_2} \).
2.3 Galerkin Wavelet Method

2.3.2 3-term Connection Coefficients

The computation of 3-term connection coefficients is somewhat more involved. The resulting $\Lambda_{lm}^{d_1 d_2 d_3}$ is not a vector, but a $(2N - 3)$-by-$(2N - 3)$ matrix. Besides, the matrix $T$ formed by the scaling equations is both homogeneous, like in the 2-term case, and rank deficient, so now it’s not enough with just one more equation to make it inhomogeneous, but we need $d + 1$ equations. The system will then be composed of three parts, namely the scaling equations, the moment equations and the so called normalization equation.

By construction, not all the elements of $\Lambda_{lm}^{d_1 d_2 d_3}$ will be nonzero, but only those for which

$$2 - N \leq l \leq N - 2, \quad 2 - N \leq m \leq N - 2, \quad \text{and} \quad |l - m| \leq N - 2$$

There are $M = 3N^2 - 9N + 7$ such elements. Denote the set of $M$ pairs $(l, m)$ by $S$, and select an arbitrary fixed bijection of the set $S$ onto the integers $\{1 \ldots M\}$. Let now $\Lambda_{lm}^{d_1 d_2 d_3}$ be a column vector which $M$ components are the nonzero elements of $\Lambda_{lm}^{d_1 d_2 d_3}$.

Again Latto et al. [1] give a system of linear equations with $\Lambda_{lm}^{d_1 d_2 d_3}$ as unknown vector:

$$T \Lambda_{lm}^{d_1 d_2 d_3} = \begin{pmatrix} 1 \end{pmatrix} \Lambda_{lm}^{d_1 d_2 d_3}$$

where

$$d := d_1 + d_2 + d_3$$

and

$$T(l,m):(q,r) = \sum_p a_p a_q - 2l + p a_r - 2m + p$$

These are the scaling equations, a system that is now both homogeneous and rank deficient. To compensate for the rank deficiency, $d$ homogeneous equations are added. One can always go from $\Lambda_{lm}^{d_1 d_2 d_3}$ to $\Lambda_{lm}^{0,b,d-1}$ integrating by parts or, equivalently, using the formula Latto et al. give in pg. 16 of [1]. For simplicity, $\Lambda_{lm}^{0,b,d_3}$ will be used instead. Then one can obtain $d_2$ homogeneous equations

$$\sum_l M_l^b \Lambda_{lm}^{0,b,d_2 d_3} = 0$$

where $0 \leq b \leq d_2 - 1$ and $m = 0$ if $d_3$ is even or $m = 1$ if $d_3$ is odd, and $d_3$ homogeneous equations

$$\sum_m M_m^c \Lambda_{lm}^{0,d_2 d_3} = 0$$

where $0 \leq c \leq d_3 - 1$ and $l = 0$ if $d_2$ is even or $l = 1$ if $d_2$ is odd.

The system is not rank deficient anymore, but it is still homogeneous. Let’s add the normalization equation

$$d_2! d_3! = \sum_{lm} M_l^{d_2} M_m^{d_3} \Lambda_{lm}^{0,d_2 d_3}$$

Finally, the system obtained is

$$\begin{pmatrix} T & -\frac{1}{d_2!} I \\ M_l^b \\ M_m^c \\ M_l^{d_2} M_m^{d_3} \end{pmatrix} \Lambda_{lm}^{d_1 d_2 d_3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ d_2! d_3! \end{pmatrix} \quad \quad (13)$$
where $M_{bl}^i$ and $M_{cm}^c$ are blocks with $d_2$ and $d_3$ rows respectively containing the moment equations, and the $M_{bl}^{d_2}M_{cm}^{d_3}$ is a row vector with the $M_{bl}^{d_2}M_{cm}^{d_3}$ products for all $lm$ corresponding to nonzero $A_{l,m}$. The moments $M$ are obtained by equation 11.

This system is ill-conditioned, and has a condition number that grows as the wavelets resolution increases. To test the accuracy of the resulting connection coefficients Restrepo et al. [2] propose a relation they call the “check-sum procedure” which says that the column sum of a 3-term matrix must equal a corresponding 2-term vector component by component. As an example they suggest

$$
\sum_l \Omega_{l,m}^{1,0,0} = \Omega_{m}^{1,0}
$$

A more general form of this test can be found in equation 23 in Latto et al. [1] where they give proof that

$$
\Omega_{l,m}^{d_2d_3} = \sum_{i=\max(l,m)}^{\min(l,m)+N-2} \Omega_{l-i,m-i}^{0,d_2d_3}
$$

(14)

3 Test Problems.

In order to test the Galerkin wavelet method, two different setups were used: one with a simple harmonic wave equation and one with a more complex biharmonic wave equation.

The first one is linear with constant coefficients, leading to 2-term connection coefficients. The second is still linear but involves an inhomogeneous coefficient, which requires both 2-term and 3-term connection coefficients.

3.1 Harmonic Wave Equation.

Take the harmonic wave equation

$$
\frac{\partial^2 f}{\partial x^2} + Df = 0 \quad \text{with} \quad \begin{cases} 
  f(0) = 0 \\
  f(1) = 1 
\end{cases}
$$

(15)

which can be solved analytically (see Appendix A) and yields

$$
f(x) = \frac{\sin(x\sqrt{D})}{\sin(\sqrt{D})}
$$

To compute the weak form, as required by the Galerkin method, multiply by a base function $\varphi_k$ and integrate over the domain:

$$
\int \varphi_k \left[ \frac{\partial^2 f}{\partial x^2} + Df \right] dx = 0
$$

Discretize space.

$$
f(x) = \sum_l f_l \varphi_l(x) \quad \text{where} \quad f_l = \langle f, \varphi_l \rangle
$$

Plugging the wavelet expansion of $f(x)$ into the weak form gives

$$
\int D \varphi_k \sum_l f_l \varphi_l(x) dx + \int \varphi_k \frac{\partial^2}{\partial x^2} \left( \sum_l f_l \varphi_l(x) \right) dx = 0
$$
which can be rewritten as
\[ \sum_l f_l \left( D \int \varphi_k \varphi_l dx + \int \varphi_k \varphi''_l dx \right) = 0 \]

So, the equation to be solved is
\[ \sum_l f_l \left( D \delta_{k,l} + \Omega^{0,2}_{l-k} \right) = 0 \] (16)

The indexes \( l \) and \( k \) have to cover the whole domain, which as stated in Section 2.3 has been expanded according to [3], so if the original domain is discretized with \( N_x \) functions and \( N-1 \) functions have to be added both at the left and the right of it, the new domain, delimited by the new “Fictitious Boundary Conditions” will expand from \(- (N-1)\) to \( N_x-1 + (N-1)\), totaling \( N_x + 2(N-1) \) functions.

Then, \(- (N-1) \leq l \leq N_x-1 + (N-1)\) and \(- (N-1) \leq k \leq N_x-1 + (N-1)\), so that equation 16 will form an \((N_x + 2(N-1))\)-by-\((N_x + 2(N-1))\) linear system
\[ A f = 0 \] (17)

where \( f \) is a column vector containing all the \( f_l \).

Let’s now impose the boundary conditions \( f(0) = 0 \) and \( f(1) = 1 \). Applying the method described in [3], the first and last rows in equation 17 will be replaced by the following equations.

For \( f(0) \) we have
\[ \sum_l f_l \varphi_l(0) = 0 \]

To avoid having to deal with the scaling functions, take the inner product of this equation with \( \varphi_k \):
\[ \int \sum_l f_l \varphi_l(0) \varphi_k(0) dx = 0 \]
which can be reorganized as
\[ \sum_l f_l \int \varphi_l(0) \varphi_k(0) dx = 0 \]

For the left boundary condition this leads to
\[ \sum_l f_l \delta_{l,k}(0) = 0 \] (18)
which constitutes the new first row in equation 17.

At the right boundary, for \( f(1) \) we have
\[ \sum_l f_l \varphi_l(1) = 1 \]

Again, to avoid having to deal with the scaling functions, take the inner product of this equation with \( \varphi_k \):
\[ \int \sum_l f_l \varphi_l(1) \varphi_k(1) dx = 1 \]
3 TEST PROBLEMS.

3.2 Biharmonic Wave Equation.

which can be reorganized as

$$\sum_l f_l \int \varphi_l(1) \varphi_k(1) dx = 1$$

So for the right boundary condition

$$\sum_l f_l \delta_{l,k}(1) = 1$$ (19)

which constitutes the new last row in equation 17.

The indexes \(l\) and \(k\) in equations 18 and 19 do not correspond to their position in matrix \(A\), but actually the \(\delta\) function has to be evaluated at 0 and 1 respectively. So for the left boundary, \(l = k = 0\) and for the right boundary \(l = k = N_x - 1\), regardless of which row they occupy in the matrix. This will be further illustrated in the implementation of the system, in Section 4.2.1.

