1998

Implementation of the Wavelet-Galerkin method for boundary value problems

Adam Scheider

Implementation of the Wavelet-Galerkin Method for Boundary Value Problems

by

Adam K. Scheider

A Thesis Submitted In
Partial Fulfillment of the
Requirement for the
Master of Science
In
Mechanical Engineering

Approved by:

Professor_____
Josef S. Török
Thesis Advisor

Professor_____
H. Ghoneim

Professor_____
K. Kochersberger

Professor_____
Charles Haines
Department Head

DEPARTMENT OF MECHANICAL ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY

MAY 1998

Implementation of the Wavelet-Galerkin Method for Boundary Value Problems

I, Adam K. Scheider, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial profit.

Date: May 20, 1998 Signature of Author: _____

DEDICATION

I dedicate this thesis first and foremost in loving memory of Scott Crouch and in living recognition of Thomas Haralson. These two men have touched my life in such a special way that I know I will never meet anyone else like them. I believe you can only meet very few people so very special. Without them I would not have grown to be the man that I am today. And I would not have completed my college career at RIT in the manner in which I have if I had not worked so hard to make them so proud.

My family took no small part in this act. If I could give any piece of advice to any person, it would be to not take your family for granted.

The prayers of family and friends have not gone unnoticed, especially those of Sr. Grace, Sr. Anne, and Fr. Gentile. I hold the Domes and the Denicks in a distinguished place as well.

Finally, I would like to acknowledge Dr. Török and the rest of the Mechanical Engineering faculty. Without them, this would not have been possible.

TABLE OF CONTENTS

Abstract

The objective of this work is to develop a systematic method of implementing the Wavelet-Galerkin method for approximating solutions of differential equations. The beginning of this project included understanding what a wavelet is, and then becoming familiar with some of the applications. The Wavelet-Galerkin method, as applied in this paper, does not use a wavelet at all. In actuality, it uses the wavelet's scaling function. The distinction between the two will be given in the following sections of this paper.

The sections of this thesis will include defining wavelets and their scaling functions. This will give the reader valued insight to wavelets and Discrete Wavelet Transforms (DWT). Following this will be a section defining the Galerkin method. The purpose of this section will be to give the reader an understanding of how weighted residual methods work, in particular, the Galerkin Method. Next will be a section on how Scaling functions will be implemented in the Galerkin method, forming the Wavelet-Galerkin Method.

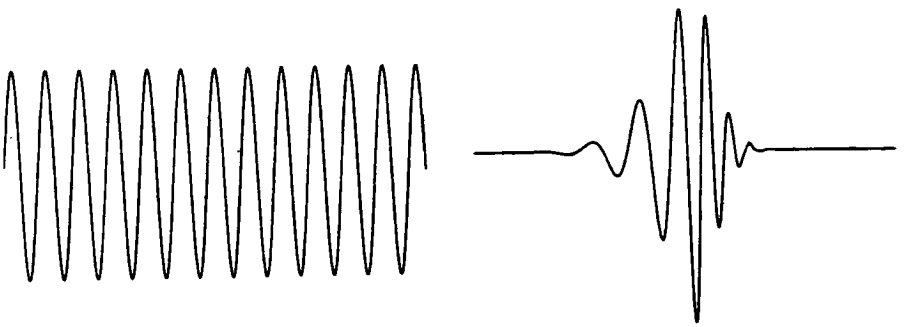
The focus of this investigation will deal with solutions to a basic homogeneous differential equation. The solution of this basic equation will be analyzed using three separate, distinct methods, and then the results will be compared. These methods include the Wavelet-Galerkin Method, the Galerkin Method using quadratic shape functions, and standard analytical means. Factors to be studied include computational time, effort, accuracy, and ease of implementing the method of solution.

After a thorough comparison has been made, there will be a section to talk about possible applications of the Wavelet-Galerkin method and recommendations for future work. Predictions of what avenues to pursue in refining the Wavelet-Galerkin method will also be stated. And suggestions on how to make the method more accurate will be given.

Introduction to Wavelets

A wave is usually defined as disturbance in time or space. Periodic waves, such as a sinusoid, repeat after a finite interval. Fourier analysis is wave analysis. It expands signals or functions in terms of sinusoids (or, equivalently, complex exponentials) which has proven to be extremely valuable in mathematics, science, and engineering, especially for periodic, time-invariant, or stationary phenomena.

A wavelet is a "small wave", which has its energy concentrated in time to give a tool for analysis of transient, nonstationary, or time-varying phenomena. It still has the oscillating wavelike characteristic but also has the ability to allow simultaneous time and frequency analysis with a flexible mathematical foundation. This is illustrated below with the wave (sinusoid) oscillating with equal amplitude over $-\infty<t<\infty$ and, therefore, having infinite energy and with the wavelet having its finite energy concentrated around a point or instant of time.



(a) A Sine Wave

(b) Daubechies' Wavelet ψ_{D20}

We will take the wavelets and use them in a series expansion of signals or functions much the same way a Fourier series uses the wave or sinusoid to

represent a signal or function. The signals are functions of a continuous variable, which often represents time or distance.

A signal or function can often be better analyzed or described or processed by decomposing it into a series or summation. An example of this would be digital signal processing (dsp). Say whatever object that is supposed to pick up the signal, a microphone, accelerometer, guitar pickup, etc... is too sensitive and the signal has a significant amount of static or interference. By expressing the signal a summation of terms, it would be possible to throw out the terms that are causing the unwanted noise, leaving only the clear portion of the signal.

Therefore, the first step is to express the signal as a series expressed by linear decomposition by

$$f(t) = \sum_k a_k \Psi_k(t) \qquad (1.1)$$

where k is an integer index for the finite or infinite sum. The coefficients a_k are real-valued expansion coefficients and $\Psi_k(t)$ are a set of real-valued functions of t called the expansion set. If the expansion set is unique, it is called a *basis*. A basis is orthogonal if its inner product of distinct basis functions is zero, that is

$$\langle \Psi_k(t), \Psi_l(t) \rangle = \int \Psi_k(t) \Psi_l(t) dt = 0 \qquad k \neq l$$

If the basis is orthogonal, then the coefficients in (1.1) may be calculated by multiplying both sides by the generic function $\Psi_k(t)$ and taking the inner product:

$$a_k = \langle f(t), \Psi_k(t) \rangle = \int f(t) \Psi_k(k) dt \qquad (1.2)$$

For a Fourier series, the orthogonal basis functions are $\sin(k\omega_o t)$ and $\cos(k\omega_o t)$ with frequencies of $k\omega_o$. For a Taylor's series, the nonorthogonal basis functions are simple monomials t^k, and for many other expansions they are various polynomials. There are expansions that use splines and even fractals.

For the wavelet expansion, a two parameter system is constructed such that the linear decomposition is of the form

$$f(t) = \sum_k \sum_j a_{j,k} \Psi_{j,k}(t) \qquad (1.3)$$

where both *j* and *k* are integer indices and the $\Psi_{j,k}(t)$ are the wavelet expansion functions that usually form an orthogonal basis. The set of expansion coefficients $a_{j,k}$ are called the *discrete wavelet transform* (DWT) of f(t) and the linear decomposition expression is the inverse transform.

Wavelet Systems

Wavelet expansions are not unique. That is, there may be more than one wavelet system that may successfully represent a signal or function. However, all of them seem to have three basic characteristics:

1. A wavelet system is a set of building blocks to construct or represent a signal or function. It is a two-dimensional expansion set (usually a basis, which means it is unique) for some class of one- (or higher) dimensional signals. In other words, if the wavelet set is given by $\Psi_{j,k}(t)$ for indices of j,k = 1,2,..., a linear expansion would be

$$f(t) = \sum_k \sum_j a_{j,k} \Psi_{j,k}(t)$$

for some set of coefficients $a_{j,k}$.

2. The wavelet expansion gives a time-frequency localization of the signal. This means most of the energy of the signal is well-represented by the discrete expansion coefficients, $a_{j,k}$.
3. The calculation of coefficients from the signal may be done efficiently. Many wavelet transforms may be calculated with O(N) number of operations. This means the number of floating-point multiplications and additions increase linearly with the length of the signal. More general wavelet transforms require O(Nlog(N)) operations, essentially the same as the Fast Fourier Transform (FFT).

A Fourier series maps a one-dimensional function of a continuous variable into a one-dimensional sequence of coefficients. Whereas the wavelet expansion maps it into a two-dimensional representation that allows localizing the signal in both time and frequency. A wavelet representation will give location in both time and frequency simultaneously. By this explanation, a wavelet representation resembles a musical score in a way, where the location of the notes tells when the tones occur and what their frequencies are.

There are three more additional characteristics, which are more specific to wavelet expansions. These are not so much characteristics but properties that make wavelets useful. They are:

1. All first-generation wavelet systems are generated from a single scaling function or wavelet by simple *scaling* and *translation.* The two-dimensional parameterization is attained from the function

$$\Psi_{j,k}(t)=2^{j/2}\Psi(2^{j}t-k)$$

where j and k are elements of **Z**, the set of all integers. This equation is sometimes referred to as the "mother wavelet". The factor $2^{j/2}$ maintains a constant norm independent scale of j. This parameterization of the time or space location by k and the frequency or scale by j (really the log of the scale) is extremely effective.

2. In addition, almost all useful wavelet systems satisfy multiresolution conditions. This means that if a set of signals can be represented by a weighted sum of $\phi(t-k)$, then a larger set (which includes the original) can be represented by a weighted sum of $\phi(2t-k)$. Basically, if the basic expansion signals are made half as wide and translated in steps half as wide, they will represent a larger class of signals exactly or give a better approximation of any signal. This last statement is in essence what makes wavelets work!
3. Lower resolution coefficients may be calculated from higher resolution coefficients (via a filter bank). This allows a very efficient calculation of the expansion coefficients and relates wavelet transforms to a known area in digital signal processing.

Multiresolution formulation requires two closely related functions, which are the wavelet $\Psi(t)$ and the scaling function $\phi(t)$. Together, these functions allow a large class of functions to be expressed by the form

$$f(t) = \sum_{k=-\infty}^{\infty} c_k \phi(t-k) + \sum_{k=-\infty}^{\infty} \sum_{j=0}^{\infty} d_{j,k} \Psi(2^j t - k) \qquad (1.4)$$

Another form of this equation is given below. It is created by substituting the mother wavelet expression into the above equation.

$$g(t) = \sum_{k} c_{j_0}(k) 2^{j_0/2} \phi(2^{j_0} t - k) + \sum_{k} \sum_{j=j_0}^{\infty} d_j(k) 2^{j/2} \Psi(2^j t - k) \qquad (1.5)$$

A very interesting fact should be noted here, before we go any farther. Each j term of the summation represents its own subspace of the approximation. The terms $\Psi_{j,k}(t)$ actually span the *differences* between the approximations spanned by the various scales of the scaling function.

To describe this better, a more general view should be taken. The goal of the discrete wavelet transform (DWT), eqn (1.5), is to approximate a function or signal over an interval in $L^2(\mathbf{R})$. This is the space of functions f(t) with a well defined integral of the square of the modulus of the function. $L^2(\mathbf{R})$ is also known as the space of finite-energy signals. The "L" signifies a Lebesque integral, the "2" denotes the integral of the square of the modulus of the function, and **R** states that the independent variable of integration *t* is a number over the whole real line. For a function g(t) to be a member of that space, it is denoted by: $g \in L^2(\mathbf{R})$ or simply $g \in L^2$. Although most of the definitions and derivations are in terms of signals that are in L^2, many of the results hold for larger classes of signals. For

example, polynomials are not in L^2, but can be expanded over any finite domain by most wavelet systems.

Anyway, if the j's reached to infinity, the approximation would span all of L^2 Since all of the j's represent the difference between the subspaces, the symbolic representation would look like this:

$$L^2 = V_0 \oplus W_0 \oplus W_1 \oplus W_2 \oplus \ldots$$

It is important to note that the scale of the initial space is arbitrary and could be chosen at a higher, or lower, resolution. That is to say that j_0 could take on any positive or negative integer value we want to assign it. This also includes negative infinity.

