Matrix Multiplication, Inversion and Partial Differential Equations via Wavelets

Daniel Beatty

October 6, 2003

Contents

Chapter 1

Introduction

One overwhelming question drives computational science, how fast can the answer be computed? Of course, the answer has to answer its own question. In this thesis, the questions are for three computational areas of mathematics. One is how fast can matrix multiplication be computed? Another is how fast can a matrix be inverted? Third is how quickly can a partial differential equation be solved? Of course there are already conventional algorithms to compute them, so where does this thesis come in?

All of these problems have one thing in common. Each of these problems in the conventional methods are mostly solved via a matrix. Obviously matrix multiplication and matrix inversion are classic matrix operations that obey the rules of linear algebra. Partial differential equations have several varieties of methods which obtain solutions via a solution matrix.

The wavelet transform is a linear orthonormal operation. The three mathematical operations are solved by linear algebra type methods. What is important about wavelets are the two most desired

qualities in computation of matrices and systems of equations, sparseness and condition number. Sparseness applied to matrix states a good majority of the elements in such a matrix are zero. Condition number determines how quickly a matrix solution will converge. The two are important due to the number one question in computational science as applied these mathematical problems, how fast can wavelet linear algebra be performed.

In this thesis, an overview is provided to describe to the reader what wavelets are and their generic applications. This overview covers a little image processing since it is the easiest means to show the qualities of a two-dimensional wavelet transform. The three chapters that follow are devoted to matrix multiplication, matrix inversion and solutions to partial differential equations in respective order. Those three chapters also discuss how wavelets can be used to solve those problems. Furthermore, this these is dedicated to show how efficient wavelets are at solving these mathematical problems.

Chapter 2

Overview on Wavelets

Goals for this introductory document on wavelets is to define a wavelet, acknowledge alternative definitions of a wavelet, specify basic wavelet theorems, and show wavelets through practical examples. Convolution provides the key for defining the wavelet, its theorems and practical examples. The Haar Wavelet Filter is used to show these concepts.

2.1 Wavelet Transform Definitions:

As stated, the wavelet transform can be viewed from both a matrix representation[1] and vector convolution. Such a representation provides means to use many linear algebra and abstract algebra methods to prove theorems about wavelets. Another form is a convolution type of operation. Convolution wavelet transform methods provide a simple computational method in general. Special case convolution methods also exist, which allow even simpler algorithms to perform wavelet transforms.

In the matrix form, a wavelet matrix is defined as a generalization of a square orthogonal or unitary matrix which is a subset of a larger class of rectangular matrices . This matrix harvest information information for defining a wavelet system. Such matrices are defined in terms of their rank and genus[1].

It was Alfred Haar himself who defined an orthonormal wavelet matrix in "Zur Theorie der orthogonalen Funktionensystem Math Annual" written in 1903. The said wavelet was named in honor of Alfred Haar, the Haar Wavelet Matrix. A Haar Wavelet Matrix has a genus of one. Such matrices have a one to one mapping to a general wavelet matrix. Also, Haar Wavelet Matrices are equivalent to the characteristic Haar Matrix. Lastly, a rank 2 Haar Matrix sufficiently shows to show all geometric considerations[1].

2.2 Wavelets via Convolution

Wavelets are defined in terms of average and difference components. Each component can further be transformed to isolate properties of each component. Typically, each component has the form $O \rightarrow (A|D)$ where O is the original signal, A is the average component, D is the difference component and $(A|D)$ the signal A concatenated with D. Each of these components are produced by an orthonormal basis. Also, these components are produced such that the original can be constructed easily from them.

Many mathematicians such as Walker[2], use a form that eliminates half of the values. Thus a form can be defined which has the same number of elements as the original. The rules for choosing the

member elements are dependent on the wavelet filter choice.