3.1.1 Dispersion Analysis.

Equation 15 has characteristic polynomial \(k^2 + D = 0\), with roots \(k = \pm i \sqrt{D}\). Since the wavelength \(\lambda\) is

$$\lambda = \frac{2\pi}{k}$$

the number of wavelengths \(\eta\) within the domain \([0, 1]\) will be

$$\eta = \frac{1 - 0}{\lambda} = \frac{\sqrt{D}}{2\pi}$$

3.2 Biharmonic Wave Equation.

Introducing a test model to account for linear mode-conversion, define the biharmonic wave equation as

$$\begin{cases} \frac{\partial}{\partial x} \left[ a(x) \frac{\partial u}{\partial x} \right] + u + v = 0 & u(-L) = 0; u(L) = 1 \\ \frac{\partial^2 v}{\partial x^2} - u = 0 & v(-L) = 0; v(L) = 0 \\ a(x) = \varepsilon \left[ \frac{x}{\sqrt{1+x^2}} + \frac{1}{x^{1+\varepsilon}} \right] \end{cases}$$ (20)

with corresponding weak form

$$\begin{cases} \int \varphi_k u \, dx + \int \varphi_k v \, dx + \int \varphi_k \frac{\partial}{\partial x} \left[ a(x) \frac{\partial u}{\partial x} \right] dx = 0 \\ \int \varphi_k \frac{\partial^2 v}{\partial x^2} \, dx - \int \varphi_k u \, dx = 0 \end{cases}$$

Discretize space.

$$u(x) = \sum_l u_l \varphi_l(x) \quad \text{where} \quad u_l = \langle u, \varphi_l \rangle$$

using corresponding expansions for \(v(x)\) and \(a(x)\), yields

$$\begin{cases} \int \varphi_k \sum_l u_l \varphi_l dx + \int \varphi_k \sum_m v_l \varphi_m dx + \int \varphi_k \left( \sum_m a_m \varphi_m' \right) \left( \sum_l u_l \varphi_l' \right) dx = 0 \\ -\int \varphi_k \sum_l u_l \varphi_l' dx + \int \varphi_k \sum_l v_l \varphi_l'' dx = 0 \end{cases}$$
3.2 Biharmonic Wave Equation

Which can be rewritten as
\[
\begin{cases}
\sum_l u_l \int \varphi_k \varphi_l \, dx + \sum_l v_l \int \varphi_k \varphi_l' \, dx + \\
\sum_l u_l \sum_m a_m \left( \int \varphi_k \varphi_m' \, dx \right) + \sum_l v_l \int \varphi_k \varphi_m \, dx = 0
\end{cases}
\]

or
\[
\begin{cases}
\sum_l u_l \left( \delta_{k,l} + \sum_m a_m \left( \Omega^{0,1,1}_{k,l,m} + \Omega^{0,2,0}_{k,l,m} \right) \right) + \sum_l v_l \delta_{k,l} = 0 \\
- \sum_l u_l \delta_{k,l} + \sum_l v_l \Omega^{0,2}_{k,l} = 0
\end{cases}
\]

(21)

Or introducing \( S = \{ \delta_{k,l} + \sum_m a_m \left( \Omega^{0,1,1}_{k,l,m} + \Omega^{0,2,0}_{k,l,m} \right) \}, \Delta = \{ \delta_{k,l} \} \) and \( C = \{ \Omega^{0,2}_{k,l} \}, \)
\[
\begin{pmatrix}
S & \Delta \\
\Delta & C
\end{pmatrix}
\begin{pmatrix}
u \\
v
\end{pmatrix}
= \begin{pmatrix}
0 \\
0
\end{pmatrix}
\]

(22)

where \( u \) is a column vector containing all the \( u_l \) and \( v \) all the \( v_l \).

The domain to be considered here, after expanding the original one with \( N - 1 \) functions in each direction, will go from \(- (N - 1) - L \) to \( L + (N - 1) \). The original domain is discretized with \( N_x \) functions, so the new system of equations will have \( N_x + 2(N - 1) \) equations and the same amount of unknowns. But that is just for each block of the matrix in equation 22, so in total there will be twice as many, namely \( 2 \left( N_x + 2(N - 1) \right) \) complex equations and unknowns.

The boundary conditions are imposed in the exact same way as in the harmonic wave equation, but extra care has to be taken with the indexes \( l \) and \( k \). Remove the first and last rows of each and all four blocks in equation 22 and impose boundary conditions on \( u \) using the \( S \) block, and on \( v \) using the \( C \) block.

Indexes \( l,k \) for the boundary conditions on \( u \) will be \( l = k = 0 \) for the left boundary, and \( N_x - 1 \) for the right boundary, and the corresponding equations will occupy the first and last row in the \( S \) block, respectively.

For the boundary conditions on \( v \), \( l = k = N_x - 1 + 3(N - 1) \) for the left boundary, and \( l = k = 2(N_x - 1) + 3(N - 1) \), and the corresponding equations will occupy the first and the last row in the \( C \) block, respectively.

Note that the identity matrix blocks will lose their first and last row as well, which will be replaced by zeros. The whole procedure will be further illustrated in the implementation of the system, in Section 4.2.2.

3.2.1 Dispersion Analysis.

Figure 1 shows that the function \( a(x) \) has a singularity at \( x = 0 \). To perform a local analysis, \( a(x) \) is assumed to remain constant and will appear finally as a parameter affecting the wavelength.

Equation 20 has characteristic polynomials
\[
\begin{cases}
\lambda t^2 u + u + v = 0 \\
t^2 v - u = 0
\end{cases}
\]

Substituting the second equation \( v = \frac{u}{t^2} \) into the first one gives
\[
\lambda t^2 u + u + \frac{u}{t^2} = 0
\]
3 TEST PROBLEMS.

3.2 Biharmonic Wave Equation.

Multiplying by $t^2$

$$at^4 u + t^2 u + u = 0$$

so

$$at^4 + t^2 + 1 = 0$$

giving

$$t^2 = \frac{-1 \pm \sqrt{1 - 4a}}{2a}$$

Since $\sqrt{1 - x} \approx 1 - \frac{x}{2}$ for small $x$, one can write

$$t_1^2 \approx \frac{-1 + 1 - 2a}{2a} \approx -1$$

$$t_2^2 \approx \frac{-1 - 1 + 2a}{2a} \approx 1 - \frac{1}{a}$$

These will lead to wavelengths $\lambda = \frac{2\pi}{t}$

$$\lambda_1 = \frac{2\pi}{t_1} \approx \frac{2\pi}{1}$$

$$\lambda_2 = \frac{2\pi}{t_2} \approx \frac{2\pi}{1 - \frac{1}{a}}$$

When $a \rightarrow 0$, $\lambda_2 \rightarrow 0$ leading to a slow wave, while $\lambda_1 = 2\pi$ is the fast wave.
4 Implementation.

4.1 Connection Coefficients.

In order to solve the test problems, 2-term and 3-term connection coefficients need to be precomputed using the algorithms described in Sections 2.3.1 and 2.3.2. The examples are given for Daubechies wavelets of genus D6.

A Fortran code written by Restrepo et al. [2] exists to compute 2-term and 3-term connection coefficients using periodized wavelets. It is based on Latto et al. [1] and has been used for comparison to build the MatLab code used in this work.

4.1.1 2-term Connection Coefficients

Using indexes \( l, q = 1 \ldots 9 \) instead of \( l, q = -4 \ldots 4 \), equation 12 can be written in MatLab as

\[
\begin{bmatrix}
T_{1,1} - c & T_{1,2} & T_{1,3} & T_{1,4} & T_{1,5} & T_{1,6} & T_{1,7} & T_{1,8} & T_{1,9} \\
T_{2,1} & T_{2,2} - c & T_{2,3} & T_{2,4} & T_{2,5} & T_{2,6} & T_{2,7} & T_{2,8} & T_{2,9} \\
T_{3,1} & T_{3,2} & T_{3,3} - c & T_{3,4} & T_{3,5} & T_{3,6} & T_{3,7} & T_{3,8} & T_{3,9} \\
T_{4,1} & T_{4,2} & T_{4,3} & T_{4,4} - c & T_{4,5} & T_{4,6} & T_{4,7} & T_{4,8} & T_{4,9} \\
T_{5,1} & T_{5,2} & T_{5,3} & T_{5,4} & T_{5,5} - c & T_{5,6} & T_{5,7} & T_{5,8} & T_{5,9} \\
T_{6,1} & T_{6,2} & T_{6,3} & T_{6,4} & T_{6,5} & T_{6,6} - c & T_{6,7} & T_{6,8} & T_{6,9} \\
T_{7,1} & T_{7,2} & T_{7,3} & T_{7,4} & T_{7,5} & T_{7,6} & T_{7,7} - c & T_{7,8} & T_{7,9} \\
T_{8,1} & T_{8,2} & T_{8,3} & T_{8,4} & T_{8,5} & T_{8,6} & T_{8,7} & T_{8,8} - c & T_{8,9} \\
T_{9,1} & T_{9,2} & T_{9,3} & T_{9,4} & T_{9,5} & T_{9,6} & T_{9,7} & T_{9,8} & T_{9,9} - c \\
M_1^d & M_2^d & M_3^d & M_4^d & M_5^d & M_6^d & M_7^d & M_8^d & M_9^d
\end{bmatrix}
\]

where the matrix coefficients \( T_{l,q} \) are given in terms of the Daubechies wavelet coefficients.

\[
T_{l,q} = \sum_p a_p a_{q-2l+p} , \quad c = -\frac{1}{2^{(d-1)}} \quad \text{and} \quad d = d_1 + d_2
\]

To avoid MatLab’s “out of range” error messages, the vector of coefficients \( a_k \) can be copied into a bigger array padding the beginning and the end with zeros. Now, the new \( a \) vector will be

\[
a = \{0, \ldots, 0, a_{-4}, a_{-3}, a_{-2}, a_{-1}, a_0, a_1, a_2, a_3, a_4, 0, \ldots, 0\}
\]

Rather than having different indexes for the matrix \( A \) and the vector \( a \), which would complicate the code, the same approach was taken for the matrix, which was embedded into a bigger matrix and padded with zeros.