Let's take a closer look at equation (1.5), the DWT formula. The first summation, the single summation, represents only one subspace. It only represents one term. It is the initial subspace. The value of c(k) is found, almost instinctually, by finding the dot product of the basic scaling function $\phi(t)$ and the function we wish to approximate g(t).

$$c(k) = c_0(k) = \langle g(t), \phi_k(t) \rangle = \int g(t)\phi_k(t)dt \qquad (1.6)$$

The $d_j(k)$ coefficients are found in a similar way by dotting the function g(t) with the wavelet $\Psi_{j,k}(t)$, not the scaling function $\phi(t)$.

$$d_j(k) = d(j,k) = \langle g(t), \Psi_{j,k}(t) \rangle = \int g(t)\Psi_{j,k}(t)dt \qquad (1.7)$$

The coefficient d(j,k) is sometimes written as $d_j(k)$ to emphasize the difference between the time translation index k and the scale parameter j. The coefficient

c(k) is also sometimes written as $c_j(k)$ or c(j,k) if a more general "starting scale" other than j=0 is used for the lower limit. These formulas may be used so long as the wavelet system being used is orthogonal. This is no minor point to overlook. Although, most wavelet systems are orthogonal in nature. This is one of the mathematical requirements that the wavelets are subjected to when being constructed.

We shall now define the scaling function $\phi(t)$ and the wavelet $\Psi(t)$. Both of these functions are based upon recursion. I will simply state the definitions.

$$\phi(t) = \sum_n h(n)\sqrt{2}\phi(2t-n) \qquad n \in \mathbf{Z} \qquad (1.8)$$

$$\Psi(t) = \sum_n h_1(n)\sqrt{2}\phi(2t-n) \qquad n \in \mathbf{Z} \qquad (1.9)$$

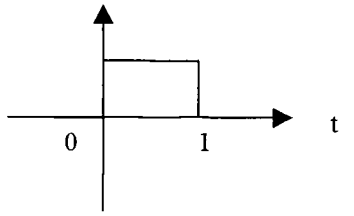
In these equations, **Z** is the set of all integers. The coefficients h(n) are a sequence of real or perhaps complex numbers called scaling functions and the $\sqrt{2}$ maintains the norm of the scaling function with the scale of two. The coefficients $h_1(n)$ are related to the scaling function coefficients. Due to the requirement that the wavelets span the "difference" between the subspaces and the orthogonality prerequisite, $h_1(n)$ is related to h(n) by

$$h_1(n) = (-1)^n h(1-n).$$

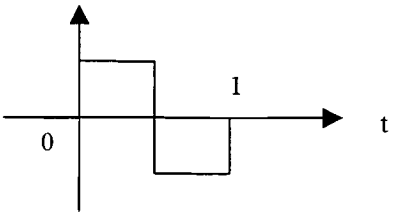
I will demonstrate how to perform a discrete wavelet transform on a simple polynomial in order to illustrate its usage. The example is in the next chapter.

Example – Haar Approximation

This example demonstrates how to use a discrete wavelet transform (DWT) to approximate a given function. The function, chosen at random, is defined as $g(t) = 5t^2 - 3t + 1$. The wavelet to be used is the simplest wavelet known. It is called the Haar wavelet. The Haar scaling function is what is commonly known as the unit step function. However, if the formula that is given for the scaling function is referred to, the scaling function has coefficients $h(0) = 1/\sqrt{2}$ and $h(1) = 1/\sqrt{2}$.



Haar Scaling Function $\phi(t)$



Haar Wavelet $\Psi(t)$

The initial scale, j_0, was chosen to be zero. As stated in the earlier section, it could be chosen at random. Zero was picked because it seemed logical. Also, the interval on which we want the approximation was picked to be from zero to one because it seemed logical.

The first step was to integrate the function times the scaling function over the internal. The scaling function is simply the unit step function from zero to one.

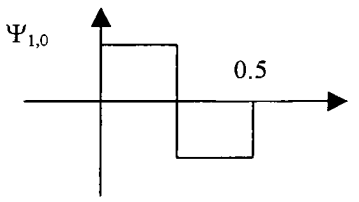
$$c(k) = \int_0^1 (5t^2 - 3t + 1)(1)dt = 1.166667$$

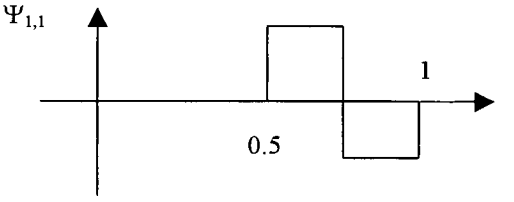
This becomes our V_0 subspace. The plot is shown on the following page. The next subspace is approximated by finding $d_0(0)$. This means finding the value of the integral of the wavelet times the test function over the interval.

$$d_0(0) = \int_0^{0.5} (5t^2 - 3t + 1)(1)dt + \int_{0.5}^{1} (5t^2 - 3t + 1)(-1)dt = -0.500000$$

This spans the subspace W_0 which is the difference between the adjacent subspaces. If the spreadsheet is referred to, look at the W_0 column. It shows a negative 0.5 value halfway down, and then a positive 0.5 the rest of the way. This is due to the DWT formula. The coefficient must be multiplied by its wavelet. In this case, the wavelet simply flops from positive one, to negative one.

The next step is finding $d_1(0)$ and $d_1(1)$. This is what the associated wavelets look like.





These graphs show how the scaling and translation principles work. The wavelets still have the same height, only they are in different places and scaled. Both of these wavelets have magnitudes of 1 everywhere. The coefficients are calculated using the following formulas.

$$d_1(0) = \int_0^{0.25} (5t^2 - 3t + 1)(1)dt + \int_{0.25}^{0.5} (5t^2 - 3t + 1)(-1)dt = 0.0312500$$

$$d_1(1) = \int_{0.5}^{0.75} (5t^2 - 3t + 1)(1)dt + \int_{0.75}^{1} (5t^2 - 3t + 1)(-1)dt = -0.2812500$$

These coefficients may then be used to calculate the values in column W1. They must be multiplied by $\sqrt{2}$ and by the wavelet, of course. This can be found in the formula. The approximation to g(t) may be found by summing all the previous subspaces, including V_0.

At any rate, it should be clear now how each successive subspace is approximated. The graphs show a few more approximations. It can be seen that the last approximation is not a very good one. However, it does serve the purpose to demonstrate how the DWT works. Also, this example clearly shows how coarse the Haar wavelet really is. The challenge is to find smoother wavelets which are still orthogonal. Ingrid Daubechies has done a lot of work in this area. She really is considered to be one of the leaders in wavelet technology.