Another useful property of wavelets via convolution is the simplicity of the operation. The general case works for all. Such an algorithm requires one nested loop as seen below:

$\forall i \in [0, M)$

    $\forall j \in [0, N)$

        n=i-j

        if $(n \in [0, M))$

            $y_i += x_n \cdot h_j$

This filter simply equates to the mathematical function: $x * h = \sum_l x_{k-l} h_k$, which is the convolution operation. As we can see the operation is slightly less than $O(n^2)$. For practical use, the filter is made smaller than the actual signal being analyized. In some cases, the filter may be much smalled than the signal. Filter size matters in extracting features from the original signal.

In the case of this convolution operator, the limit is acutally M, not N. The value of M is the size of the original signal. Since the resulting vector is the same as the original, the vector is said to be fully qualified. Only half of those values are necessary to reconstruct the original (every other element).

To perform a wavelet transform via convolution, each signal is convolved twice.

$A_i = A_{i-1} * V$

$D_i = A_{i-1} * W$

where

V is the scaling wavelet vector

W is the differencing wavelet vector

A is the average vector (scaled vector)

D is the difference component vector

$\forall i \in [1, L)$ and $A_0 = f$ which is the original signal

L is the limit on the number resolutions that signal can have based on the wavelet type.

2.3 Class 2-D Wavelet: Complete Form

The convolution version can be used to derive a Wavelet Matrix. For a general case, it is simpler to use the convolution method. The matrix form becomes practical in repetitive special case applications.

The 2-D transform has four components: the average, vertical, horizontal, and diagonal. Two general computational means exist to generate a one-resolution transform. These can derive means for performing many resolutions.

A complete transform method returns a result matrix which is the same size as the source matrix.

The result contains the four components. Each component resides on 4 corners of the matrix. Given a matrix B, the transform is to yield the following form:

$$B \Rightarrow \begin{matrix} H & D \\ A & V \end{matrix}$$

where A is the average component, H is the horizontal component, V is the vertical component, and D is the difference component. There is another form which is also used as an example:

$$B \Rightarrow \begin{matrix} A & V \\ H & D \end{matrix}$$

The first version is simple in concept, but provides a few more possibilities for error and confusion. Regardless of the case, the four components have the following definitions:

1. Average component: produced by filtering the row vectors and the column vectors with the averaging filter.
2. Vertical Component: produced by applying the average filter to the column vectors and the difference filter to the row vectors.
3. Horizontal component: produced by applying the average filter to the row vectors and the difference to the column vectors.
4. Diagonal component: produced by applying the difference filter to both the row and column vectors.

2.3.1 Proof of Concept

Two methods of convolving a matrix are easily conceived. First is to use 1-D wavelet. The other is to apply the convolution scheme straight to the matrix.

Included in the wavelet experiment are both. Realistically, both can and do achieve the same result. However, the direct method achieves speed advantages by the lack of overhead. The direct method leaves a temporary vector resident in memory. Also, there are 2 fewer transfers per row and columns.

1-D to 2-D Method

Both rows 1-D and 2-D and columns 1-D and 2-D transform are performed similarly. The obvious difference is the indexing of rows and columns.

Given 1-D wavelet transform

source matrix

Algorithm: (Row Transform)

$\forall i \in rows$

- $\forall j \in columns$
- $S[j] \leftarrow source[i][j]$
- $S \Rightarrow^{W} R$

This principle of this algorithm is simple. Only three intuitive steps are necessary per row or column. Two of these steps are array transfers (row/column transfer to an array). These arrays are fed into the 1-D transforms.

However, the 1-D wavelet transform itself includes a series of memory allocation and deallocation operations. Each memory call is at the minimum a system call.

Vector - Matrix Method

The principle of this algorithm is more complicated. All functionality, such as convolution, is built into the method. There are fewer calls and passing of structures to external functions to compute the transform.

This method has a few givens. The source matrix, the Haar average filter, and Haar difference filter are given arguments. The result argument is the return argument. The transform signals sub-functions row transform and column transform to perform the work.