Once all the sums \( T_{l,q} \) are computed and the matrix is formed with the Moment equation in place, the extra zeros are taken away and we are left only with a \((2N - 3 + 1)\)-by-\((2N - 3)\) matrix to compute the \(2N - 3\) unknown connection coefficients.

The unknowns and the right hand side:

\[
A_{d_1,d_2}^{d_1,d_2} = \begin{bmatrix}
\Omega_4^{d_1,d_2} \\
\vdots \\
\Omega_9^{d_1,d_2}
\end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix}
0 \\
\vdots \\
0
\end{bmatrix}
\]

To solve this system MatLab’s \textit{backslash} solver was used. Restrepo et al. used a modification of the Moment vector by Chebichev polynomials in order to cope with the high condition
number of the resulting matrix; for 2-term connection coefficients, the condition number will not be so high as to compromise the accuracy of the solution except for high resolutions.

### 4.1.2 3-term Connection Coefficients

For the computation of 3-term connection coefficients the code provided by Restrepo et al. [2] was used. An outline of the implementation is given here.

Recall from Section 2.3.2 that $\Lambda_{lm}^{d_1,d_2,d_3}$ is not a vector but a matrix which has only $M$ nonzero elements. These nonzero elements can be reorganized in a vector $\Lambda^{d_1,d_2,d_3}$ which can, integrating by parts, be transformed into $\Lambda^{0,b,d-b}$. It is this vector that the system has to be solved for, and for simplicity it will be called $\Lambda^{0,d_2,d_3}$, where $d_3$ and $d_3$ are actually $b$ and $d-b$, and not the original $d_2$ and $d_3$ anymore.

For $D6$, $M = 61$, and the system given in equation 13 will then be over-determined with a $(61 + d + 1)$-by-61 matrix.

$$
\begin{bmatrix}
T_{1,1} - c & T_{1,2} & T_{1,3} & \cdots & T_{1,59} & T_{1,60} & T_{1,61} \\
T_{2,1} & T_{2,2} - c & T_{2,3} & \cdots & T_{2,59} & T_{2,60} & T_{2,61} \\
T_{3,1} & T_{3,2} & T_{3,3} - c & \cdots & T_{3,59} & T_{3,60} & T_{3,61} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
T_{59,1} & T_{59,2} & T_{59,3} & \cdots & T_{59,59} - c & T_{59,60} & T_{59,61} \\
T_{60,1} & T_{60,2} & T_{60,3} & \cdots & T_{60,60} - c & T_{60,61} & T_{60,61} \\
M_1^0 & M_1^0 & M_2^0 & \cdots & M_1^0 & M_2^0 & \cdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
M_{d_2}^0 & M_{d_2}^0 & M_{d_3}^0 & \cdots & M_{d_2}^0 & M_{d_3}^0 & \cdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
M_{d_2}^0 M_{d_3}^0 M_{d_3}^0 & M_{d_2}^0 M_{d_3}^0 & M_{d_2}^0 & \cdots & M_{d_2}^0 M_{d_3}^0 & M_{d_2}^0 & \cdots \\
\end{bmatrix}
$$

where

$$
T_{i,q} = \sum_p a_p a_{q - 2l + p} a_{r - 2m + p} \quad , \quad c = -\frac{1}{2(d-1)} \quad \text{and} \quad d = d_2 + d_3
$$

Note that the coefficients of neither the moment equations ($M_l^b$ and $M_m^r$) nor the normalization equation ($M_l^{d_2} M_m^{d_3}$) are placed in the right positions in this example. They have to be placed according to the index mapping from the $(l,m)$ pairs to the $M$-vector \{1...M\} mentioned in Section 2.3.2.

The unknowns and the right hand side:

$$
\Lambda^{0,d_2,d_3} = \begin{bmatrix}
\Omega_1^{0,d_2,d_3} \\
\vdots \\
\Omega_{61}^{0,d_2,d_3}
\end{bmatrix}
$$

and

$$
b = \begin{bmatrix}
0 \\
\vdots \\
0 \\
d_2 d_3 !
\end{bmatrix}
$$

Restrepo et al. [2] use a modification of the Moment vectors by Chebichev polynomials in order to cope with the high condition number of the resulting matrix, but as they point out in their paper, it doesn’t help.
4.2 Linear Systems.

After applying the wavelet Galerkin method to the test problems, and imposing boundary conditions following [3], let’s now look at how to build the resulting matrices after discretizing with \( N_x = 2^j \) scaling functions, where \( j \) is the wavelet resolution. In these examples the wavelets used are \( D6 \), which 2-term connection coefficient is a vector with 9 elements, with \( l - m = -4, \ldots, 0, \ldots, 4 \), and which 3-term connection coefficient is a 9-by-9 matrix with \( l - k, m - k = -4, \ldots, 0, \ldots, 4 \).

4.2.1 Harmonic Wave Equation.

After discretizing equation 15 with \( N_x = 2^j \) scaling functions, where \( j \) is the wavelet resolution, one can now build the system \( A \mathbf{f} = \mathbf{b} \) of equations to solve. \( A \) is a \((2N - 3)\)-diagonal \((N_x + (N - 1))\)-by-\((N_x + (N - 1))\) matrix. In this example, with \( D6 \), the upper rows of the 9-diagonal matrix \( A \) will be

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\
\Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 & \cdots \\
\Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 & \cdots \\
\Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 & \cdots \\
0 & \Omega_{-2}^{0,2} & \Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 & \cdots \\
& & & \ddots & & & & & \ddots \\
\end{bmatrix}
\]

And the lower rows of \( A \) will be:

\[
\begin{bmatrix}
\cdots & 0 & \Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 \\
\cdots & 0 & \Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 \\
\cdots & 0 & \Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 \\
\cdots & 0 & \Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 \\
\cdots & 0 & \Omega_{-3}^{0,2} & \Omega_{-2}^{0,2} & \Omega_{-1}^{0,2} & D + \Omega_0^{0,2} & \Omega_1^{0,2} & \Omega_2^{0,2} & \Omega_3^{0,2} & \Omega_4^{0,2} & 0 \\
\cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

And finally the unknowns and the right hand side:

\[
\mathbf{f} = \begin{bmatrix} f_{-N+1} \\ \vdots \\ f_{N-1+N_x} \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}
\]

Note how the first and last rows of matrix \( A \) and vector \( \mathbf{b} \) have been modified by the boundary conditions.

4.2.2 Biharmonic Wave Equation.

After discretizing equation 20 with \( N_x = 2^j \) scaling functions, where \( j \) is the wavelet resolution, the system of equation 22 can now be built.
Block $S$ is a $(2N-3)$-diagonal $(N_x+(N-1))$-by-$(N_x+(N-1))$ matrix. In this example, with $D6$, the upper rows of the 9-diagonal block matrix $S$ will be

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & \cdots
\end{bmatrix}
\]

And the lower rows of $A$ will be:

\[
\begin{bmatrix}
\cdot & \cdot & \cdot
\end{bmatrix}
\]

Let's now build the $C$ block, which is a $(2N-3)$-diagonal $(N_x+(N-1))$-by-$(N_x+(N-1))$ matrix. The upper rows of the 9-diagonal block matrix $C$ will be

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & \cdots
\end{bmatrix}
\]

And the lower rows of $C$ will be:

\[
\begin{bmatrix}
\cdot & \cdot & \cdot
\end{bmatrix}
\]

When imposing boundary conditions, the $\Delta$ block matrices are modified as follows

\[
\Delta = \begin{bmatrix}
0 & 0 \\
1 & 1 \\
\cdot & \cdot \\
0 & 0
\end{bmatrix}
\]
And finally the unknowns and the right hand side:

\[
\begin{bmatrix}
  u_{-N+1} \\
  \vdots \\
  u_{N-1+N_x} \\
  v_{-N+1} \\
  \vdots \\
  v_{N-1+N_x}
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
  0 \\
  \vdots \\
  0 \\
  1 \\
  \vdots \\
  0
\end{bmatrix}
\]

Note how the first and last rows of block matrix \( S \) and block matrix \( C \) have been modified by the boundary conditions for \( u \) and \( v \) respectively. The right hand side vector \( b \) has also been modified accordingly.

## 5 Results

All results presented here were obtained on a Sun Blade 1500 workstation using MatLab 6.5.0.180913a Release 13 June 18 2002, and Fortran compiler f77: WorkShop Compilers 4.2 30 Oct 1996 FORTRAN 77 4.2.