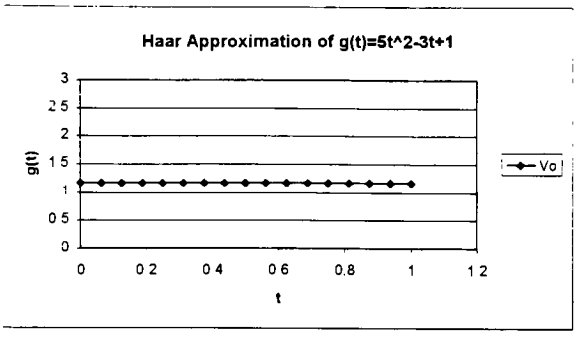


Figure 1

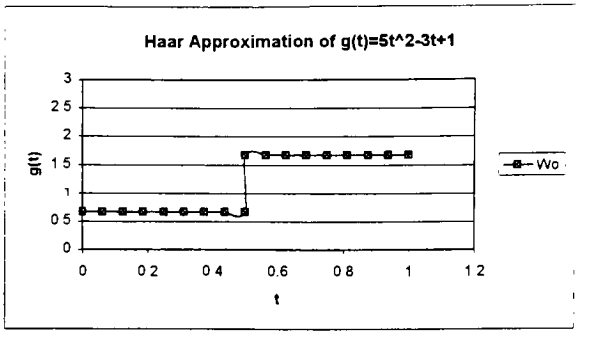


Figure 2

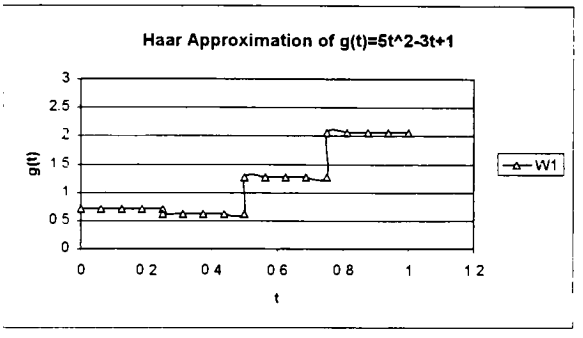


Figure 3

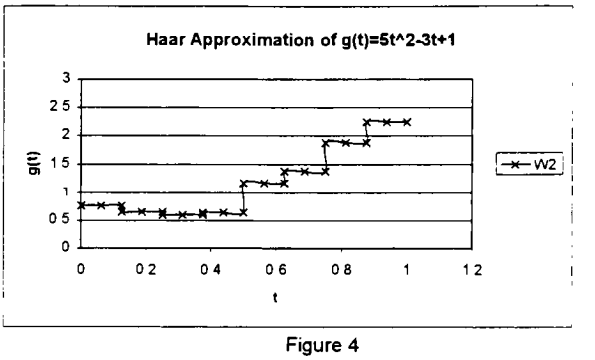


Figure 4

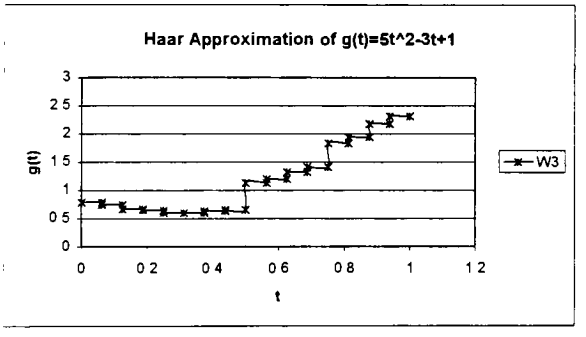


Figure 5

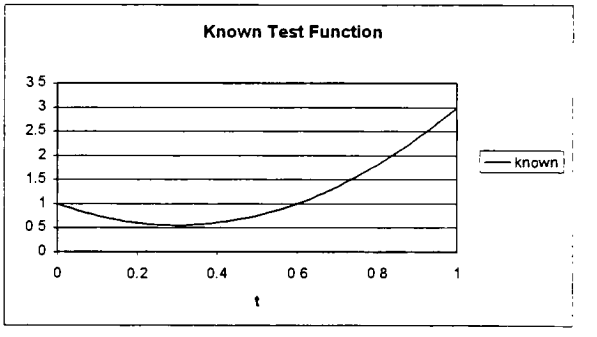


Figure 6

Table 2.1, Subspaces of Haar Approximation										
t	Vo	Wo	g	W1	g	W2	g	W3	g	known
0	1.16666667	-0.5	0.666667	0.044194	0.710861	0.0546875	0.765548	0.026240291	0.791788635	1
0.0625	1.16666667	-0.5	0.666667	0.044194	0.710861	0.0546875	0.765548	0.026240291	0.791788635	0.832031
0.0625	1.16666667	-0.5	0.666667	0.044194	0.710861	0.0546875	0.765548	-0.02624029	0.739308053	0.832031
0.125	1.16666667	-0.5	0.666667	0.044194	0.710861	0.0546875	0.765548	-0.02624029	0.739308053	0.703125
0.125	1.16666667	-0.5	0.666667	0.044194	0.710861	-0.054688	0.656173	0.012429611	0.668602955	0.703125
0.1875	1.16666667	-0.5	0.666667	0.044194	0.710861	-0.054688	0.656173	0.012429611	0.668602955	0.613281
0.1875	1.16666667	-0.5	0.666667	0.044194	0.710861	-0.054688	0.656173	-0.01242961	0.643743732	0.613281
0.25	1.16666667	-0.5	0.666667	0.044194	0.710861	-0.054688	0.656173	-0.01242961	0.643743732	0.5625
0.25	1.16666667	-0.5	0.666667	-0.044194	0.622472	-0.023438	0.599035	-0.00138107	0.597653928	0.5625
0.3125	1.16666667	-0.5	0.666667	-0.044194	0.622472	-0.023438	0.599035	-0.00138107	0.597653928	0.550781
0.3125	1.16666667	-0.5	0.666667	-0.044194	0.622472	-0.023438	0.599035	0.001381068	0.600416064	0.550781
0.375	1.16666667	-0.5	0.666667	-0.044194	0.622472	-0.023438	0.599035	0.001381068	0.600416064	0.578125
0.375	1.16666667	-0.5	0.666667	-0.044194	0.622472	0.0234375	0.64591	-0.01519175	0.630718249	0.578125
0.4375	1.16666667	-0.5	0.666667	-0.044194	0.622472	0.0234375	0.64591	-0.01519175	0.630718249	0.644531
0.4375	1.16666667	-0.5	0.666667	-0.044194	0.622472	0.0234375	0.64591	0.015191747	0.661101743	0.644531
0.5	1.16666667	-0.5	0.666667	-0.044194	0.622472	0.0234375	0.64591	0.015191747	0.661101743	0.75
0.5	1.16666667	0.5	1.666667	-0.397748	1.268919	-0.101563	1.167357	-0.02900243	1.138354179	0.75
0.5625	1.16666667	0.5	1.666667	-0.397748	1.268919	-0.101563	1.167357	-0.02900243	1.138354179	0.894531
0.5625	1.16666667	0.5	1.666667	-0.397748	1.268919	-0.101563	1.167357	0.029002427	1.196359032	0.894531
0.625	1.16666667	0.5	1.666667	-0.397748	1.268919	-0.101563	1.167357	0.029002427	1.196359032	1.078125
0.625	1.16666667	0.5	1.666667	-0.397748	1.268919	0.1015625	1.370482	-0.04281311	1.3276685	1.078125
0.6875	1.16666667	0.5	1.666667	-0.397748	1.268919	0.1015625	1.370482	-0.04281311	1.3276685	1.300781
0.6875	1.16666667	0.5	1.666667	-0.397748	1.268919	0.1015625	1.370482	0.042813106	1.413294711	1.300781
0.75	1.16666667	0.5	1.666667	-0.397748	1.268919	0.1015625	1.370482	0.042813106	1.413294711	1.5625
0.75	1.16666667	0.5	1.666667	0.397748	2.064414	-0.179688	1.884727	-0.05662379	1.828102949	1.5625
0.8125	1.16666667	0.5	1.666667	0.397748	2.064414	-0.179688	1.884727	-0.05662379	1.828102949	1.863281
0.8125	1.16666667	0.5	1.666667	0.397748	2.064414	-0.179688	1.884727	0.056623785	1.94135052	1.863281
0.875	1.16666667	0.5	1.666667	0.397748	2.064414	-0.179688	1.884727	0.056623785	1.94135052	2.203125
0.875	1.16666667	0.5	1.666667	0.397748	2.064414	0.1796875	2.244102	-0.07043446	2.17366727	2.203125
0.9375	1.16666667	0.5	1.666667	0.397748	2.064414	0.1796875	2.244102	-0.07043446	2.17366727	2.582031
0.9375	1.16666667	0.5	1.666667	0.397748	2.064414	0.1796875	2.244102	0.070434465	2.314536199	2.582031
1	1.16666667	0.5	1.666667	0.397748	2.064414	0.1796875	2.244102	0.070434465	2.314536199	3

Daubechies Wavelets

The family of compactly supported wavelets constructed by Daubechies in 1988 opened the door to a whole new territory in mathematics. *Compactly supported* means being defined over a finite, usually small, domain. In fact, the impact of her work is so powerful that the Wavelet-Galerkin method should be renamed the Daubechies-Galerkin method.

The fundamental aspect is that the Daubechies set of wavelets provide an orthogonal basis with which to approximate functions. As with all wavelets, the basic recursion, or dyadic, or multi-resolutional equation takes the form

$$\varphi(x) = \sum_k a_k \varphi(2x - k). \qquad (3.1)$$

The a_k's are a collection of coefficients that categorize the specific wavelet basis. The mother wavelet also takes the conventional form

$$\psi(x) = \sum_k (-1)^k a_{1-k} \varphi(2x - k) \qquad (3.2)$$

These formulas are standard for all the wavelets encountered in practice. Daubechies work begins when she sets the rules on how to define the coefficients a_k. First, the scaling function must be normalized so that $\int \varphi dx = 1$. This provides for the *normalization condition*.

$$\sum_{k=0}^{N-1} a_k = 2 \qquad (3.3)$$

Hence, we refer to $\varphi(x)$ and $\psi(x)$ of this form as a *multiplier 2 system*. This also is gives the $2^{j/2}$ term in the DWT significance as a normalizer. Just for reference,

one can generalize wavelet systems to any arbitrary nonnegative integer.

The translates of φ are required to be orthonormal, that is

$$\int \varphi(x-k)\varphi(x-m) = \delta_{k,m} \qquad (3.4)$$

From the scaling relation this implies the condition

$$\sum_{k=0}^{N-1} a_k a_{k-2m} = \delta_{0m} \qquad \text{for m=0, 1, ..., (N/2)-1.} \qquad (3.5)$$

where δ_{ij} is the Kronecker delta symbol. This is the *orthonormal condition*. For coefficients satisfying these two conditions, the functions consisting of translates and dilations of the wavelet function, $\psi(2^j x - k)$, form a complete, orthogonal basis for square integrable functions on the real line, $L^2(\mathbf{R})$. "L" signifies a Lebesque integral, the "2" denotes the integral of the square of the modulus of the function, and **R** means that the independent variable of integration is a number over the whole real line. In other words, this is the space of all functions with a well defined integral of the square of the modulus of the function.

Daubechies also states that if only a finite number of the a_k are nonzero, then φ will have compact support. Thus,

$$\int \varphi(x)\psi(x-m)dx = \sum_k (-1)^k a_{1-k} a_{k-2m} = 0$$

allows the translates of the scaling function and wavelet to define *summable* orthogonal subspaces! This will be a significant aspect.

Smooth scaling functions arise as a consequence of the degree of approximation of the individual translates. The conditions that the monomials

$1, x, \ldots, x^{p-1}$ be expressed as a linear combination of the translates of $\varphi(x-k)$ is implied by the condition

$$\sum_{k=0}^{N-1}(-1)^k k^m a_k = 0 \qquad \text{for m=0, 1, ..., (N/2)-1.} \qquad (3.6)$$

The above equation is referred to as the *moment zero condition.*

Throughout these equations, j is the dilation parameter or simply the scale. In the approximation to solutions of differential equations, j is also called the approximation level. For a certain value of j and N, the support of the scaling function $\varphi(2^j x-k)$ is given as follows:

$$\text{supp}\,(\varphi(2^j x-k)) = \left[\frac{k}{2^j}, \frac{N+k-1}{2^j}\right]$$

These three conditions make it possible to express equations in the now familiar form:

$$f(x) = \sum_k 2^{j/2} c_k \varphi(2^j x-k) \qquad (3.7)$$

Here it is worth emphasizing that there are two convergence properties used in the above expansion. One is the uniform convergence for the level of approximation in relation to the scale j and the other is the rapid convergence for smoother scaling functions which relate to the variable N. These properties are not shared at the same time by the usual classical orthogonal functions. The trade off for the N and j is very important. The bigger j and N gives higher accuracy and faster convergence; it also gives a larger system of equations and a larger number of connection coefficients needed to be calculated. A proper couple of j and N give both rapid convergence and satisfactory accuracy.

For a better understanding of this, please refer to the paper by Qian and Weiss[6] This paper goes into more detail about this aspect. The authors states that a value greater than N=20 should not be considered. By looking at the error graphs in this paper (page 165), one should not use a Daubechies wavelet with N>12. However, on page 160, Qian and Weiss[6] state that for N=6, the Daubechies-Galerkin method actually solves the Helmholtz equation in fewer operations than the dealiased FFT algorithm (which uses shifted grids to eliminate aliasing terms). Also, considering that reputable sources have published values for N=6 Daubechies wavelet (or simply D6) connection coefficients and moments, I have chosen to perform all operations using D6.

Variational Formulation

This process begins with a differential equation. Suppose you are given a differential equation defined over some boundary or interval. We let the differential equation take the form

$$Au = f$$

within the boundary or interval where A is the differential operator and the form

$$Bu = g$$

on the boundary or interval where B is the boundary operator.

Let us consider the problem of the following differential equation:

$$-\frac{d}{dx}\left[a(x)\frac{du}{dx}\right] = q(x) \qquad \text{for } 0<x<L$$

with the following boundary conditions:

$$u(0)=u_0 \qquad \text{and} \qquad \left(a\frac{du}{dx}\right)_{x=L} = Q_0$$

In these expressions, a and q are functions of x, and u_0 and Q_0 are specified values. L is the length of the one-dimensional domain. u is the *dependent variable*. We will take this problem to have nonhomogeneous boundary values, which means the specified values u_0 and Q_0 are not equal to zero, for arguments sake. This type of equation may be commonly found in the areas of heat transfer and fluid flow, to name only two applications.

To start the variational formulation, all the terms must be moved to one side of the equation. Then, the equation is multiplied by a function w called a test function or *weight* function. Next, integrate over the domain $\Omega = (0,L)$.

$$0 = \int_0^L w\left[-\frac{d}{dx}\left(a\frac{du}{dx}\right) - q\right]dx$$

The resulting equation above is called the *weighted-integral* or *weighted-residual* statement. The expression within the brackets may be called the residual. Since the function w is called the weight function, it is easy to see where the term "weighted-residual" came from. The residual does not equal zero when replaced by its approximation.

The weight function is any function that is zero on the differential boundary and such that the integral makes sense. In essence, it can be any nonzero, integrable function.

The second important step is to integrate the first term of the expression by parts. Recall that integrating by parts is simply using the equivalent expressions below.

$$\int_a^b w dv = \left[wv\right]_a^b - \int_a^b v dw$$

At any rate, the expression $0 = \int_0^L w\left[-\frac{d}{dx}\left(a\frac{du}{dx}\right) - q\right]dx$ becomes

$$\int_0^L \left(\frac{dw}{dx}a\frac{du}{dx} - wq\right)dx - \left[wa\frac{du}{dx}\right]_0^L = 0$$

Notice that the weight function is required to be differentiable at least once, ruling out constants as valid weight functions. Of special note, this is called the *weak form* of the original differential equation. "Weak" refers to the reduced (i.e.,

weakened) continuity of u, which is required to be twice-differentiable in the weighted integral form, but only once-differentiable in the weak form.

Boundary conditions should be given some attention now. Boundary conditions are of two types: *natural* or *Neumann* and *essential* or *Dirichlet* conditions. The following rule is used to identify the natural boundary conditions and their form. After completing the integration by parts, examine all boundary terms of the integral statement. The boundary terms will involve both the weight function and the dependent variable. Coefficients of the weight function and its derivatives in the boundary expressions are termed the *secondary variables* (SV). Specification of secondary variables on the boundary constitutes the *natural boundary conditions* (NBC). For this example, the boundary term is w(a du/dx). The coefficient of the weight function is a du/dx. Therefore, the secondary variable is of the form a du/dx. The secondary variables always have physical meaning, and are often quantities of interest.

The dependent variable of the problem, expressed in the same form as the weight function appearing in the boundary term, is called the *primary variable* (PV), and its specification on the boundary constitutes the *essential boundary conditions* (EBC). Above, the weight function appears in the boundary expression as w. Therefore, the dependent variable u is the primary variable, and the EBC involves specifying u at the boundary points.

It should be noted that the number and form of the primary and secondary variables depend on the order of the differential equation. The number of primary and secondary variables is always the same, and with each primary

variable there is an associated secondary variable. However, only one of the pair may be specified at a point on the boundary!

The third and last step is to incorporate the boundary conditions. We require the weight function w to vanish at the boundary points where the essential or Dirichlet conditions occur. Accordingly, the weight function w is required to satisfy the following conditions

$$W(0) = 0, \qquad \text{because } u(0) = u_0$$

This leaves our equation of the form:

$$0 = \int_0^L \left(\frac{dw}{dx} a \frac{dr}{dx} - wq \right) dx - w(L)Q_0 \qquad \text{where} \qquad Q_0 = \left(a \frac{du}{dx} \right)_{x=L}$$

This completes the development of the weak or *variational form* of a differential equation.