The algorithm is as follows for the row tranform (and is similar for the column transform):

$\forall i \in rows$

1. Initialize temporary array/vector to all zeros (x).
2. $\forall k \in columns$
   (a) $\forall l \in ha.Size$ x += $\begin{cases} S_{i,k-l} * hA_l & k-l \in columns \\ 0 & otherwise \end{cases}$
3. $\forall k \in columns/2$

$result_{i,k} = x_{2k+1}$ (In other words, odd split)

4. Initialize x to all zeros.

5. $\forall k \in columns$

   (a) $\forall l \in hd.Size$ x += $\begin{cases} S_{i,k-l} * hD_l & k-l \in columns \\ 0 & otherwise \end{cases}$

6. $\forall k \in columns/2$

   $result_{i,k+columns/2} = x_{2k+1}$ (In other words, odd split)

2.3.2 Multi-Resolution

The multi-resolution wavelet transform and the inverse multi-resolution transform resemble the vector-matrix version. All functionality is built into this method. However, there are structural changes.

The wavelet transform (multiresolution) uses private members of the class (hA, hD, xD/yD, xA/yA). Both Haar filters are maintained this way. Also both row and column transforms have average and difference myVector classes for temporary storage. All of these members are allocated and destroyed by the wavelet transform method itself. The simplified algorithm of the row transform is the following:

1. Initialize xA and xD to zero

2. $\forall k \in columns \forall l \in filter$

   - n = k - l

- if ($n \in columns$)

  $xA_k = W_{i,n} * hA_l$

  $xD_k = W_{i,n} * hD_l$

3. Transfer back to W

   $W_i = xA|xD$

1. initialize yA and yD to zero

2. $\forall k \in rows \forall l \in filter$

   - n = k - l
   - if ($n \in columns$)

     $yA_k = W_{i,n} * hA_l$

     $yD_k = W_{i,n} * hD_l$

3. Transfer back to W

   $W_j = yA|yD$

Note: W_i names the row vectors and W_j names the column vectors, and $W_{i,j}$ is the element from the ith row and jth column.

2.3.3 Computational Cost:

The cost of this algorithm is computed first for each row and each column. This value is used to compute the cost of the matrix. The cost of computing the matrix is used to compute the

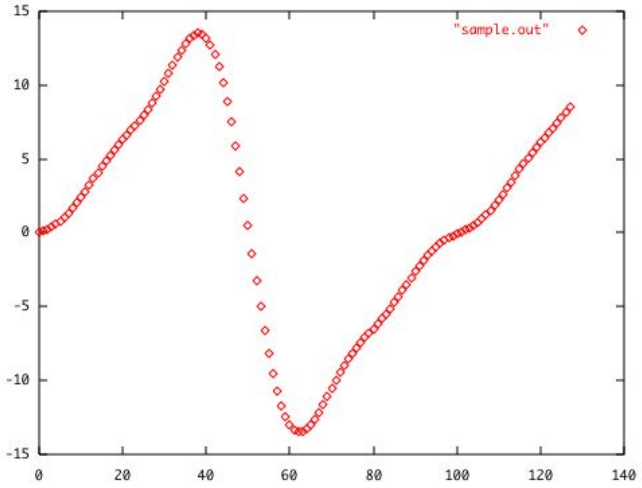

Figure 2.1: Sample function. The x-axis is the array index (index n). The y value is simple – the value y[n].

cost of the multi-resolution steps. Per row the cost is 3k, where k is the number of columns. Per column the cost is 3l, where l is the number of rows. For the whole matrix, one resolution costs 6kl operations to compute the wavelet transform. Per resolution, the rows and columns shrink by 2^i for each resolution, i, performed. The limit of this cost equals $12kl$ operations. Thus, the cost is linear.

2.4 Results - 1-D Wavelet Transform

Testing of the 1-D wavelet was performed on a sinusoidal wave form of 128 elements. The given function has the equation shown in Figure 2.1.

$$y[n] = 10\sin(\tfrac{n}{128}) - 5\sin(\tfrac{n}{64}) + 2\sin(\tfrac{3n}{128}) - \sin(\tfrac{n}{32})$$

The first version of the 1-D transform used the even elements of both convolutions to generate the

wavelet transform. These even elements came from the over-complete form and naturally allow the potential to have complete information. However, in doing so, a fundamental flaw appears.