### 5.1 Harmonic Wave Equation.

Equation 15 with \( D = 891 = (9.5\pi)^2 \) was solved. This value of \( D \) was chosen to be able to compare results obtained here with those by Lu et al. [3].

![Figure 2](image-url)

**Figure 2:** With \( D6 \), resolution \( j = 5 \) is too low.

As expected, results presented here are consistent with theirs. Figure 2 and Figure 3 show the exact solution versus the computed solution approximated first with an insufficient resolution \( j = 5 \), that is \( 2^j = 32 \) scaling functions.
5 RESULTS

5.1 Harmonic Wave Equation.

Figure 3: Better with higher order wavelets, but $j = 5$ is still too low.

Figure 4: Exact vs. computed solution, $D6$, $j = 8$.

The agreement improves as the resolution is increased: Figure 4 shows that already with $j = 8$, the computed solution is indistinguishable from the exact one with the naked eye.

Figure 5 shows the local error defined by

$$\text{error} = \left( \frac{u_{\text{wav}} - u_{\text{exact}}}{u_{\text{exact}}} \right)^2$$

where $u_{\text{wav}}$ is the approximated solution and $u_{\text{exact}}$ is the exact solution.

Figures 6 and 7 are linear and logarithmic plots respectively, showing how the error decreases as resolution increases using $D6$ wavelets. The logarithmic plot of Figure 7 shows an
anomaly in convergence rate for resolutions \( j > 10 \). The same behavior appears with \( D8 \), but it tends to disappear with \( D10 \) or higher. Figure 8 shows the convergence rate with \( D12 \).

This change in the convergence rate for \( j > 10 \) can be explained by the Galerkin system getting more and more ill-conditioned as resolution increases. This deterioration of the system is less noticeable with higher order wavelets. Indeed, as can be seen in Table 1, the condition number of the system matrix increases with resolution, leading to somewhat less accurate solutions. The condition number for \( D12 \) is consistently lower than that for \( D6 \), accounting
for a less distorted convergence rate for the higher order wavelet. The error, defined as

$$\| \text{error} \|_{L^2} = \left\| \frac{u_{\text{wav}} - u_{\text{exact}}}{u_{\text{exact}}} \right\|_{L^2}$$

where $u_{\text{wav}}$ is the approximated solution and $u_{\text{exact}}$ is the exact solution, is also presented.

But there is more to it. The accuracy of the connection coefficients will also affect the accuracy of the solution, and thus it has to be taken into account. Recall from Section 4.1.1 that the system for computing the coefficients itself is ill-conditioned, particularly for high

Figure 7: Convergence rate with $D_6$.

Figure 8: Convergence rate with $D_{12}$. 
5.2 Biharmonic Wave Equation.

The results obtained for the biharmonic wave equation are not satisfactory so far. One of the tests, though, turned out with an interesting result.

Table 1: Condition number of Galerkin’s system matrix and error norm for the computation of $f(x)$.

<table>
<thead>
<tr>
<th>$j$</th>
<th>Cond. num.</th>
<th>$|error|_{L^2}$</th>
<th>Cond. num.</th>
<th>$|error|_{L^2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3119947.1678</td>
<td>0.014646</td>
<td>2477881.2506</td>
<td>0.011115</td>
</tr>
<tr>
<td>7</td>
<td>2071326.7391</td>
<td>0.0026269</td>
<td>16090320.8111</td>
<td>0.0025938</td>
</tr>
<tr>
<td>8</td>
<td>156019363.9527</td>
<td>0.00062719</td>
<td>117972917.5396</td>
<td>0.00061895</td>
</tr>
<tr>
<td>9</td>
<td>121983360.5394</td>
<td>0.00015673</td>
<td>898783047.9842</td>
<td>0.00014203</td>
</tr>
<tr>
<td>10</td>
<td>9670841055.0175</td>
<td>4.568 $10^{-5}$</td>
<td>7028952330.1376</td>
<td>2.9078 $10^{-5}$</td>
</tr>
<tr>
<td>11</td>
<td>77066191074.7306</td>
<td>2.1318 $10^{-5}$</td>
<td>55676641929.0709</td>
<td>4.0202 $10^{-6}$</td>
</tr>
<tr>
<td>12</td>
<td>615435552677.291</td>
<td>1.6968 $10^{-5}$</td>
<td>443419807197.552</td>
<td>6.3096 $10^{-7}$</td>
</tr>
</tbody>
</table>

Table 2: Condition number for the computation of $\Omega_l^{0,2}$ for increasing resolution.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$D6$</th>
<th>$D12$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>298557.015</td>
<td>252604.5944</td>
</tr>
<tr>
<td>7</td>
<td>1688892.7743</td>
<td>1428921.0655</td>
</tr>
<tr>
<td>8</td>
<td>9553820.1353</td>
<td>8083193.5517</td>
</tr>
<tr>
<td>9</td>
<td>54044568.008</td>
<td>45725446.9705</td>
</tr>
<tr>
<td>10</td>
<td>305722244.1938</td>
<td>258602188.8596</td>
</tr>
<tr>
<td>11</td>
<td>1729426176.2315</td>
<td>1463214302.2079</td>
</tr>
<tr>
<td>12</td>
<td>9783111814.1981</td>
<td>8277190043.3577</td>
</tr>
</tbody>
</table>

Table 3: Condition numbers at $j = 8$ for increasing derivatives of the wavelet bases.

The condition number also depends on the order of the derivatives. Indeed, condition numbers for e.g. $\Omega_l^{0,1}$ are two orders of magnitude lower, and increase at least one order of magnitude with each order of derivation. See Table 3.

5.2 Biharmonic Wave Equation.

The results obtained for the biharmonic wave equation are not satisfactory so far. One of the tests, though, turned out with an interesting result.
Assume \( a(x) \) to be a constant \( a(x) = a \ \forall x \), then the corresponding term for \( u \) in equation 21 can be written using 3-term connection coefficients as

\[
\sum_I u_I \left( \delta_{k,l} + a \sum_m \left( \Omega_{k,l,m}^{0,1,1} + \Omega_{k,l,m}^{0,2,0} \right) \right)
\]

On the other hand, if the analysis is started already with a constant \( a(x) \) in mind, equation 20 can be written using 2-term connection coefficients

\[
\begin{align*}
\frac{\partial^2 u}{\partial x^2} + u + v &= 0 \\
\frac{\partial^2 v}{\partial x^2} - u &= 0
\end{align*}
\]

and the two first terms in the first equation will lead to a term

\[
\sum_I u_I \left( \delta_{k,l} + a \Omega_{k,l}^{0,2} \right)
\]

So, obviously the equality

\[
\Omega_{k,l}^{0,2} = \sum_m \left( \Omega_{k,l,m}^{0,1,1} + \Omega_{k,l,m}^{0,2,0} \right)
\]

must hold.

In this implementation, it only holds for resolution \( j = 0 \). Furthermore, (a) computing a solution for equation 21 substituting just one value of \( a \) for the \( a_m \) in equation 22, effectively turning it into a constant, and (b) computing a solution using equation 22 with \( S = \delta_{k,l} + a \Omega_{k,l}^{0,2} \) both lead to the same solution for \( j = 0 \) but not for higher \( j \). These two facts lead me to believe that the place to look for inaccuracies would be the 3-term connection coefficients.

Recall from Section 2.3.2 that the column sum of a 3-term matrix must equal a corresponding 2-term vector component by component. Using values for the 2 and 3-term connection coefficients obtained by the Fortran code from Restrepo et al. the example

\[
\sum_I \Omega_{l,m}^{1,0,0} = \Omega_{m}^{1,0}
\]

was tested and, again, the equality held only for \( j = 0 \) and not for higher resolutions.

<table>
<thead>
<tr>
<th>( j )</th>
<th>( D6 )</th>
<th>( D12 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>447971.3429</td>
<td>333895.2663</td>
</tr>
<tr>
<td>7</td>
<td>3570823.4907</td>
<td>2547690.9329</td>
</tr>
<tr>
<td>8</td>
<td>28545773.6596</td>
<td>20291584.7671</td>
</tr>
<tr>
<td>9</td>
<td>228310318.3449</td>
<td>162257013.7269</td>
</tr>
<tr>
<td>10</td>
<td>1826295290.7235</td>
<td>1297961173.8320</td>
</tr>
</tbody>
</table>

Table 4: Condition number for the computation of \( \Omega_{l,m}^{0,2,0} \).

This observation is consistent with the values of condition number obtained from the Fortran code used in the connection coefficients computations. See Table 4 for some examples.
6 Conclusions and Future Work.

The Galerkin wavelet method for solving partial differential equations has been tested for two particular cases: a harmonic and a biharmonic wave equation. MatLab functions have been implemented for the solution of the Galerkin linear systems as well as the computation of 2-term connection coefficients (see Appendix B).

The harmonic wave equation results show that the Galerkin wavelet method works, and that the implementation of the computation of 2-term connection coefficients given here is accurate enough even for second order derivatives of the scaling functions involved. Special attention has however to be given to the high condition number of both the matrix coming from the Galerkin discretization and the one involved in the computation of connection coefficients, particularly the 3-term case.