Approximation Methods to the Variational Form

To solve the variational form, we will use an approximation. This will be done out of necessity basically. For large or hard problems, an exact solution may be overly difficult. On a Global level, the variational form may be solved using the Rayleigh-Ritz method. It also may be solved on a local level using what is called a weighted residual method. These methods include the Galerkin, Least Squares, and Collocation methods. This paper will focus on the Galerkin method.

In any case, we will make the following assumption:

$$u \cong u^* = \sum_n \Psi_i c_i \qquad (5.1)$$

This says that the exact solution u is approximated by u* which is equal to the summation of approximation functions (Ψ) multiplied by constants (c). This summation may now be substituted into the variational form. Since u* is only an approximation of u, the resulting Residual equation is not equivalent to the equation into which it was substituted. That is why the Residual does not equal zero, as mentioned in the earlier section.

In order for an approximation technique to be considered Galerkin, the weight function w in the variational form must be exchanged with Ψ. Ψ is sometimes referred to as a shape function. This simple fact that Ψ=w is what separates the Galerkin method from other methods.

The solution to the Galerkin method is found by solving the weighted-integral form. Its solution is pinpoint accurate with zero percent difference from

the exact solution, at the points where a solution is found. With this knowledge, it is easy to see why the Galerkin method has gained so much popularity over the past fifty years.

As stated above, shape functions are function approximations. In variational methods, the shape function must fulfill certain requirements in order for the approximation solution u^* to be convergent to the actual solution u as the number of elements increase. These are:

1. The approximate solution should be continuous over the element, and differentiable.
2. It should be a complete polynomial, i.e., include all lower-order terms up to the highest order used.
3. It should be an interpolant of the primary variables at the nodes of the finite element.

Galerkin Method

The Galerkin Method is a weighted residual method of approximating solutions to differential equations. The Least Squares Method and Collocation are also weighted residual methods, however, we will be focusing on the Galerkin method. It is an incredibly accurate method that has gained popularity during the past fifty years. The actual variables that will be solved for (displacement for a structural problem, temperature for a thermal problem, etc...) are pinpoint accurate with zero percent difference! It is this aspect which makes the Galerkin Method so well received.

As stated above, the Galerkin Method is a weighted residual method. The first step of the method is to make a substitution for whatever variable the functional is in terms of. For instance, if the functional is in terms of u, and f is a function of x,

$$L(u) + f = 0 \qquad (6.1)$$

We must make a substitution for all u's as follows:

$$u \cong u^* = \sum_{i}^{n} \Psi_i U_i \qquad (\text{eqn } 5.1)$$

where u^* is an approximation of u, Ψ_i is called the shape functions and U_i are the values of the solution. Shape functions are function approximations.

Now, the Residual, R, is as follows:

$$R = L(u^*) + f \neq 0 \qquad (6.3)$$

To get a solution to the problem, we must multiply R by a weighting function, $W_i(x)$, and integrate over the element size, h, and set this equal to zero.

$$\int_0^h W_i(x)Rdx = 0 \qquad \text{i=1,2,3,...,n} \qquad (6.4)$$

However, for Galerkin Method, $W_i(x) = \Psi_i(x)$. Now, Integrate n times per element. The number n will be determined by the type of shape function used. The resulting equations will need to be assembled in a master equation which will take the form

$$[A](U_i) = (F)$$

The matrix A will be a square matrix. The solution vector (U_i) may be found now using standard Linear Algebra techniques and whatever Boundary Conditions exist in the problem.

Take as an example the differential equation:

$$3U'' + 4U' - 5x = 0 \qquad 0 \le x \le L$$

The primes that follow the variable denote the derivative with respect to displacement.

$$U' = \frac{dU}{dx}, U'' = \frac{d}{dx}\left(\frac{dU}{dx}\right)$$

Therefore,

$$R = (3\sum \Psi_i' U_i)' + (4\sum \Psi_i' U_i) - 5x \neq 0$$

Following the procedure,

$$\int W_j R dx = 0, W_j = \Psi_j$$

This is what distinguishes the Galerkin method from other weighted-residual methods.

$$\int_0^h \Psi_j 3(\sum \Psi_i' U_i)' dx + \int_0^h \Psi_j 4(\sum \Psi_i' U_i) dx = \int_0^h \Psi_j 5x dx$$

Now, the equation is ready to be solved. The first term on the left side of the equation must be integrated by parts. The term on the right must be evaluated at $x = x_k + x$, where x_k is a constant. This means that the constant term will change as the elements progress. In other words, x_k will equal the previous x_k plus element size h. Each element will have its own set of equations. A Global equation must be formed by assembly of the element equations. As stated earlier, standard Linear Algebra techniques may be used to solve for the values of U.

The requirements for shape functions may be found on page 25 in Chapter Four. There are three common types of shape functions which satisfy these requirements. They are linear, quadratic, and cubic hermite. Linear elements have two equations per element (n=2) and are affected only by two nodes. The nodes are on opposite sides of the element. This makes straight-line approximations. The quality of the results will be more dependent on the number of elements (the more elements the better). The Quadratic elements make use of three nodes (n=3). The nodes are located at the beginning, middle, and end of the element. The three nodes provide for smoother fitting of the approximation. Cubic Hermite uses four equations (n=4). However, there are two nodes located at the beginning and the end of the element. At each node there are two equations, one for position and one for slope. This makes for a very smooth approximation. The cubic Hermite gives the best results. All shape

elements will give the same precise values at the nodes, but the different types will give progressively better results between the nodes. Therefore, a trade off exists. Cubic Hermite will not need as many elements, but will need more equations per element. There is a distinct relationship between the number of calculations needed and type of shape function. It is up to the person solving the problem to decide which shape function to use.

Wavelet-Galerkin Method

The Wavelet-Galerkin Method is very interesting. The method uses the scaling functions of the wavelet as shape functions. The name "Wavelet-Galerkin" is actually a misnomer due to this fact. Wavelets themselves are never actually used, only there scaling functions. Since the scaling function is being used, the scale *j* must be chosen in order to complete the analysis. Any scale greater than one will do. However, the larger the scale the more accurate the approximation will be. This should be more than apparent from the previous sections. Of course, the scaling which is to be used must be chosen as well. The wavelets of choice are Daubechies wavelets.

The first step in forming the approximation is forming the Residual equation by making the substitution

$$u(x) \cong \sum_k c_k 2^{j/2} \phi(2^j x - k), \qquad k \in \mathbf{Z} \qquad (7.1)$$

Notice that this is eqn 3.7 in another form. This equation should be used in place of eqn 5.1 in Chapter 5; in this way the scaling function becomes a replacement for the shape function. Once this substitution has been made, all that remains is finding out what the coefficients c_k are and plugging them back into the above equation. This can be done by simply following the Galerkin method. Since the scaling function $\phi(2^j x - k)$ is the shape function, operation of the inner product with the scaling function on the residual equation is performed. This will give an equation of the form

$$\int \phi(2^j x - p) \left[L(\sum c_k 2^{j/2} \phi(2^j x - k) \right] dx = 0. \qquad (7.2)$$

The key to solving this equation lies in determining what are called "connection coefficients". These are the values of an inner product of a scaling function with one or more of its derivatives. A second order differential equation would contain at least the following connection coefficient

$$\int \phi_{xx}(2^j x - k)\phi(2^j x - p)dx$$

where the notation ϕ_{xx} denotes the second derivative of the scaling function with respect to x. Without these known values the Wavelet-Galerkin method will not work! The way to determine these coefficients is described in the paper by Latto, et al.

Now, everything in the equation is obtainable except for the coefficients c_k. There are a couple of substitutions that prove to be useful. Namely, these are

$$y = 2^j x \qquad \text{and} \qquad C_k = 2^{j/2} c_k.$$

A system of matrices may be made and then the column vector containing the c_k's may be solved for using Gaussian elimination or another method. The treatment of the boundary conditions depends upon the specific problem to be solved. The treatment according to the example problem I have chosen will be discussed Chapter 9.

This explanation of the Wavelet-Galerkin method may seem brief. However, I feel that all questions should be answered by following the example problem. It also should be recognized that the main theory behind using this method is simply substituting a scaling function into the Galerkin method.

Example – Analytical

This section will solve the problem I have chosen analytically. This is an *exact* solution. The purpose of this paper is to compare the three methods: analytical, Galerkin, and Wavelet-Galerkin. At any time if a percent difference is referred to, the analytical method is the known solution for comparison.

A simple homogeneous equation with essential or Dirichlet boundary conditions is chosen. This equation may be used to describe a freely vibrating spring-mass system, to name one application. It is as follows

$$u'' + \frac{k}{m}u = 0 \qquad u(1)=1.5,\ u(4)=0.75$$

k will equal 3 and m will equal 2. Therefore, k/m will equal 3/2.

From any course in Differential Equations or course book[11], it can be shown that the solution to the homogeneous equation will take the form

$$u_h(x) = A\cos(\sqrt{1.5}x) + B\sin(\sqrt{1.5}x)$$

Substituting the values of the boundary conditions into the equation above will solve for the coefficients A and B. This will result in two equations and two unknowns. Very simply, the resulting analytical solution is

$$u(x) = 4.29195671462\cos(\sqrt{1.5}x) + 0.047015528941\sin(\sqrt{1.5}x)$$

This was a very simple solution. There would be no need to use an approximation method for such a problem. However, as all scientists and engineers realize, times do arise in which an approximation is called for. No one

would ever use an approximation for this problem. However, the insight gained by solving this problem will apply to other cases in which an approximation method would be applicable.

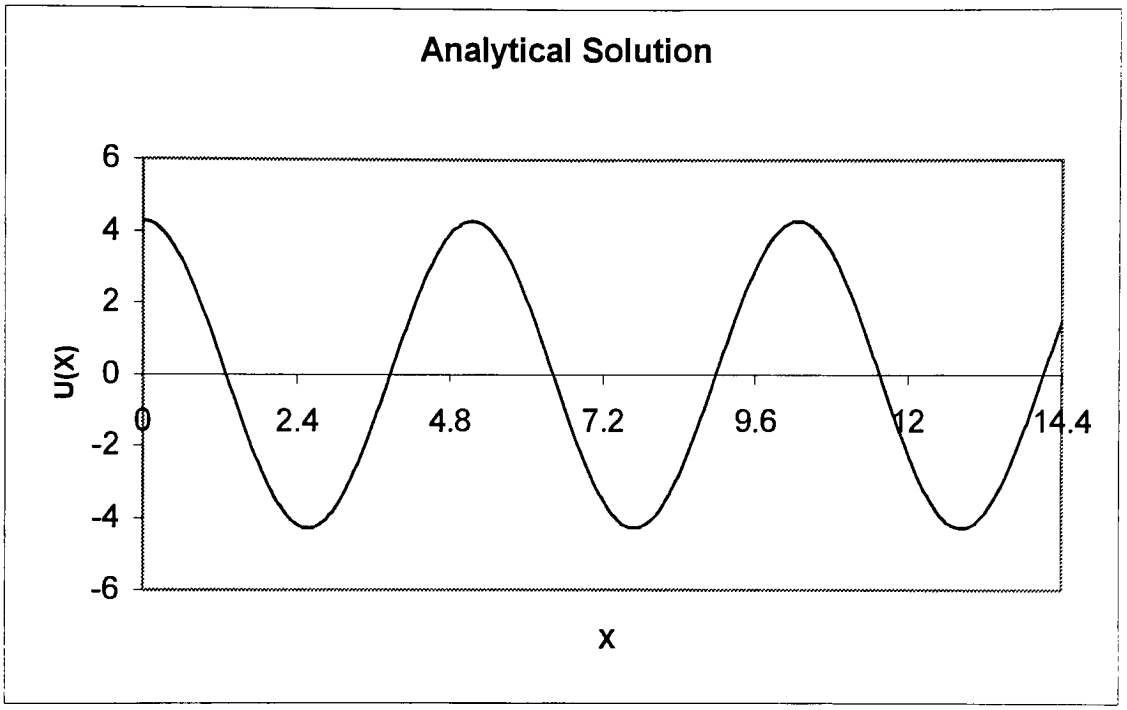


Figure 8.1

Example Wavelet-Galerkin

We start with the following problem.

$$u'' + \frac{3}{2}u = 0 \qquad u(1)=1.5,\ u(4)=0.75$$

The first thing is to substitute the scaling function approximation to u(x).

$$u(x) \cong \sum_k c_k 2^{j/2} \phi(2^j x - k) \qquad \text{(eqn 3.7)}$$

At this time it should be noted that a scale of j=4 is selected. This is purely arbitrary. Any scale greater than 1 would suffice. Also, the Daubechies scaling function for the length 6 wavelet was selected.