In order to evaluate the effectiveness of the wavelet transform three tests have been devised. First, energy equivalence is used to determine how much energy is retained in the transform from the original. The general shape is used on a the first resolution to test if the average signal has the same general shape as the original. Lastly, the inverse transform is used to recover the original signal. A comparison is made between the original and the recovered signal.

After one resolution, the transformed signal has the same energy as the original. This is good since it allows the original to be recovered from the transform. Also, the average component of the transform has the same shape as the original. This is good. However, the recovered signal is missing the last element. Refer to Figure 2.2

The secret is in which elements are used from the over-complete to make the complete. The over-complete in this project comes from the average component and difference which are simply the result of convolution.

The convolution means is at the heart of the issue. The convolution operator in this case starts with the first element of the filter against the first element of the signal. In the simple Haar Wavelet case, there is a transformation pairing. $(S_i, S_{i-1}) \rightarrow A_i$ and $(S_i, S_{i-1}) \rightarrow D_i$

In this pairing with zero indexed signals, the odd indexed elements from the over-complete must be used to have all elements of the original accounted for.

Also this produces a functional difference between wavelet inverse transform for odd and even versions. The difference is slight; however, the last element is lost in the even indexed form.

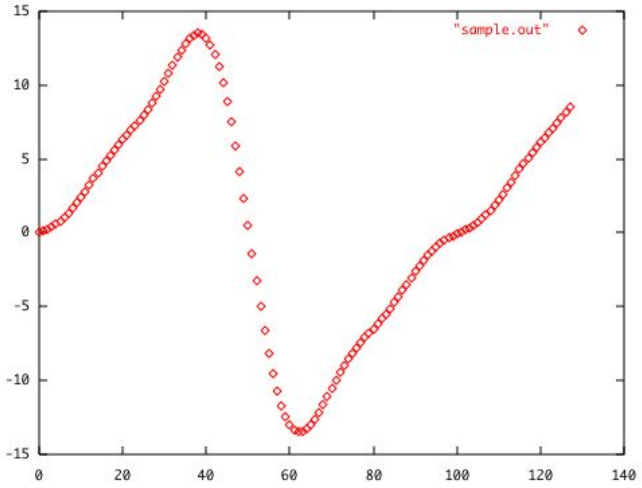

Figure 2.2: Recovered function. The x-axis is the array index (index n). The y value is simply the value y[n]. The function was recovered from an even indexed wavelet transform.

- Odd $R_{2i} = (A_i - D_i)\sqrt{1/2}$ $R_{2i+1} = (A_i + D_i)\sqrt{1/2}$
- Even $R_{2i} = (A_i + D_i)\sqrt{1/2}$ $R_{2i-1} = (A_i - D_i)\sqrt{1/2}$

An odd indexed wavelet transform yields the same energy. However, all of the values are accounted for. Refer to Figure 2.3.

2.5 Results: 2-D Wavelet Transform

A simple room picture shows the difference correct indexing produces in the wavelet transform and its inverse. The 1-D to 2-D method shows the incorrectly indexed case. A correctly indexed version is shown in the vector-matrix method.

The 1-D to 2-D method has a serious issue with memory leak errors (Macintosh OSX, using gcc 3.1). Memory is allocated and deallocated quickly, and on some platforms shows up as an error.