In this context, Romine et al. [5] suggest that it is enough to compute what they call “proper” connection coefficients at scale \( j = 0 \) and then obtain the ones at higher resolutions from those. Indeed, the scaling functions at resolution \( j \) are simply scaled, translated and dilated versions of the mother scale function. Whether this is applicable to the connection coefficients Latto et al. [1] provide should be further investigated. If it turns out this is not the case then alternative methods to deal with the high condition number of the system of equations 13 have to be considered in order to provide accurate 3-term connection coefficients at the desired resolution level.

Yet another approach would be to consider alternative methods to that of Latto et al. to evaluate the wavelet overlap integrals. See for instance [10].

Until these problems are solved, the question as to whether the Galerkin wavelet method is of use when it comes to biharmonic wave equations will remain unanswered.

7 Acknowledgements.

I thank my supervisor, André Jaun, for his guidance and support throughout this thesis work. Thanks go also to Michael Hanke for many valuable discussions.

References


2The URLs given here where valid at the time of writing (May 2004)
A Exact Solution of the Harmonic Wave Equation.

The exact solution of the harmonic equation is given here. Take
\[ \frac{\partial^2 f}{\partial x^2} + D f = 0 \]
with boundary conditions \( f(0) = 0 \) and \( f(1) = 1 \), and which has characteristic polynomial \( \lambda^2 + D = 0 \) so that \( \lambda = \pm i \sqrt{D} \). Such a differential equation has a solution of the form
\[ f(x) = e^{ax} (f_0 \cos(bx) + f_1 \sin(bx)) \]
where \( a = \Re \{ \lambda \} \) and \( b = \Im \{ \lambda \} \). So we have
\[ f = e^{0} \left( f_0 \cos(x\sqrt{D}) + f_1 \sin(x\sqrt{D}) \right) \]
Imposing boundary conditions
\[ f(0) = 0 = f_0 \cos(0) + f_1 \sin(0) = f_0 \Rightarrow f_0 = 0 \]
\[ f(1) = 1 = 0 \cos(\sqrt{D}) + f_1 \sin(\sqrt{D}) = f_1 \sin(\sqrt{D}) \Rightarrow f_1 = \frac{1}{\sin(\sqrt{D})} \]
gives the exact solution
\[ f(x) = \frac{\sin(x\sqrt{D})}{\sin(\sqrt{D})} \]
B Code.

B.1 concoeff_2tuple.m

Computes the 2-term connection coefficients.

```matlab
function lmbd = concoeff_2tuple(pj,N,d1,d2)
eps = 1.0e-30;
format long e;

% Initialization of parameters

df = factorial(d)*2^(-pj*0.5);
Lm100 = Lm + offset + 1;
N100 = N + offset + 1;

% Preparing the Right Hand Side
rhs = zeros(Lm1,1);
rhs(L,1) = df;

% Preparing the solution vector and the Moments vector
lmbd = zeros(Lm1,1);

% Preparing the matrix A

A = zeros(size); % The Daubechies coefficients. I build such a big vector
% to avoid problems with the indexes, which will become
% "negative" at times (think q-2l+p).

% The DH wavelet coefficients according to "Numerical Recipes"
a/h = sqrt(2) => a = sqrt(2) * h

% The DH wavelet coefficients according to "Numerical Recipes"
% a/h = sqrt(2) => a = sqrt(2) * h
a(mi:ni100) = daubechies(N) * sqrt(2);
```

for l = ini:Lm100 % 1:9 (-4:4)
    for q = ini:Lm100 % 1:9 (-4:4)
        for p = ini:N100 % 1:6 (0:5)
            A(l,q) = A(l,q) + a(p) * a(L(q)-2*L(l)+p);
        end
    end
end
```

A = zeros(size); % A doesn't need to be this big, but this way
% I avoid having different indexes for A and a.
```
```
% The DH wavelet coefficients according to "Numerical Recipes"
% a/h = sqrt(2) => a = sqrt(2) * h
a(mi:ni100) = daubechies(N) * sqrt(2);
```

for l = ini:Lm100 % 1:9 (-4:4)
    for q = ini:Lm100 % 1:9 (-4:4)
        for p = ini:N100 % 1:6 (0:5)
            A(l,q) = A(l,q) + a(p) * a(L(q)-2*L(l)+p);
        end
    end
end
end
```
```

```
```
```
```
```
```
```
B. CODE.

A = A-DaHalf;
clear DaHalf;
% Compute the moments M and set them in place.
Mom = moments(N,pj,d,a);
A(Lm100+1,:) = Mom(d+1,:);
% Solve the system A*lmbd=rhs
B = A(ini:Lm100+1,ini:Lm100);
% This system will give \Lambda^{0,d}, but
% \Lambda^{d_1,d_2} = (-1)^d_1 \Lambda^{0,d_2} where d = d_1 +d_2
% So to convert it to the original coefficients,
% simply multiply by (-1)^d_1.

lmbd = (B*rhs)*(-1)^d1;
else % This is the delta function at l-k.
lmbd = zeros(Lm,1);
lmbd(Q) = 1;
end

B.2 moments.m

Computes the moments needed by the 2-term and the 3-term connection coefficients code.

%!MOMENTS Wavelet Moment equations.
%! [M_L] = MOMENTS(DN,J,D,A) Returns a D-row matrix M_L where each row 
%! contains the L moments of the derivative of a wavelet of genus DN, with 
%! coefficients A at resolution J. The order of the derivative increases 
%! with row number.
%! function [M_L] = moments(N,pj,d,a)

% M = 3*N^2-9*N+7;
sizes = 2*(M+d);
offset = sizes/2;
sq2 = sqrt(2);
sq3 = sqrt(2^pj);
Lm = 2*N-3; % Lm = 9;
Lm100 = Lm + offset; 
N100 = N + offset; 
M_0 = zeros(d+1,sizes);
M_L = zeros(d+1,sizes);
ini = 1 + offset;
in2 = 2 + offset;
D100 = d + 1 + offset;
Q = floor((Lm100-ini)/2 + ini);
L = zeros(1,sizes);
L1 = [2:N:N-2]; % L = [-4 ... 0 ... 4];
L(ini:Lm100) = L1;
if d > 0
  for k = ini:Lm100  % 1:2N-3 (2-N:N-2)
    M_0(1,k) = 1; % At least for Daubechies wavelets.
    M_L(1,k) = M_0(1,k) * sq3;
  end
  for k = 2:d+1  % 2:d+1 (1:d)
    m_sum = 0;
    for j = 1:k  % 1:k (0:k-1)
      a_sum = 0;
      for i = ini:N100  % 2-N (1:N-1)
        a_sum = a_sum + a(i)* (i-ini)^(k-j); % (i-1)^((k-1)-(j-1))
      end
      factor = factorial(k-1)/(factorial(j-1)*factorial(k-j));
      m_sum = m_sum + (factor * M_0(j,Q) * a_sum);
    end
    M_0(k,Q) = 1/2*(2^((k-1)-(j-1)) + m_sum);
  end
  for k = 2:d+1  % 2:d+1 (1:d)
    for li = ini:Lm100  % 1:2N-3 (2-N:N-2)
      M_L(k,li) = 0;
      for j = 1:k  % 1:k (0:k-1)
        factor = factorial(k-1)/(factorial(j-1)*factorial(k-j));
        M_L(k,li) = factor * L(li)^((k-1)-(j-1)) * M_0(j,Q);
      end
    end
end
B.3 daubechies.m

Contains Daubechies wavelets coefficients for $D_4$, $D_6$, $D_8$, $D_{10}$, $D_{12}$, $D_{14}$, $D_{16}$, $D_{18}$ and $D_{20}$, computed so that

$$\sum_{k=0}^{N-1} a_k = \sqrt{2}$$

function qmfb = daubechies(n)
% DAUBECHIES Daubechies wavelet coefficients.
% [a] = DAUBECHIES(DN) Returns the Daubechies wavelet coefficients
% of genus DN such that SUM(a) is SQRT(2).
% %
% % Valid genus are 4, 6, 8, 10, 12, 14, 16, 18 and 20.
% %
% % SUBROUTINE quadftr(qmfb, n)
% adapted from the "Numerical Recipes" book by Press et al.
% Daubechies scaling coefficients were computed using
% Paul Abbott's mathematica software (paul@earwax.pd.uwa.edu.au)
% %
% % Adapted for use within MatLab by Jordi Besora, 20040426.
% %
% switch n
% case 4
% qmfb = [4.829629131445341e-1 8.365163037378079e-1 ...
% 2.241438680420134e-1 -1.294095225512604e-1 ...]
% case 6
% qmfb = [3.326705529500826e-1 8.069150931109261e-1 ...
% 4.597755211949161e-1 -1.350110200102546e-1 ...]
% case 8
% qmfb = [2.307381330889654e-1 7.148657055215164e-1 ...
% 6.308807928859815e-2 -2.798376941685985e-2 ...]
% case 10
% qmfb = [1.60103979419226e-1 6.389292697918971e-1 ...
% 7.24308528477729e-1 1.384214690132076e-1 ...]
% case 12
% qmfb = [1.116824366399905e-1 3.96258398948530e-1 ...
% 7.513009802109545e-1 1.52035107019789e-1 ...]
% case 14
% qmfb = [7.852054060501892e-2 3.96539194819173e-1 ...
% 7.29130248265360e-1 4.67922874065193e-1 ...]
% case 16
% qmfb = [5.44107224231040e-2 3.1287159014300e-1 ...
% 6.75630762972898e-1 5.83654863542067e-1 ...]
% case 18
% qmfb = [3.8774976387835e-2 2.4383674612590e-1 ...
% 6.0483612486111e-1 6.5728067051300e-1 ...]
% case 20
% qmfb = [2.6700570055555e-2 1.88178000776915e-1 ...
% end
% M_L(k,:)=M_L(k,:)*2^(-pj*(k-1+0.5));
% end

B.3 daubechies.m
B.4 testcase.m

Solves equation 15.