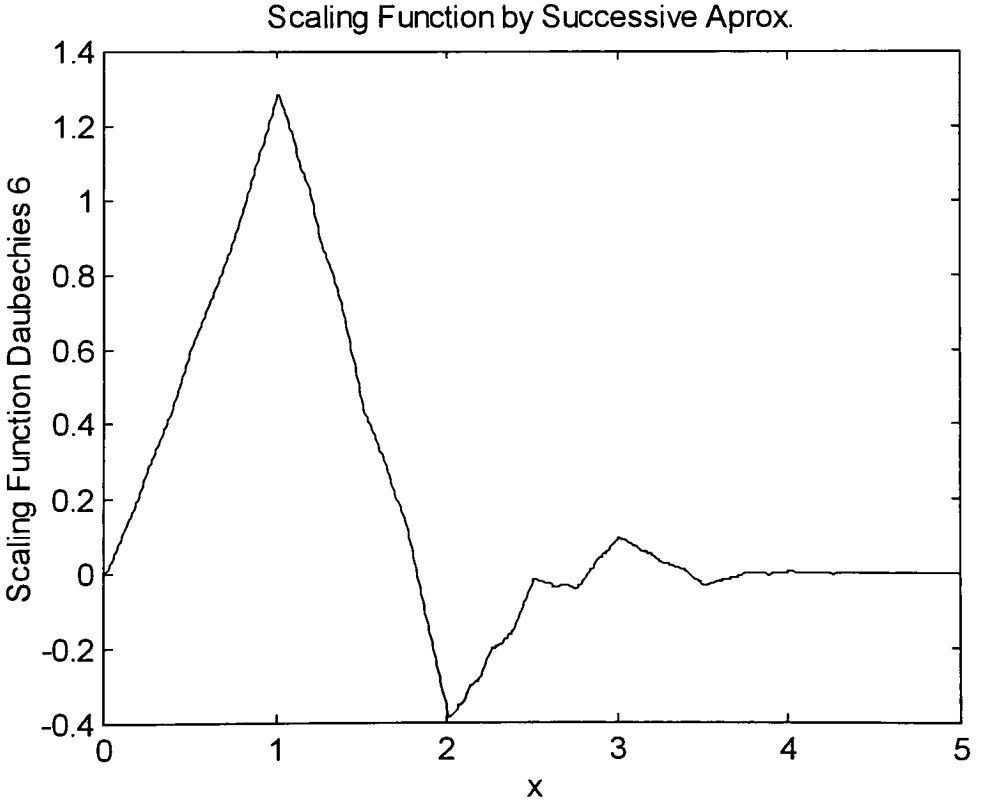


Figure 9.1

The next step would be to plug this approximation back into the original equation, forming the residual equation.

$$\frac{d^2}{dx^2}\sum_k c_k 2^{J/2}\phi(2^J x-k) + {}^3/_2\sum_k c_k 2^{J/2}\phi(2^J x-k) \neq 0$$

The substitutions $y = 2^j x$ and $C_k = 2^{j/2}c_k$ may now be made, as well as some simplification. This results in

$$2^{2j}\sum_k C_k\phi_{xx}(y-k) + {}^3/_2\sum_k C_k\phi(y-k) \neq 0$$

where the notation ϕ_{xx} denotes the second derivative of ϕ with respect to x. The inner product of this residual equation with the scaling function $\phi(y-p)$ may now be computed. This gives

$$2^{2J}\sum_k C_k\int\phi_{xx}(y-k)\phi(y-p)dy + {}^3/_2\sum_k C_k\int\phi(y-k)\phi(y-p)dy = 0 \qquad (9.1)$$

It is interesting to note that the second integral can be simplified due to the orthogonality of the Daubechies wavelet. It simplifies as follows

$$\int\phi(y-k)\phi(y-p)dy = \delta_{pk} \qquad (\textit{orthonormal condition})$$

Therefore, all that is left of this integral is ${}^3/_2C_p$. This simplifies matters greatly. The integral $\int\phi_{xx}(y-k)\phi(y-p)dy$ defines what is called the connection coefficient. The values for the connection coefficients of this particular type are already defined in the paper by Latto, et al[8]. They will be used discretely in the upcoming computations. Let the symbol Ω represent the coefficients such that

$$\Omega_{p-k} = \int\phi_{xx}(y-k)\phi(y-p)dy \qquad (9.2)$$

The working equation, (eqn 9.1), may now be expressed by

$$2^{2j}\sum_k C_k \Omega_{p-k} + \frac{3}{2}C_p = 0 \qquad (9.3)$$

This equation may be represented in matrix form by

$$TC = D$$

The matrix T will be of the form

$$T = 2^{2j}\begin{bmatrix}
\Omega_0 + 2^{-2j}(1.5) & \Omega_{-1} & \Omega_{-2} & & \cdots & \cdots & & \Omega_1 \\
\Omega_1 & \Omega_0 + 2^{-2j}(1.5) & \Omega_{-1} & & & & & \\
\cdots & \Omega_1 & \cdots & \Omega_{-1} & & & & \\
\Omega_{N-2} & & \Omega_1 & \cdots & \Omega_{-1} & & & \cdots \\
0 & \cdots & & \Omega_1 & \cdots & \Omega_{-1} & & \cdots \\
\cdots & 0 & & & \Omega_1 & \cdots & \Omega_{-1} & \\
0 & \cdots & & & & \Omega_1 & \cdots & \\
\Omega_{2-N} & & \cdots & & & & \Omega_1 & \\
\cdots & & & & & & & \Omega_{-1} \\
\Omega_{-1} & \Omega_{-2} & \cdots & 0 & & & & \Omega_0 + 2^{-2j}(1.5)
\end{bmatrix}$$

The N here represents the size of the Daubechies wavelet. In this case N=6, as mentioned before. It should be noted that all the columns in the matrix T are the same as the previous column but shifted down one row. This is what is called a circulant matrix and the first column in known as the convolution kernel. This will be of importance when the system is solved.

The matrix C will be a column vector containing the unknown coefficients C_k. The matrix D will be defined using the paper by Oyoshi, et al[3] The matrix D will allow us to incorporate the boundary conditions. It will take the form

$$D = \begin{bmatrix} u_a \\ 0 \\ \dots \\ 0 \\ u_b \end{bmatrix}$$

size $(2^j b - 2^j a + N + 2W) \times 1$

In this matrix, u_a and u_b are the values at the boundary conditions. For this example, a = 1, b = 4, u_a = 1.5, and u_b = 0.75. The dimensions of these matrices will be dictated by the scale selected, the scaling function selected, and the values of the end conditions. Namely, T will be a square matrix of size $(2^j b - 2^j a + N + 2W) \times (2^j b - 2^j a + N + 2W)$. W represents the additional number of wavelet components to expand the boundary conditions by one support of the scaling function. Basically what W does is to increase the accuracy at the boundary conditions. Wavelets have a hard time analyzing functions close to the end conditions. They need some room between the boundary conditions and the point of initial approximation. The extra space created by W gives the wavelets the room they need. The size of W is usually equal to N+1, as is the case in this example. The idea of this treatment has been reported in numerous papers including the ones by Oyoshi, et al[3] and Amaratunga, et al[4,5].

At any rate, it is important to note that both C and D are column vectors of size $(2^j b - 2^j a + N + 2W)$. Notice that b and a have a scale of 2^j in front of them. This implies that the C_k's that make up the matrix C are already at scale 2^j! Therefore, the coefficient in front of the T matrix should be 2^j and not 2^{2j}. The most important thing is to change the first term of the convolution kernel to $\Omega_0 + 2^{-j}(1.5)$ instead of $\Omega_0 + 2^{-2j}(1.5)$. This will give the solution the proper magnitude when solved.

We may exploit the properties of the circulant matrix when solving. Gaussian elimination is equivalent to convolution of the D matrix with the convolution kernel. This may further be exploited by the fact that convolution in physical space is equal to multiplication in frequency space. Let K be the convolution kernel and FK be the Fast Fourier Transform (FFT) of K. FD and FC will be the FFT of D and C, respectively. Therefore,

$$FC = FD \,./\, FK$$

Where . / represents component by component division. The inverse FFT of FC will give the desired C vector. The use of the FFT in this case eases programming and makes computation more efficient. The computer program FastC performs these operations.

Now that the C's have been obtained they may be used to approximate the solution u using equation (1). Remember that C_k is equal to $2^{j/2}c_k$ and y is equal to $2^j x$. The C_k's may now be multiplied by the scaling function. Daubechie's scaling functions can only be approximated and this will be done using the program psa.m supplied in the book by Burrus, et al[2]. The approximation is a 640-term expansion. This implies that each k is 128, since 640/5 equals 128. The 5 comes from the length of the D6 scaling function. The approximation to u(x) is completed using the computer program shift. It may be followed in the appendix.

The results are very interesting. The resulting vector is 9216 components long.

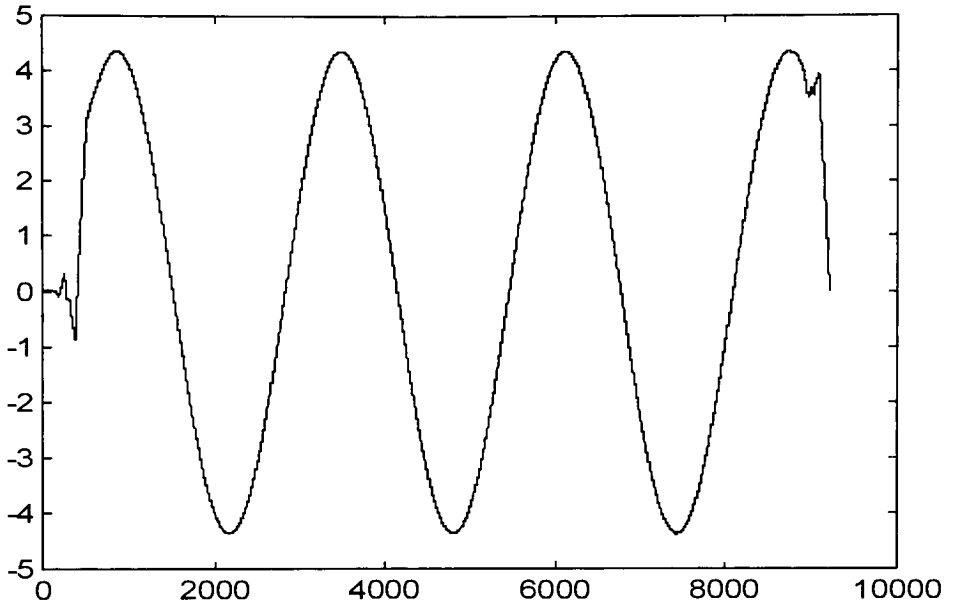


Figure 9.2

Now, $^{9216}/_{640}$ is equal to 14.4. Even though I started this approximation looking for the interval from 1 to 4 (length 3), I got an approximation of length 14.4! This is not bad. Admittedly, this does not seem correct at first, but upon inspection, I've found it to be legitimate. The plot of the Wavelet-Galerkin solution with the first and last 896 terms removed (W*128=896) shows this well.