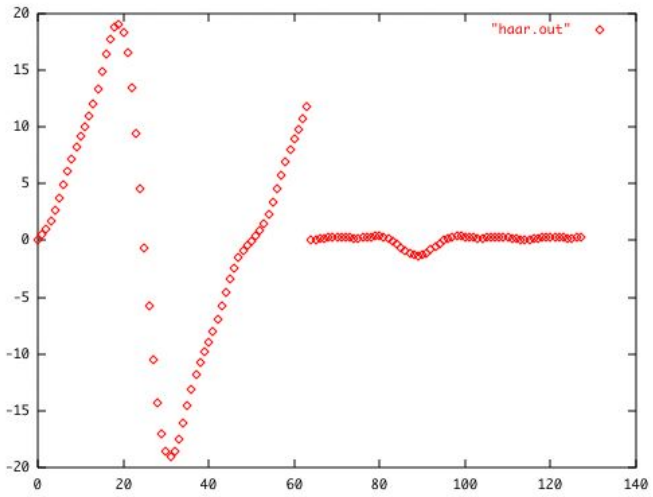

Figure 2.3: Recovered function. The x-axis is the array index (index n). The y value is simply the value y[n]. The function was recovered from an odd indexed wavelet transform.

On other platforms, the result is degraded performance (IRIX, SGI Octane2 using gcc 2.9). An example image of 720 x 486 requires nearly 10 minutes to compute the wavelet transform by this method on an SGI Octane2. However, this method does eventually return a correct result.

The matrix-vector method also yields the correct result. However, there is less memory overhead in this method as compared to the 1-D to 2-D method. As a result, both the row wavelet transform and column wavelet transforms are performed more quickly, with fewer memory transfers and allocations. Obviously, this also allows for the operation to be conducted almost entirely in cache memory on both the SGI Octane2 and Macintosh G4 based machines. A Macintosh G3 based machine still requires main memory at a minimum to execute the same operation which yields a slower performance.

A correct result must also be matched to a correct inverse method. The indexing order matters. The inverse transform method is a forward inverse transform method. In the case of 1-D to 2-D

Figure 2.4: Original Image. This image is the original image.

transform, the ordering was reverse indexed (Figure 2.6). As a result of an error in indexing, ringing is seen on edges in this method(Figure 2.8) for a case in point. Caution is incredibly important when matching both forward and reverse indexing, since matching the mathematics to the actual ordering can be obscure and tricky.

A correct result is shown in Figure **??**. In this case, the indexing was matched up and ringing is not present. It is clear that the recovered image and the original (Figure 2.4) are nearly indistinguishable.

2.5.1 Multi -resolution

The expected result is a picture within a picture. Each average component has a further transform on it. The three resolution transform has the form:

Figure 2.5: Recovered Image. This image is the recovered image. Depending on whether the image was saved as a picture first can affect the white spots in the picture. Ringing is also an issue.

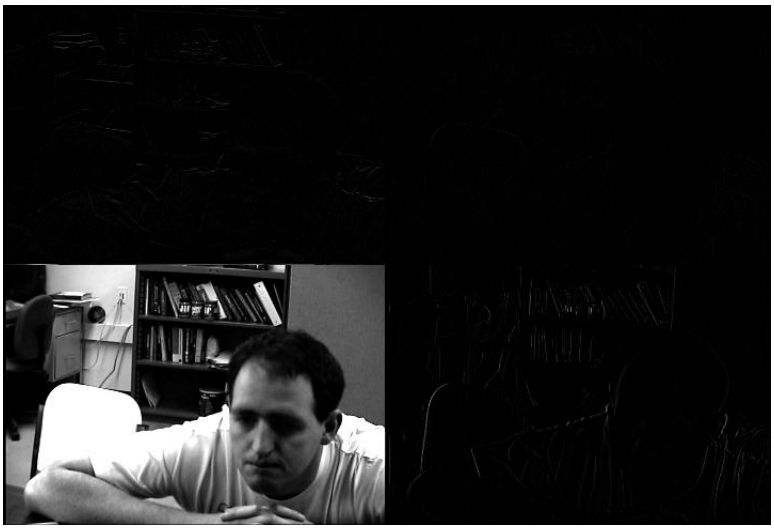

Figure 2.6: Wavelet Transform Image. This image is divided in to average, horizontal, vertical and diagonal components.

Figure 2.7: Wavelet Transform Image. This image is divided in to average, horizontal, vertical and diagonal components, using the vector-matrix version.