```matlab
function error = testcase(D,DN,deltar,showit,pic)
if nargin<3
    disp('Wrong number of parameters.');
elseif nargin<4
    showit=0;
pic=0;
else nargin<5
    pic=2;
end

% TESTCASE Solves the one-dimensional differential equation
% f'' + D * f = 0 using the Wavelet Galerkin Method.
% TESTCASE(D,DN,DELTAX) Solves f'' + D*f = 0 Daubechies
% wavelets of order DN, with DELTAX resolution. The wavelet
% resolution j is computed as ceil(log2(1/DELTAX)).
% TESTCASE(D,DN,DELTAX,SHOWIT) Plots the exact solution
% together with the computed solution, and the point-by-point
% error on a separate subplot. All in the current figure.
% TESTCASE(D,DN,DELTAX,SHOWIT,PIC) Plots in figure number PIC.
% omega11 = [];
% localerror= []; Ais = [];

N= (1-0)/deltar; % Ns = number of points. Ns = 2^j, so j = log2(N)
%Wavelet parameter p
p = ceil(log2(N));

D = 2*DN-3; %
DH = 2; % 2-DN <= l <= DN-2
Q = DN-1; % boundary points: DN-1 at left, DN-1 at right
BC = 2Q; % add fictitious boundary points: DN-1 at left, DN-1 at right
BC = N+BC;
A = apex(N,BC,BC)+D;
Ais = zeros(1,DS); % these are the rows

% Compute the connection coefficients
omega02 = concoeff_2tuple(p,DN,0,2);
% Build the system
for i = 1:N
    for j = 1:DS
        DH(j) = i-DH2+j-1;
        A(i,DH) = A(i,DH) + omega02(j); %delta(ij)*D-omega11(j);
    end
end
```

B CODE. B.4 testcase.m

5.272011889317256e-1, 6.884590394536036e-1 ...
2.81723446655775e-1, 1.2769360357933e-1 ...
-1.2946273737370e-1, 1.2769360357933e-1 ...
9.30573663057236e-2, 2.79417563057236e-2 ...
-2.9475526218756e-2, 3.321267405934100e-2 ...
3.606553566956170e-3, 1.073317548333058e-2 ...
1.992405295185056e-3, 1.073317548333058e-2 ...
-6.884590394536036e-4, 1.073317548333058e-2 ...
9.30573663057236e-5, 1.073317548333058e-2 ...
otherwise
disp('Unimplemented value in quad mirror filter.')
disp('Valid values: 4, 6, 8, 10, 12, 14, 16, 18, 20')
end

Solves equation 15.

```matlab
function error = testcase(D,DN,deltar,showit,pic)
if nargin<3
    disp('Wrong number of parameters.');
elseif nargin<4
    showit=0;
pic=0;
else nargin<5
    pic=2;
end

% TESTCASE Solves the one-dimensional differential equation
% f'' + D * f = 0 using the Wavelet Galerkin Method.
% TESTCASE(D,DN,DELTAX) Solves f'' + D*f = 0 Daubechies
% wavelets of order DN, with DELTAX resolution. The wavelet
% resolution j is computed as ceil(log2(1/DELTAX)).
% TESTCASE(D,DN,DELTAX,SHOWIT) Plots the exact solution
% together with the computed solution, and the point-by-point
% error on a separate subplot. All in the current figure.
% TESTCASE(D,DN,DELTAX,SHOWIT,PIC) Plots in figure number PIC.
% omega11 = [];
% localerror= []; Ais = [];

N= (1-0)/deltar; % Ns = number of points. Ns = 2^j, so j = log2(N)
%Wavelet parameter p
p = ceil(log2(N));

D = 2*DN-3; %
DH = 2; % 2-DN <= l <= DN-2
Q = DN-1; % boundary points: DN-1 at left, DN-1 at right
BC = 2Q; % add fictitious boundary points: DN-1 at left, DN-1 at right
BC = N+BC;
A = apex(N,BC,BC)+D;
Ais = zeros(1,DS); % these are the rows

% Compute the connection coefficients
omega02 = concoeff_2tuple(p,DN,0,2);
% Build the system
for i = 1:N
    for j = 1:DS
        DH(j) = i-DH2+j-1;
        A(i,DH) = A(i,DH) + omega02(j); %delta(ij)*D-omega11(j);
    end
end
```

end

B CODE. B.4 testcase.m

5.272011889317256e-1, 6.884590394536036e-1 ...
2.81723446655775e-1, 1.2769360357933e-1 ...
-1.2946273737370e-1, 1.2769360357933e-1 ...
9.30573663057236e-2, 2.79417563057236e-2 ...
-2.9475526218756e-2, 3.321267405934100e-2 ...
3.606553566956170e-3, 1.073317548333058e-2 ...
1.992405295185056e-3, 1.073317548333058e-2 ...
-6.884590394536036e-4, 1.073317548333058e-2 ...
9.30573663057236e-5, 1.073317548333058e-2 ...
otherwise
disp('Unimplemented value in quad mirror filter.')
disp('Valid values: 4, 6, 8, 10, 12, 14, 16, 18, 20')
end

Solves equation 15.

```matlab
function error = testcase(D,DN,deltar,showit,pic)
if nargin<3
    disp('Wrong number of parameters.');
elseif nargin<4
    showit=0;
pic=0;
else nargin<5
    pic=2;
end

% TESTCASE Solves the one-dimensional differential equation
% f'' + D * f = 0 using the Wavelet Galerkin Method.
% TESTCASE(D,DN,DELTAX) Solves f'' + D*f = 0 Daubechies
% wavelets of order DN, with DELTAX resolution. The wavelet
% resolution j is computed as ceil(log2(1/DELTAX)).
% TESTCASE(D,DN,DELTAX,SHOWIT) Plots the exact solution
% together with the computed solution, and the point-by-point
% error on a separate subplot. All in the current figure.
% TESTCASE(D,DN,DELTAX,SHOWIT,PIC) Plots in figure number PIC.
% omega11 = [];
% localerror= []; Ais = [];

N= (1-0)/deltar; % Ns = number of points. Ns = 2^j, so j = log2(N)
%Wavelet parameter p
p = ceil(log2(N));

D = 2*DN-3; %
DH = 2; % 2-DN <= l <= DN-2
Q = DN-1; % boundary points: DN-1 at left, DN-1 at right
BC = 2Q; % add fictitious boundary points: DN-1 at left, DN-1 at right
BC = N+BC;
A = apex(N,BC,BC)+D;
Ais = zeros(1,DS); % these are the rows