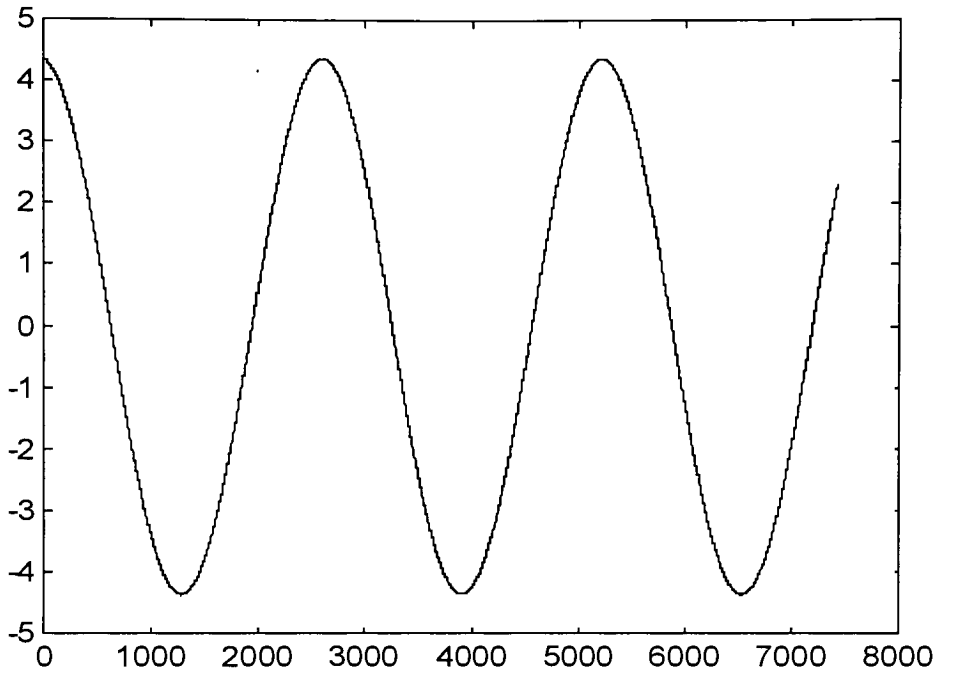
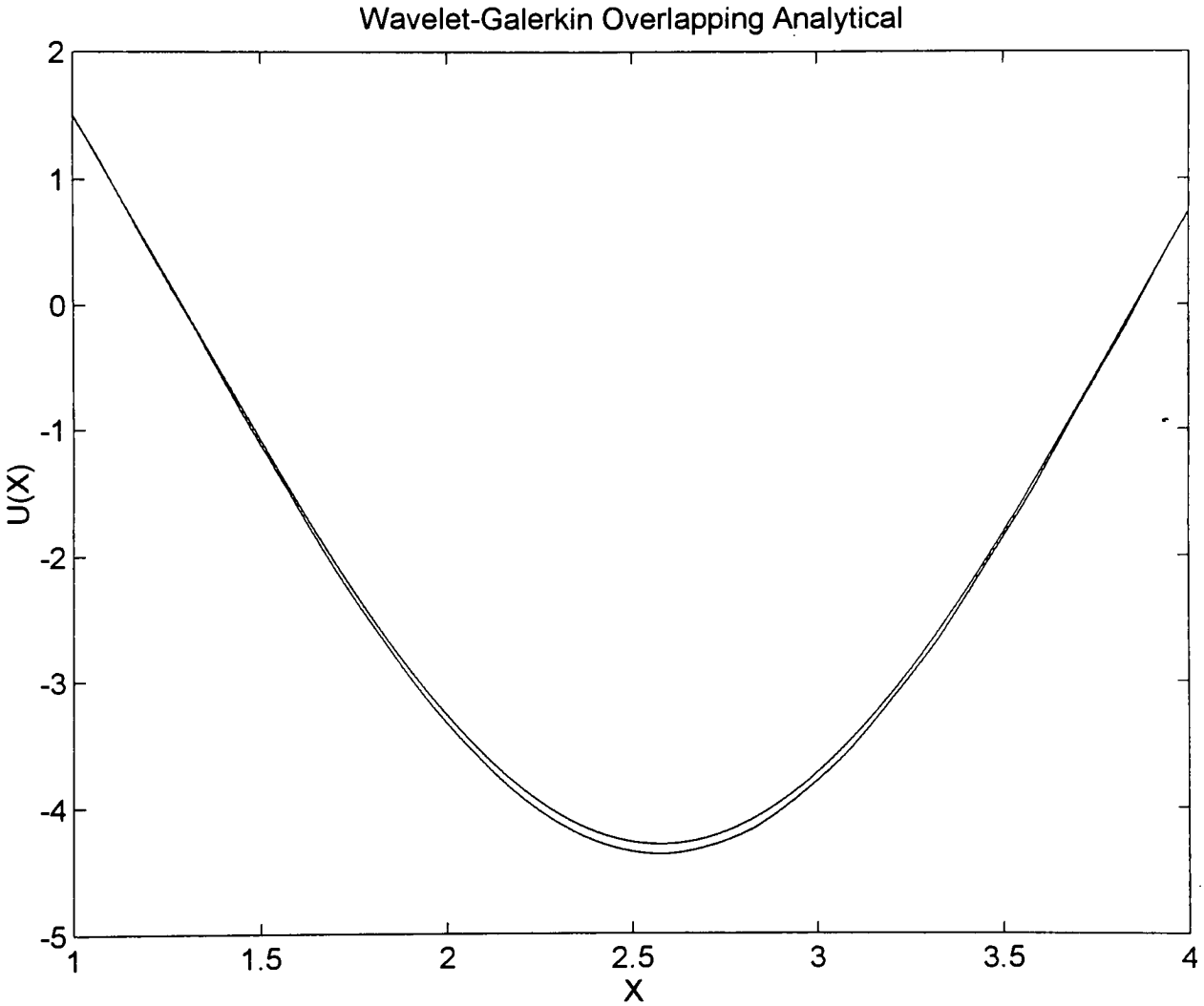


Figure 9.3

To approximate the interval from 1 to 4, all that needs to be done is finding the values of the Dirichlet conditions. In this case, 1.5 and 0.75. Simply analyze the data looking for these values in the ranges where they should be and plot these components against a linespace from 1 to 4. The result is less than a two percent error.


Wavelet-Galerkin Overlapping Analytical
2
1
0
-1
-2
-3
-4
-5
U(X)
1
1.5
2
2.5
3
3.5
4
X

Example Galerkin-Quadratic

Again, we will start with the following problem

$$u'' + {}^{3}/_{2}u = 0 \qquad u(1)=1.5,\ u(4)=0.75$$

To start the approximation, quadratic elements will be chosen. These give better approximation between the nodes. Six elements will be used, space 0.5 units apart.

The first thing to do is define the approximation of u.

$$u \cong u^* \qquad u^* = \sum_i \Psi_i(x) U_i(x) \qquad \text{(eqn 5.1)}$$

For quadratic elements, i=1,2,3. For an element spanning 0.5 units, h(element size) = 0.25. The Ψ's are as follows.

$$\Psi_1 = -\frac{x}{2h}\left(1-\frac{x}{h}\right) \qquad \Psi_2 = 1-\left(\frac{x}{h}\right)^2 \qquad \Psi_3 = \frac{x}{2h}\left(1-\frac{x}{h}\right)$$

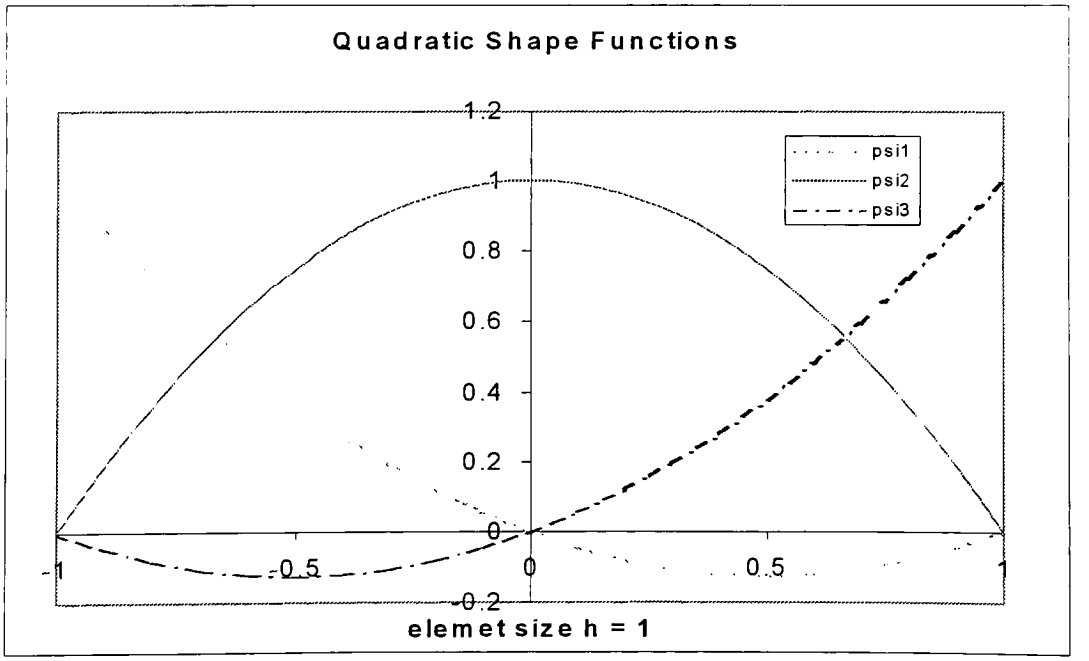


Figure 10.1

The Residual Equation may now be formed by substituting the formula for the approximation u* into the original equation.

$$R = \left(\sum \Psi_i' U_i\right)' + 3/2 \sum \Psi_i U_i \neq 0$$

The primes represent derivatives with respect to x. The residual equation R is now ready for the inner product with the shape functions.

$$\int_{-h}^{h} \Psi_i \left(\sum \Psi_i' U_i\right)' dx + 3/2 \int_{-h}^{h} \Psi_i (\sum \Psi_i U_i) dx = 0$$

Integration by parts yields

$$-\int_{-h}^{h} \Psi_i' \left(\sum \Psi_i' U_i\right) dx + 3/2 \int_{-h}^{h} \Psi_i (\sum \Psi_i U_i) dx + \left[\Psi_i \sum \Psi_i' U_i\right]_{-h}^{h} = 0 \qquad (10.1)$$

The last term in brackets may be considered to be an equivalent body force. All of the inner body force terms (all except at the end points) will cancel upon matrix assembly. All that will be left will be the body forces at the endpoints. Since there are no body forces described in the problem statement, they can be ignored, or set equal to zero.

From the rest of the equation, the general matrix form may be described. After substituting the values of Ψ into eqn 10.1 and evaluating, we find

$$\left\{ \frac{1}{6h} \begin{bmatrix} 7 & -8 & 1 \\ -8 & 16 & -8 \\ 1 & -8 & 7 \end{bmatrix} + \frac{-1.5h}{15} \begin{bmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{bmatrix} \right\} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \qquad (10.2)$$

This may be verified. The first row third column of the first matrix is 1/6h.

$$\int_{-h}^{h} \Psi_1' \Psi_3' dx = \int_{-h}^{h} \left(\frac{x}{h^2} - \frac{1}{2h}\right)\left(\frac{x}{h^2} + \frac{1}{2h}\right) dx = \frac{1}{6h}$$

The second row second column of the second matrix is –1.5*16h/15.

$$-1.5\int_{-h}^{h}\Psi_2^2 dx = -1.5\int_{-h}^{h}\left(1-\left(\frac{x}{h}\right)\right)^2 dx = \frac{-1.5(16h)}{15}$$

Eqn 10.2 will now simplify into

$$\frac{1}{120}\begin{bmatrix} 548 & -646 & 83 \\ -646 & 1232 & -646 \\ 83 & -646 & 548 \end{bmatrix}\begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \qquad (10.3)$$

Six quadratic elements results in thirteen nodes. The assembled matrix is as follows