Figure 2.8: Recovered Image (Vector-Matrix Method). This image is the recovered image. This version avoids the ringing by using the vector-matrix version which is more aligned for the inverse wavelet transform.

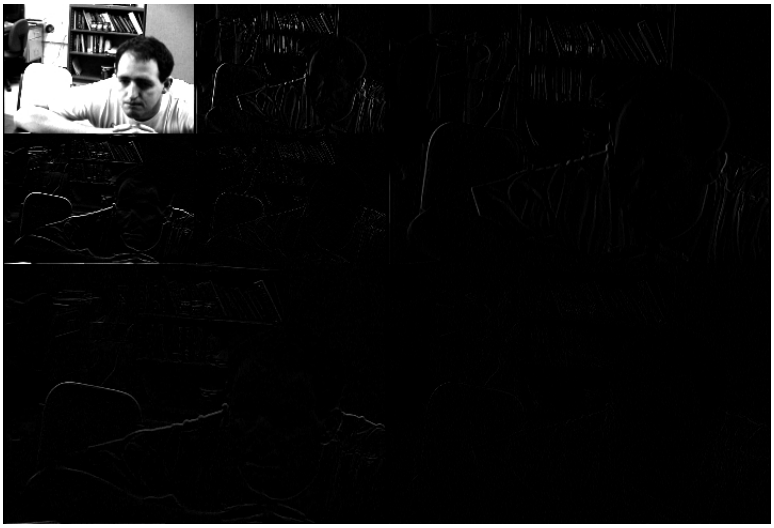

Figure 2.9: Wavelet Transform Image. This image is divided in to average, horizontal, vertical and diagonal components using multi-resolution wavelet transform. Note the the average component was transformed one step further.

$$W_3 = \begin{array}{cccc} A_3 & V_3 & & \\ & & V_2 & \\ H_3 & D_3 & & V_1 \\ H_2 & & D_2 & \\ H_1 & & & D_1 \end{array}$$

Refer to Figure 2.9 for the image transform results.

To obtain the inverse, an exact reverse procedure is necessary, otherwise the distortion is hideous. The first attempt of the wavelet inverse transform was out of order, refer to Figure 2.10 . A correct picture was obtained during the second attempt. Correct order yielded correct results, refer to Figure 2.8 .

Figure 2.10: Recovered Image - Wrong Order (Multi-Resolution). This image shows a 2-D wavelet transform after it was recovered out of order. Obviously, the distortion is hideous.

2.5.2 Threshold FIltering

After a triple resolution, a 0.02 threshold will eliminate 81.1706 percent of elements in the original sample picture. Also at this point, the effects of removing these elements becomes visually evident (Figure 2.11). At a 0.01 threshold, 66.0205 percent of the elements are removed. Visually, the recovered sample and the original appear to be the same (Figure 2.14). At a threshold of 0.1, 92.9987 percent of the elements are reduced to zero. However, the distortions are clearly visible at this level of thresholding (Figure 2.12). Even at a threshold of 0.001 which is below the numerical precision of the original, 16.0814 elements are reduced to zero. At a threshold of 0.002, 28.9683 percent is removed.

Consequently after a triple resolution, nearly 29 % of the data was irrelevant for the image's brightness resolution (which also applies to color). 60 % to 85 % of was not noticeable to human perception. Which leaves only 15 % to 40 % of the data actually contributing or being necessary to reconstruct the image.

Figure 2.11: Recovered Image - 2% threshold (Multi-Resolution). This image had nearly 83% of its elements removed in the triple resolution wavelet transform.

Figure 2.12: Recovered Image - 10% threshold (Multi-Resolution). This image had nearly 93% of its elements removed in the triple resolution wavelet transform.

Figure 2.13: Recovered Image - 5% threshold (Multi-Resolution). This image had nearly 85% of its elements removed in the triple resolution wavelet transform.