% Compute the connection coefficients
omega02 = concoeff_2tuple(p,DN,0,2);
% Build the system
for i = 1:N
    for j = 1:DS
        DH(j) = i-DH2+j-1;
        A(i,DH) = A(i,DH) + omega02(j); %delta(ij)*D-omega11(j);
    end
end
```
% equations Ns+Q+1 to Ns+BC-1
DSi = DSi;
for i = Ns+Q+1:BC-1
    for j = 1:DSi;
        \[A(i,j)] = A(i,j) + omega2(j); \]
        \[\text{Diagonal} \]
        \[A(i,i) = A(i,i) - 1; \]
        \[\text{Main diagonal} \]
    end
end

% Right hand side
rhs = zeros(BBC,1);

% Boundary conditions
A(1,1) = 0; % get rid of D
A(BC,BC) = 0; % get rid of D

% Left boundary
A(1,Q+1) = 1;
rhs(1) = 0; % f(0)=0

% Right Boundary
A(BC,Nx+Q) = 1;
rhs(BC) = 1; % f(1)=1;

%Solve the system
f_w = zeros(BBC,1);

if(showit)
    condition = condest(A);
    msg = ['A has estimated condition number ' num2str(condition)];
    disp(msg);
end

backslash = 1;
if(backslash)
    f_w = A \ rhs;
else
    [L,U] = lu(A);
    for i=1:2
        d = rhs - A*f_w;
        e = -U \ (L \ d);
        f_w = f_w - e;
    end
end

x_trimmed = [0:Ns-1]/Ns;
f_w_trimmed = f_w(Q+1:Nx+Q)';
if(showit)
    % Show the solution 'as is'.
    figure(pics);
    title = ['D' num2str(DN) ', ' num2str(Nx) ' points. '];
    end

% Exact solution
f_exact = zeros(BBC,1);

if(showit)
    % Use 2+f_w_trimmed and 2+f_ex to avoid division by zero.
    localerror = (f_w_trimmed - f_ex)^2 / (2+f_ex)^2;
    if(showit)
        subplot(2,1,1);
        plot(x_trimmed,f_ex,'-',x_trimmed,f_w_trimmed,'r-.');
        legend('Exact solution','WG Solution');
        title(thetitle);
        xlabel('x');
        subplot(2,1,2);
        plot(x_trimmed,localerror);
        title('Error.');
        xlabel('x');
        end
    end

% Convergence
% Use 2+f_w_trimmed and 2+f_ex to avoid division by zero.
error = norm(localerror);
B.5  concoeff_3tuple.m

Computation of 3-term connection coefficients. Inaccurate experimental code.

% CONCOEFF_3TUPLE 3-term wavelet connection coefficient
% \[ \text{CONCOEFF}_3TUPLE \] Computes the 3-term connection
% coefficient matrix with wavelet derivatives 0, D2, and D3, using
% Daubechies wavelets of genus DN at resolution J.
% This code is experimental!!!
% Jordi Besora, NADA, KTH, May 2004.

function lambd = concoeff_3tuple(pj,N,d2,d3)

eps = 1.0e-30;
format long e;

% Initialization of parameters

d = d2 + d3;
M = 3*N^2-9*N+7;
sizes = 2*M+2;
offset = sizes/2;
ini = offset + 1;
Lm = 2*N-3; \% Lm = 9;
Lm1 = Lm + 1; \% Lm1 = 10;
Q = Lm+1:1; \% Q = 5;
N100 = N + offset;

df = factorial(d2)*factorial(d3)*2^(-pj);

L = zeros(1,sizes);
Li = L;
L1=[2:N:N-2]; \% L = \{-4 ... 0 ... 4\};
Li(ini:Lm100) = L1;

Q = Lm1-N+1; \% Q = 5;
Lm100 = Lm + offset;

% Preparing the matrix A

a=zeros(1,sizes); \% The Daubechies coefficients.
% The DN wavelet coefficients according to "Numerical Recipes"
a/h = sqrt(2) => a = sqrt(2) * h
a(ini:N100) = daubechies(N) * sqrt(2);

A = zeros(sizes);
lm = 0;

for l = ini:Lm100 % 1:9 (-4:4)
    for m = ini:Lm100 % 1:9 (-4:4)
        if (abs(l-m)<=(N-2))
            lm = lm + 1; \% (l-ini)*Lm + m-offset;
            qr = 0;
            for q = ini:Lm100 % 1:9 (-4:4)
                for r = ini:Lm100 % 1:9 (-4:4)
                    if (abs(q-r)<=(N-2))
                        qr = qr + 1; \% (q-ini)*Lm + r-offset;
                        for p = ini:N100 % 1:6 (0:5)
                            A(lm,qr) = A(lm,qr) + a(p) * a(L(q)-2*L(l)+p) * a(L(r)-2*L(m)+p);
                        end
                    end
                end
            end
        end
    end
end

% Display

% Pause

if(0)
    for lm = 1:8
        for qr = 1:8
            mssg = ['A(' num2str(lm) ',', num2str(qr) ') = ' num2str(A(lm,qr),18)];
            disp(mssg);
        end
    end
end

if(0)
    for lm = M-8:M
        for qr = M-8:M
            mssg = ['A(' num2str(lm) ',', num2str(qr) ') = ' num2str(A(lm,qr),18)];
            disp(mssg);
        end
    end
end

if(0)
B.5 concoeff_3tuple.m

```matlab
k = 1;
for l = 1:M
    for m = 1:M
        A_v(k) = A(l,m);
        k = k+1;
    end
end

% Compute the moments N and set them in place.

% From here on I just 'copy' from Restrepo's fortran code.

nms = zeros(1,sizes);
sm = zeros(1,lms);
m = zeros(1,sizes);

for m = 1:N-2 % 1...4
    nms(m) = 1 + 2*(N-2) + L1(m);
end
for m = N-1:N
    nms(m) = 1 + 2*(N-2) - L1(m);
end

nms(ini:Lm100) = nms;
clear nms;

for m = ini:Lm100
    nms = nms + nms(m);
end

%disp('Moments for d2*!');
if (mod(d2,2)==0)
    m = 0;
else
    m = 1;
end

for b = 1:d2
    i = M + b;
    for l = ini+L1(i) : ini+L1(i)+L1(m-1)-1
        j = 1 + L(l) + n + (N-2) + nt(ini+N-2+m-1);
        A(i,j) = Mom(b,l);
    end
end

%disp('Moments for d3*!');
if (mod(d3,2)==0)
    l = 0;
else
    l = 1;
end

for c = 1:d3
    i = M + d2 + c;
    for m = ini+Q-2 : ini+Q-2 + L1(i-1)-1
        j = 1 + L(l) + n + (N-2) + nt(ini-N+2-m-1);
        A(i,j) = Mom(c,m);
    end
end
```

38
B.5 conoeff_3tuple.m

```matlab
i = M + d + 1;
for m = ini:ini+Q-2 % 1:4 (-4:-1)
    for l = ini:L(m-1) % 1:9-m (-4:4-m)
        j = 1 + L(l) + (N-2) + nt(m-1);
        A(i,j) = Mom(d2+1,l) * Mom(d3+1,m);
        % mssg = ['A(' num2str(i) ',' num2str(j) ') = ' num2str(A(i,j),18)];
        % disp(mssg);
    end
end

form = ini+Q-1:Lm100 % 5:9 (0:4)
for l = ini+L(m):Lm100 % 1+m:9 (-4+m:4)
    j = 1 + L(l) - L(m) + (N-2) + nt(m-1);
    A(i,j) = Mom(d2+1,l) * Mom(d3+1,m);
    % mssg = ['A(' num2str(i) ',' num2str(j) ') = ' num2str(A(i,j),18)];
    % disp(mssg);
end

% Preparing the Right Hand Side
rhs = zeros(M+d2+d3+1,1);
rhs(M+d2+d3+1,1) = df;

% Preparing the solution vector
lmbd_a = zeros(M,1);
lmbd = zeros(Lm,Lm); % 9x9

% Solve the system A*lmbd=rhs
B = A(1:M+d2+d3+1,1:M);

% Testing my A against Restrepo's ac3
if (0)
    mytol = 1e-13;
    disp('--------------------------------------------');
    mssg = ['Testing my A against Restrepo''s ac3 Tol = ' num2str(mytol)];
    disp(mssg);
    mssg = ['A has condition number ' num2str(cond(B),15)];
    disp(mssg);
    mssg = ['The error is computed as (mine - his)/his. '];
    disp(mssg);
    k = 1;
    for i = 1:M+d+1
        for j = 1:M
            myac3(k) = B(i,j); % getting my A in one vector.
            k = k + 1;
        end
    end
    hisac3 = ac3(3); % his A, his ac3 that is.
    isitright = [];
    % size(myac3)
    % size(hisac3)
    hismin = 1 + abs(min(hisac3));
    for i = 1:M
        minusv = myac3(i) - hisac3(i);
        plusv = myac3(i) + hisac3(i);
        if((minusv<0)&(plusv<0))
            isitright(i) = 0;
        else
            isitright(i) = 2*minusv/plusv; % errors?? what errors?? :-)
        end
    end
    for i = 1:size(isitright,1)
        if(isitright(i)<mytol) % A little threshold.
            isitright(i) = 0;
        end
    end
    mx = max(abs(isitright));
    mn = min(abs(isitright));
    if (mx<0 | mn < 0)
        mssg = ['max error = ' num2str(mx) ... ' , min error = ' num2str(mn)];
        disp(mssg);
        nonzero = find(isitright);
        numzeros = size(nonzero);
        mssg = ['There are ' num2str(numzeros(1)) ' errors. '];
        disp(mssg);
    else
        disp('No differences within tolerance. ');
    end
end

% End of A vs ac3 test.
if(backslash)
    lmbd_s = lmbd(B,rhs);
else
    [L,U] = lu(B);
    lmbd_s = U\(L\cdot rhs);
for i=1:2
    d = rhs - B\lmbd_s;
    e = -U\(L\cdot d);
    lmbd_s = lmbd_s - e;
end
end

% Put the result in matrix form.

lmbd_w = zeros(Lm,1);
j = 1;
for l = 1:Q+i % The first half
    for m = 1:Q+i
        lmbd(l,m) = lmbd_s(j);
        j = j+1;
    end
    i = i+1;
end
i= Q-2;
for l = Q+1:Lm % The second half
    for m = Q-i:Lm
        lmbd(l,m) = lmbd_s(j);
        j = j+1;
    end
    i= i-1;
end