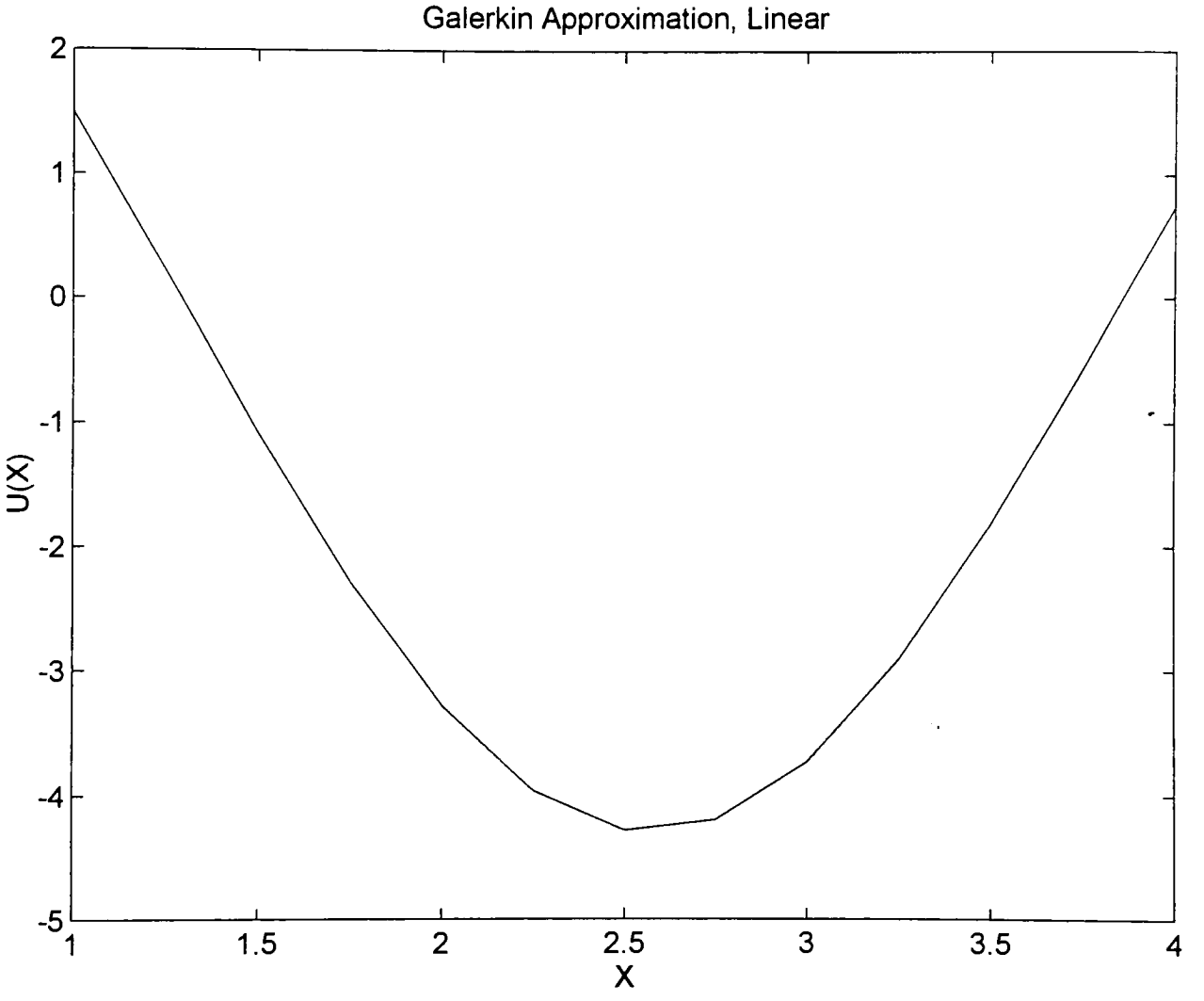
$$\frac{1}{120}\begin{bmatrix} 548 & -646 & 83 & 0 & \cdots & & \cdots & & \cdots & & 0 \\ -646 & 1232 & -646 & & & & & & & & \cdots \\ 83 & -646 & 1096 & -646 & 83 & & & & & & \\ 0 & & -646 & 1232 & -646 & & & & & & \cdots \\ & & 83 & -646 & \cdots & \cdots & \cdots & & & & \\ \cdots & & & & \cdots & \cdots & \cdots & & & & \\ & & & & \cdots & \cdots & \cdots & \cdots & & & \cdots \\ & & & & & & & \cdots & & & \\ \cdots & & & & & & & & \cdots & & \\ & & & & & & & & \cdots & & 0 \\ & & & & & & & & 1096 & -646 & 83 \\ & & & & & & & & -646 & 1232 & -646 \\ 0 & & \cdots & & \cdots & & & 0 & 83 & -646 & 548 \end{bmatrix}\begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \\ U_9 \\ U_{10} \\ U_{11} \\ U_{12} \\ U_{13} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

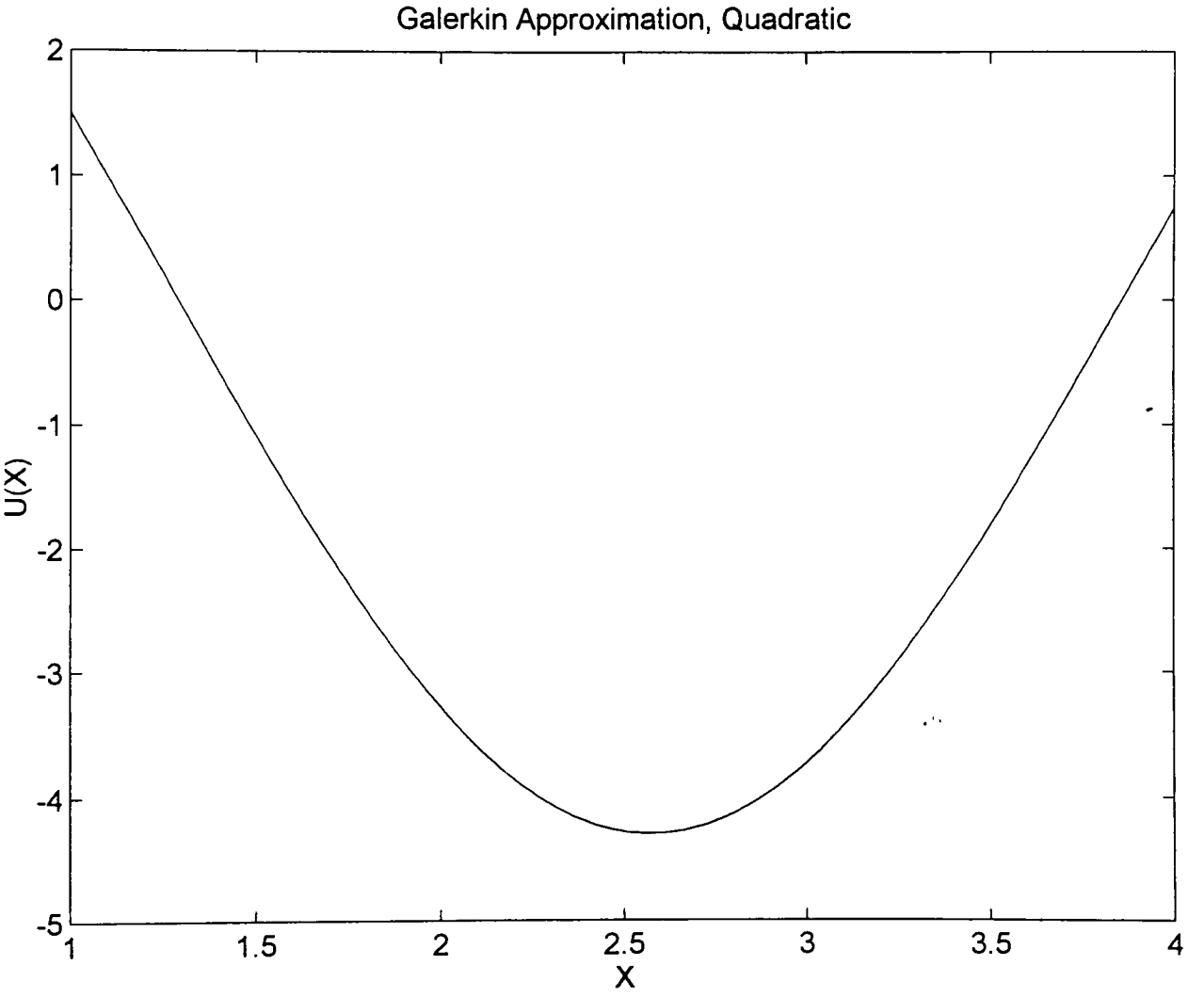
The values of U_1 and U_{13} are known to be 1.5 and 0.75 respectively. These values may be plugged into the U vector and the system of equations may be reduced down to an 11 x 11 system.

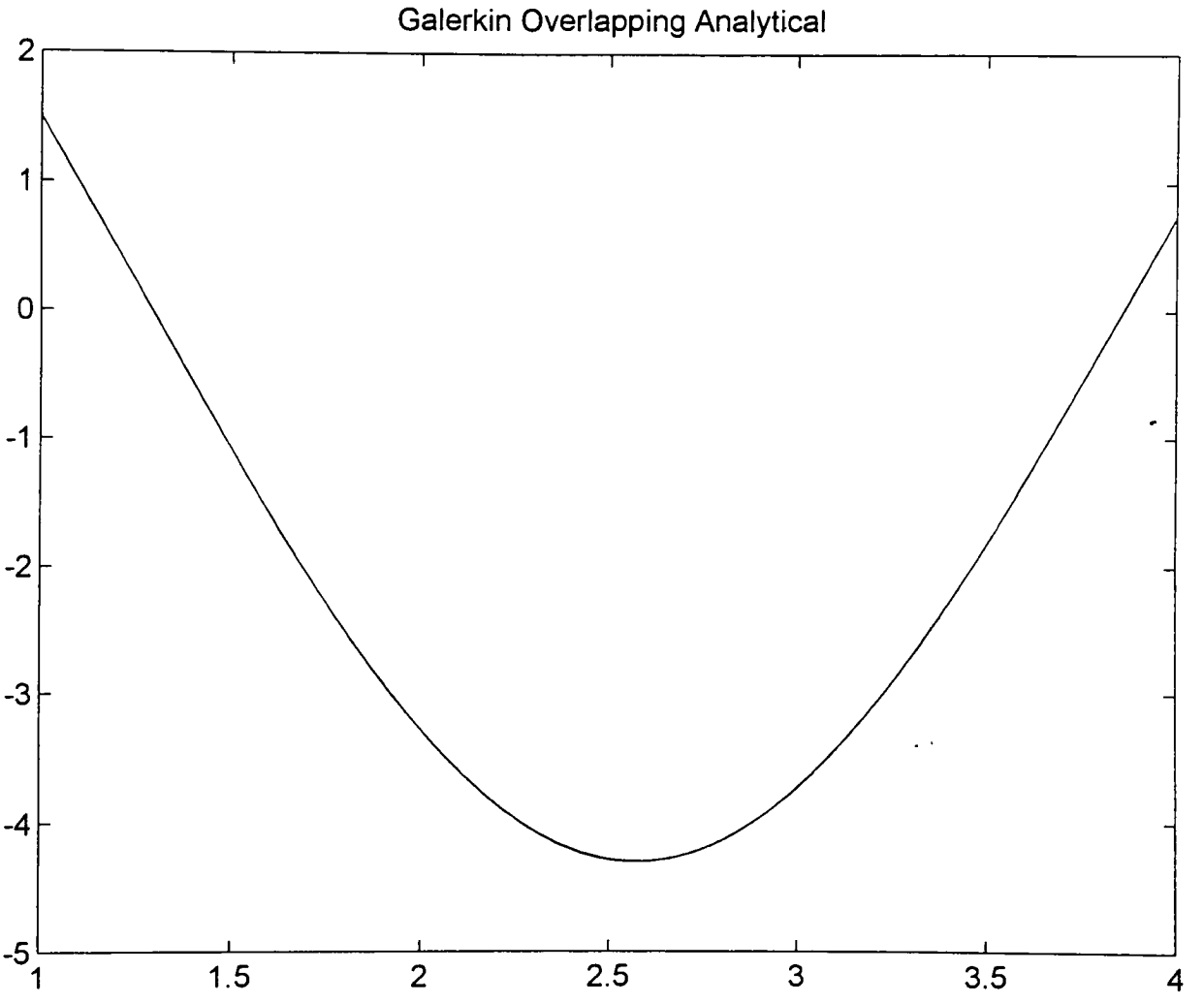
$$\frac{1}{120}\begin{bmatrix} 1232 & -646 & 0 & & \cdots & & \cdots & & & 0 \\ -646 & 1096 & -646 & 83 & & & & & & \\ 0 & -646 & 1232 & -646 & & & & & & \\ & 83 & -646 & 1096 & & & & & & \\ & & & & \cdots & & & & & \\ \cdots & & & & & \cdots & & & & \\ & & & & & & \cdots & & & \\ \cdots & & & & & & \cdots & & & \\ & & & & & & & \cdots & -646 & 0 \\ & & & & & & 83 & -646 & 1096 & -646 \\ 0 & & \cdots & & \cdots & \cdots & & 0 & -646 & 1232 \end{bmatrix} \begin{pmatrix} U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \\ U_9 \\ U_{10} \\ U_{11} \\ U_{12} \end{pmatrix} = \begin{pmatrix} 8.075 \\ -1.0375 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.51875 \\ 4.0375 \end{pmatrix}$$

This matrix is now ready to be solved. Gaussian elimination was used and the values for the U vector were obtained.