Figure 2.14: Recovered Image - 1% threshold (Multi-Resolution). This image had nearly 60% of its elements removed in the triple resolution wavelet transform.

Bibliography

[1] Howard L. Resnikoff. and Raymond O. Wells, Jr. *Wavelet Analysis: The Scalable Structure of Information* copyright Springer-Verlag New York, Inc. New York, NY 10010, USA, 1998

[2] James Walker *A Primer on Wavelets and Their Scientific Applications* copyright Chapman & Hall/CRC : Boca Raton, FL, USA, 1999

[3] Gavin Tabor *Wavelets - The Idiots Guide* http://monet.me.ic.ac.uk/people/gavin/java/waveletDemos.html

[4] Amara Graps *Introduction to Wavelets* http://www.amara.com/current/wavelet.html

Chapter 3

Matrix Multiplication via Wavelets

3.1 Wavelet Matrix Multiplication

One of the key points for wavelet matrix multiplication is the proof that $W(A) \times W(B) = W(A \times B)$ If this is the case, then it is obvious that $W(A) \times W(B) = W(C) = W(A \times B = C)$. So far the proof is still weak. The reason is that an example proof is useful for proving something to not be the case, rather than being the case. However, a simple example does show some intuitive steps that would be necessary for a proof.

3.1.1 A 2×2 example

Conventional Multiplication

Conventional multiplication is spelled out as

$$c_{i,j} = \sum_k a_{i,k} b_{k,j}$$

For a 2×2 matrix, there is the following:

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{pmatrix}$$

Wavelet Transform of two $2\times$ 2 matrices

For a wavelet transform on both matrix A and B, the results are:

$$W(A) = \frac{1}{2} \begin{pmatrix} a_{1,1} + a_{2,1} + a_{1,2} + a_{2,2} & a_{1,1} + a_{2,1} - a_{1,2} - a_{2,2} \\ a_{1,1} - a_{2,1} + a_{1,2} - a_{2,2} & a_{1,1} - a_{2,1} - a_{1,2} + a_{2,2} \end{pmatrix}$$

$$W(B) = \frac{1}{2} \begin{pmatrix} b_{1,1} + b_{2,1} + b_{1,2} + b_{2,2} & b_{1,1} + b_{2,1} - b_{1,2} - b_{2,2} \\ b_{1,1} - b_{2,1} + b_{1,2} - b_{2,2} & b_{1,1} - b_{2,1} - b_{1,2} + b_{2,2} \end{pmatrix}$$

Product of A and B in wavelet space

The conventional product of A and B can be transformed into wavelet space. The results of this matrix transform is as follows:

$$W(A\times B) = W \begin{pmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} (a_{1,1}b_{1,1} + a_{1,2}b_{2,1} + a_{1,1}b_{1,2} + a_{1,2}b_{2,2}) + (a_{2,1}b \\ (a_{1,1}b_{1,1} + a_{1,2}b_{2,1} + a_{1,1}b_{1,2} + a_{1,2}b_{2,2}) - (a_{2,1}b \end{pmatrix}$$

$$W(A\times B) == \frac{1}{2}\begin{pmatrix} (a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,1}b_{1,2}+a_{1,2}b_{2,2}+a_{2,1}b_{1,1}+a_{2,2}b_{2,1}+a_{2,1}b_{1,2}+a_{2,2}b_{2,2}) & (a_{1,1}b_{1,1}+a_{1,2}b \\ (a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,1}b_{1,2}+a_{1,2}b_{2,2}-a_{2,1}b_{1,1}-a_{2,2}b_{2,1}-a_{2,1}b_{1,2}-a_{2,2}b_{2,2}) & (a_{1,1}b_{1,1}+a_{1,2}b \end{pmatrix}$$