% Testing my result against Restrepo's
if (0)
    mytol = 1e-15;
    disp('--------------------------------------------');
    mssg= ['Testing my result against Restrepo''s. Tol = ' num2str(mytol)];
    disp(mssg);
    mssg= ['The error is computed as 2*(mine - his)/(mine + his). '];
    disp(mssg);
    lmbd_v = lmbd(B,rhs);
    j = 1;
    for l = 1:Lm
        for m = 1:Lm
            lmbd_v (j,1) = lmbd(l,m); % my result in vector form.
            j = j+1;
        end
    end
    size(lmbd_v);
    hislambda = his3tuples(3); % his lambda.
    size(hislambda);
    % hismin = 1 + abs(min(hislambda));
    for i = 1:Lm
        minusv = lmbd_v(i) - hislambda(i);
        plusv = lmbd_v(i) + hislambda(i);
        if((minusv==0)&(plusv==0))
            isitright(i) = 0;
        else
            isitright(i) = 2*minusv/plusv; % errors?? what errors?? :-)
        end
    end
    % isitright = 2*(lmbd_v-hislambda)/(lmbd_v+hislambda);
    for i = 1:size(isitright,1)
        if(isitright(i)<mytol) % A little threshold.
            isitright(i) = 0;
        end
    end
    mx = max(abs(isitright));
    nn = min(abs(isitright));
    if(mx>0)
        m = 2
        msg = [maximum error = ' num2str(mx) ...
              ' minimum error = ' num2str(nn)];
        disp(msg);
        nonzero = find(isitright);
        nonzero
        msg = [There are ' num2str(num_nonzero) ' errors. '];
        disp(msg);
        nonzero
        else
            disp('No differences within tolerance.');
        end
    disp('--------------------------------------------');
end

% End of lmbd vs lambda test.
B.6 multiscale.m

Solution of equation 20. Not sufficiently debugged.

\%
clear;
format long e;
eps = 1.0e-30;
showofs = 1; \%=1 if a(x) is to be plotted as well.
\%
Input parameters
DN = 6; \% Order of the wavelet.
deltar = 2^-9; \% scale resolution. (2^{-j})
\%
Input parameters.
\%
Connection coefficient vectors
omega11 = []; omega011 = []; omega020 = [];
xa = -5; xb = 5; bma = xb-xa;
Nx = (bma)/deltar; \% Nx = number of points. \% =2^j, so j = log2(Nx)
p = ceil(log2(Nx/bma));
\%
DS = 2*DN-3; \% DS = 2^-3; \% 2^-3 <= l <= 2^-3.
Q = DN-1; \% Boundary points: DN-1 at left, DN-1 at right
HC = 2^Q; \% add fictitious boundary points: DN-1 at left, DN-1 at right
NBC = Nx+HC;
\%
\% =========== Solution vector ===========
uv = zeros(2*NBC,1);
\%
\% =========== Coefficient function a(x) ===========
ep = 0.01;
x = [-Q:Nx/2-1]*bma/Nx;
aofx = a(x,ep);
x_t = [-Q:Nx/2-1]*bma/Nx;
\%
\% =========== Connection coefficients ===========
omega11 = concoeff_2tuple(p,DN,1,1);
omega02 = (concoeff_2tuple(p,DN,0,2));
omega011 = D6p9011; \% D6p9011;
omega020 = D6p9020; \% D6p9020;
omega020 = concoeff_3tuple(p,DN,2,0);
thesomega = (omega011 + omega020); %*2^(ceil(p/2));
\%
\% =========== Building the building blocks ===========
\%
\% Delta blocks ===========
DB = speye(NBC,NBC);
DB(1,1) = 0;
DB(NBC,NBC) = 0;
\%
\% Omega block ===========
GB = speye(NBC,NBC); \% It will be overwritten
GB(1,1) = 0;
GB(NBC,NBC) = 0;
\%
\% Main body of the omega block
for k = Q+1:Q+Nx \% rows
  for j = 1:DS \% columns
    GB(k,k-DN2+j-1) = omega02(j); \% omega11(j);
  end
end
\%
\% equations 2 to DN-2 of the omega block
for k = 2:DN-2 \% rows
  for j = 1:DN \% columns
    GB(k,k-DN2+j-1) = omega02(j); \% omega11(j);
  end
end
\%
\% equations Q+1 to HC+1 of the omega block
DSk = (DN+1) \% rows
for k = Q+1:Q+Nx \% rows
  for j = 1:DN \% columns
    GB(k,k-DN+j-1) = omega02(j); \% omega11(j);
  end
end
end
DSk = DSk - 1;
end

% First block
FB = speye(NBC,NBC);
FB(1,1) = 0;
FB(NBC,NBC) = 0;

% Main body of the first block
for k = Q+1:Q+Nx % rows
  for j = 1:DS % columns
    DNj = k-DN2+j-1;
    sum = 0;
    mm = 1;
    for m = k-DN2:k+DN2
      sum = sum + aofx(m) * theomega(j,mm);
      mm = mm + 1;
    end
    FB(k,DNj) = FB(k,DNj) + sum;
  end
end

% equations 2 to DN-2 of the first block
for k = 2:Q
  DNk = DN-k;
  for j = DNk:DS
    sum = 0;
    mm = Q+1-k;
    for m = 1:DN2+k
      sum = sum + aofx(m) * theomega(j,mm);
      mm = mm + 1;
    end
    DNj= j-DNk+1;
    FB(k,DNj) = FB(k,DNj) + sum;
  end
end

% equations Nx+Q+1 to Nx+BC-1 of the first block
DSk = DS;
for k = Nx+Q+1:NBC-1
  for j = 1:DSk;
    DNj = k-DN2+j-1;
    sum = 0;
    mm = 1;
    DNm = k-DN2;
    for m = DNm:NBC
      sum = sum + aofx(m) * theomega(j,mm);
      mm = mm + 1;
    end
    FB(k,DNj)= FB(k,DNj) + sum;
  end
  DSk= DSk - 1;
end

% Building the matrix
A = [ FB DB
       -DB OB ];
% Test with a constant a(x) = a
%A = [DB+aofx(Nx/2)*OB DB
%      -DB OB];
% Right hand side
rhs = zeros(2*NBC,1);
% Boundary conditions
A(1,Q+1) = 1; % u(a) = 0
rhs(1) = 0;
A(NBC+1,NBC+Q+1) = 1; % v(a) = 0
rhs(NBC+1) = 0;
A(NBC,Nx+Q) = 1; % u(b) = 1
rhs(NBC) = 1;
A(end,NBC+Nx+Q) = 1; % v(b) = 0
rhs(2*NBC) = 0;
% Solving the system
if(0)
  mssg = ['multiscale: D' num2str(DN) ' p = ' num2str(p)];
disp(mssg);
  condition = condest(A);
  mssg = ['multiscale: A has estimated condition number ' num2str(condition) '.'];
  disp(mssg);
end
B.7  a.m

Returns $a(x)$ in equation 20. Needed by multiscale.m.

function a_out = a(x,epsilon)
    a_out = [];
    maxindex = max(size(x,1),size(x,2));
    for k = 1:maxindex
        a_out(k) = epsilon*[(x(k)/sqrt(1+x(k)^2)) + (1/(x(k)+1i*epsilon))];
    end

B.8  testcoeff.m

Performs the test described in equation 25

% TESTCOEFF

clear
DS = 6;
p = 0;
DS = 2*DS-3;

% ------------------- Connection coefficients -------------------
omegatw1 = concoeff_2tuple(p,DS,5,1,1);
omegatw2 = concoeff_2tuple(p,DS,0,0,2);
B.9 coechecksum.m

Performs the check-sum procedure example of Section 2.3.2

```matlab
% COEFFCHECKSUM

clear
DN = 12;
p = 0;
DS = 2*DN-3;

% =================== Connection coefficients ======================

omega011 = concoeff_2tuple(p,DN,1,1);
omega100 = concoeff_2tuple(p,DN,1,0);
omega001 = 12*p001; % D6p10011;
omega010 = concoeff_2tuple(p,DN,1,0);
omega020 = concoeff_2tuple(p,DN,2,0);
theomega = -(omega001 + omega010);

newomega = zeros(max(size(omega10)),1);
for m = 1:DS
    summ = 0;
    for l=1:DS
        summ = summ + theomega(l,m);
    end
    newomega(m) = summ;
end
omega10-newomega
```

B.10 D6p9020.m

Example code to pass connection coefficients computed with existing Fortran code to the MatLab functions. Note how the name describes the order of the wavelets $D$, the resolution $j$ and the order of the three derivatives involved. This example is $\Omega^{0,2,0}_{4,9}$ with $D6$ at resolution $j = 9$. The values inside a_in are just cut and pasted from file 3-tuples generated by the Fortran code.

```matlab
function a_out = D6p9020
DN = 6;
Q = DN-1;
Lm = 2*DN-3;
a_in = [-4 -4 -0.1839850377535e+03 -4 -3 0.1347314526641e+04
-4 -2 -0.6826143664399e+04 -4 -1 0.3127865336009e+05
-4 0 0.6160812255632e+04 -4 1 0.0000000000000e+00
-4 2 0.0000000000000e+00 -4 3 0.0000000000000e+00
-4 4 0.0000000000000e+00];
```
% Put the result in matrix form.

j = 1; % index for a_in

for l = 1:Lm
    for m = 1:Lm
        a_out(l,m) = a_in(j,3);
        j = j+1;
    end
end