The results were plotted using Matlab. The use of quadratic elements to interpolate the function between nodes proved to be *very* effective in approximating the function. Throughout, there was less than 0.2% difference between this method and the analytical.


Galerkin Approximation, Linear
2
1
0
-1
-2
-3
-4
-5
U(X)
1
1.5
2
2.5
3
3.5
4
X


Galerkin Approximation, Quadratic
U(X)
X
2
1
0
-1
-2
-3
-4
-5
1
1.5
2
2.5
3
3.5
4


Galerkin Overlapping Analytical
2
1
0
-1
-2
-3
-4
-5
1
1.5
2
2.5
3
3.5
4

Conclusions

The Wavelet-Galerkin method has proven to be a valid approximation method. Although it was not as precise as the standard Galerkin method, remember that as scale of j=4 was used. A larger scale of say j=6 would have obviously given it a better resolution. Also, the Daubechies D6 wavelet was used. For a truly accurate approximation, it is best to use a wavelet of higher order.

The good part is that it was fairly simple to set up, suprisingly. In fact it was much easier to set up the analysis than the standard Galerkin method. The whole key is knowing the connection coefficients. One of my suggestions for future mathematical pursuit would be to create a table of connection coefficients for the family of Daubechies scaling functions. This would probably be best left to the CRC press. The paper by Lattto, et al[8] describes how to do it, it just plain isn't clear. At any rate, this should be addressed.

Considering as well that the approximation spanned a time frame of 14.4 seconds as compared to only 3 seconds, it's plain to see that if a rough and ready estimate is desired and lives are not at stake, the Wavelet-Galerkin method is the way to go. However, if the problem calls for the most precise estimation possible, I would have to go with the standard Galerkin Method.

As far as computational time is concerned, the Wavelet-Galerkin method wins hands down. It was much easier to create a program. If one looks at the program, there are simply fewer steps. Finding the U vector in the Galerkin method is work enough. However, if the function is to be interpolated between

nodes using quadratic functions, this becomes a chore very quickly. I used twenty interpolation points between nodes. The computation time shot way up when this happened. However, the pay of was a percent difference so small the naked eye could not detect it on the graph. The plots appear to overlap e*xactly*.

I performed all of these calculations using Matlab 4.2 on my home computer, which has a Pentium Pro 200 processor. Nonetheless, when the interpolation took place, there was no doubt my computer struggled for a few moments. The Wavelet-Galerkin method also made the processor work when forming the solution vector, but it was not for as long.

Another avenue which must be pursued in the Wavelet-Galerkin method is the addition of a forcing function into the original equation. The recent paper by Oyoshi, et al[3] was referred to in this paper because of its new technique for integrating the boundary conditions. However, the idea had only been applied to homogeneous equations. I did attempt to apply it to non-homogeneous equations, but without luck. I did get something that resembled a solution, but it was obvious that further investigation was necessary. For all I know, they may be working on this very problem over in Akita right now!

Finally, natural or Dirichlet conditions should be incorporated. This would expand the usefulness of the method even further. The method will be able to tackle a greater number of problems than is currently possible. I feel that with the work that has been laid out in this paper, another student could take on this challenge in a future paper.

Bibliography

1) Reddy, J.N., An Introduction to the Finite Element Method, McGraw-Hill, Inc., 1976.

2) Burrus, C. S., Gopinath, R. A., and Guo, H., Introduction to Wavelets and Wavelet Transforms, A Primer., Prentice Hall, Inc., 1998.

3) Oyoshi, T., Lu, D., Sibuya, Y., Miura, K., Wavelet-Galerkin Analysis for a Thermally Affected Inhomogeneous Layer., 1997.

4) Amaratunga, K., Williams, J., Qian, S., Weiss, J., Wavelet-Galerkin Solutions for One-Dimensional Partial Differential Equations., Int. J. for Numerical Methods in Eng., vol. 37, pages 2703-2716, 1994.

5) Amaratunga, K., and Williams, J., Wavelet Based Green's Function Approach to 2D PDE's, Engineering Computations, vol. 10, pages 349-367, 1993.

6) Qian, S., and Weiss, J., Wavelets and the Numerical Solution of Partial Differential Equations., Journal of Computational Physics, vol. 106, pages 155-175, 1993.

7) Chen, M. Q., Hwang, C., and Shih, Y. P., The Computation of Wavelet-Galerkin Approximation on a Bounded Interval., Inter. J. Num. Method. Eng., Vol. 39, pages 2921-2944, 1996.

8) Latto, A., Resnikoff, H., and Tenenbaum, E., The Evaluation of Connection Coefficients of Compactly Supported Wavelets (1992), Proc. French – USA workshop on Wavelets and Turbulence, Princeton Univ., June 1991, Springer, NY

9) Strang, G., and Nguyen, T., Wavelets and Filter Banks, Wellesley – Cambridge Press, Inc., 1996.

10) Part-Enander, E., Sjoberg, A., Melin., Isaksson., P., The Matlab Handbook., Addison Wesley Longman, 1996.

11) Nagle, R., and Saff, E., Fundamentals of Differential Equations and Boundary Value Problems., Addison-Wesley Publishing Company, Inc., 1994.

APPENDIX A

```
function p = Psa(h,kk)
% p = psa(h,kk)   calculates samples of the scaling function
% phi(t) = p by kk successive approximations from the
% scaling coefficients h.  Initial iteration is a constant.
% Phi_k(t) is plotted at each iteration.        csb 5/19/93
%
if nargin==1, kk=11; end;          % Default number of iterations
h2= h*2/sum(h);                    % normalize h(n)
K = length(h2)-1; S = 128;         % Sets sample density
p = [ones(1,3*S*K),0]/(3*K);       % Sets initial iteration
P = p(1:K*S);                      % Stores for later plotting
axis([0 K*S+2 -.5 1.4]);
hu = upsam(h2,S);                  % upsample h(n) by S
for iter = 0:kk                    % Successive approximation
  p = dnsample(conv(hu,p));        % convolve and down-sample
  plot(p); pause;                  % plot each iteration
%       P = [P;p(1:K*S)];          % store each iter. for plotting
end
p = p(1:K*S);                      % only the supported part
L = length(p);
x = ([1:L])/(S);
axis([0 3 -.5 1.4]);
plot(x,p);                         % Final plot
title('Scaling Function by Successive Aprox.');
ylabel('Scaling Function Daubechies 6');
xlabel('x');


function y = upsam(x,S)
% y = upsam(x,S) inserts S-1 zeros between each term in the row vector
 x.
%  for example:    [1 0 2 0 3 0] = upsample([1 2 3]).        csb3/1/93
.
L = length(x);
y(:) = [x;zeros(S-1,L)]; y = y.';
y = y(1:S*L-1);



function y = dnsample(x)
% y = dnsample(x) samples x by removing the even terms in x.
% for example:    [1 3] = dnsample([1 2 3 4]).   csb 3/1/93.
L = length(x);
y = x(1:2:L);
```

```
A = [1232 -646 0 0 0 0 0 0 0 0 0
       -646 1096 -646 83 0 0 0 0 0 0 0
       0 -646 1232 -646 0 0 0 0 0 0 0
       0 83 -646 1096 -646 83 0 0 0 0 0
       0 0 0 -646 1232 -646 0 0 0 0 0
       0 0 0 83 -646 1096 -646 83 0 0 0
       0 0 0 0 0 -646 1232 -646 0 0 0
       0 0 0 0 0 83 -646 1096 -646 83 0
       0 0 0 0 0 0 0 -646 1232 -646 0
       0 0 0 0 0 0 0 83 -646 1096 -646
       0 0 0 0 0 0 0 0 0 -646 1232];
A = A/120;

B = [8.075 -1.0375 0 0 0 0 0 0 0 -0.51875 4.0375];
B = B';

X = A\B;

U = zeros(13,1);
       for k = 1:11;
              U(k+1) = X(k);
       end;
U(1,1) = 1.5; U(13,1) = 0.75;


       for k = 1:19;
              US12(k,1) = -(k/40)*(1-k/20)*U(11,1) + (1-(k/20)^2)*U(
12,1) + k/40*(1+k/20)*U(13,1);
       end;
       for k = 1:19;
              US11(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(11,1) + (1-((2
0-k)/20)^2)*U(12,1) - (20-k)/(40)*(1-(20-k)/20)*U(13,1);
       end;
       for k = 1:19;
              US10(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(10,1) + (1-((2
0-k)/20)^2)*U(11,1) - (20-k)/(40)*(1-(20-k)/20)*U(12,1);
       end;
       for k = 1:19;
              US9(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(9,1) + (1-((20-
k)/20)^2)*U(10,1) - (20-k)/(40)*(1-(20-k)/20)*U(11,1);
       end;
       for k = 1:19;
              US8(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(8,1) + (1-((20-
k)/20)^2)*U(9,1) - (20-k)/(40)*(1-(20-k)/20)*U(10,1);
       end;
       for k = 1:19;
              US7(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(7,1) + (1-((20-
k)/20)^2)*U(8,1) - (20-k)/(40)*(1-(20-k)/20)*U(9,1);
```

```
        end;
        for k = 1:19;
                US6(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(6,1) + (1-((20-
k)/20)^2)*U(7,1) - (20-k)/(40)*(1-(20-k)/20)*U(8,1);
        end;
        for k = 1:19;
                US5(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(5,1) + (1-((20-
k)/20)^2)*U(6,1) - (20-k)/(40)*(1-(20-k)/20)*U(7,1);
        end;
        for k = 1:19;
                US4(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(4,1) + (1-((20-
k)/20)^2)*U(5,1) - (20-k)/(40)*(1-(20-k)/20)*U(6,1);

        end;
        for k = 1:19;
                US3(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(3,1) + (1-((20-
k)/20)^2)*U(4,1) - (20-k)/(40)*(1-(20-k)/20)*U(5,1);
        end;
        for k = 1:19;
                US2(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(2,1) + (1-((20-
k)/20)^2)*U(3,1) - (20-k)/(40)*(1-(20-k)/20)*U(4,1);
        end;
        for k = 1:19;
                US1(k,1) = ((20-k)/40)*(1+(20-k)/20)*U(1,1) + (1-((20-
k)/20)^2)*U(2,1) - (20-k)/(40)*(1-(20-k)/20)*U(3,1);
        end;
 USA = [U(1,1),US1',U(2,1),US2',U(3,1),US3',U(4,1),US4',U(5,1),US5',U(
6,1),US6',U(7,1),US7',U(8,1),US8',U(9,1),US9',U(10,1),US10',U(11,1),US
11',U(12,1),US12',U(13,1)];


 fus = (1:1/80:4);


 plot(fus,USA)
 title('Galerkin Approximation, Quadratic'); xlabel('X'); ylabel('U(X)
');
```

```
t = [-5.26785714285743+0.09375
3.39047619047638
-0.87619047638
0.11428571428571
0.00535714285714
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0.00535714285714
0.11428571428571
-0.87619047619052
3.39047619047638];

%       This is the convolution kernel which had to be entered manuall
y.
```

```
D = zeros(68,1);          % creates D matrix
D(1,1) = 1.5;
D(68,1) = 0.75;

Cs = (fft(D))./(fft(t));          %  shows component by component divisi
on
Cs = ifft(Cs);                    %  solves for Cs solution matrix
Cs = real(Cs);                    %  ensures that Cs matrix is real
```

```
        for k = 1:68;
                u(k,:) = Cs(k)*phi;
        end;


B = zeros(68,9216);

        for k = 1:68;
                a = u(k,:);
                a = [zeros(1,(k-1)*128),a];
                [row col] = size(a);
                B(k,(1:col)) = a;
        end

y = sum(B);

        for k = 1:9216;
                ya(1,k) = y(1,9216-(k-1));
        end;

plot(ya)
```