What is $W(A) \times W(B)$

Straight forward multiplication of $W(A) \times W(B)$ works out as follows:

$$W(A)\times W(B) = \frac{1}{2}\begin{pmatrix} a_{1,1}+a_{2,1}+a_{1,2}+a_{2,2} & a_{1,1}+a_{2,1}-a_{1,2}-a_{2,2} \\ a_{1,1}-a_{2,1}+a_{1,2}-a_{2,2} & a_{1,1}-a_{2,1}-a_{1,2}+a_{2,2} \end{pmatrix} \times \frac{1}{2}\begin{pmatrix} b_{1,1}+b_{2,1}+b_{1,2}+b_{2,2} & b_{1,1}+b_2 \\ b_{1,1}-b_{2,1}+b_{1,2}-b_{2,2} & b_{1,1}-b_2 \end{pmatrix}$$

Of course this is better simplified.

$$\frac{1}{2}\begin{pmatrix} (a_{1,1}b_{1,1}+a_{2,1}b_{1,1}+a_{1,2}b_{2,1}+a_{2,2}b_{2,1}+a_{1,1}b_{1,2}a_{2,1}b_{1,2}+a_{1,2}b_{2,2}+a_{2,2}b_{2,2}) & (a_{1,1}b_{1,1}+a_{2,1}b_{1,1}+a_{1,2}b_{2,1}+ \\ (a_{1,1}b_{1,1}-a_{2,1}b_{1,1}+a_{1,2}b_{2,1}-a_{2,2}b_{2,1}+a_{1,1}b_{1,2}-a_{2,1}b_{1,2}+a_{1,2}b_{2,2}-a_{2,2}b_{2,2}) & (a_{1,1}b_{1,1}-a_{2,1}b_{1,1}+a_{1,2}b_{2,1}- \end{pmatrix}$$

Compared with $W(C)$ computed the conventional way:

$$W(C) = \frac{1}{2}\begin{pmatrix} (a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,1}b_{1,2}+a_{1,2}b_{2,2}+a_{2,1}b_{1,1}+a_{2,2}b_{2,1}+a_{2,1}b_{1,2}+a_{2,2}b_{2,2}) & (a_{1,1}b_{1,1}+a_{1,2}b_{2,1}- \\ (a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,1}b_{1,2}+a_{1,2}b_{2,2}-a_{2,1}b_{1,1}-a_{2,2}b_{2,1}-a_{2,1}b_{1,2}-a_{2,2}b_{2,2}) & (a_{1,1}b_{1,1}+a_{1,2}b_{2,1}- \end{pmatrix}$$

Notice that $W(A) \times W(B) = W(A \times B)$, in the case of 2×2 matrices.

3.1.2 Proof of Wavelet Matrix Multiplication

A reminder from Dr. Sinzinger, wavelets and their inverses are linear operators. As he correctly reminded me any linear operator has the following properties:

1. $L(AB) = L(A)\dot{L}(B)$
2. ψ^{-1} is a linear operator
3. $\psi^{-1}(\psi(A)\dot{\psi}(B)) = \psi^{-1}(\psi(A))\dot{\psi}^{-1}(\psi(B))$
4. $\psi^{-1}(\psi(A)) = A$
5. $\psi^{-1}(\psi(B)) = B$
6. Therefore: $AB = \psi^{-1}(\psi(A)\dot{\psi}(B))$

Thus this is sufficient proof that wavelet matrix multiplication is sound.

3.2 Chain Multiplication Structure

Recall the chain structure. This structure can be referenced from most introductory algorithm books. The big idea is to be able to locate specific items quickly by some key which reduces the search to a family of keys that close together by a hashing scheme. In the case of matrix multiplication, no hash key is needed. The row or column identifier is sufficient for this. Of course a description of this scheme is in order.

Each chain is organized as a one-dimensional array. Each array has an array list, called a link, which contains the actual data, and array list count showing how many items are in each array. A

next and previous value in this case would be superfluous since columns and rows are arranges in lexicographical order. The array list contains the following items:

- key
- item value (row value/ column value)
- previous key
- next key

In the case of a column chain, the keys are row identifiers. In the case of a row chain, the keys are column identifier. This structure is similar to a matrix and can be represented by a matrix. The essential thing for the chain is the arrangement of the array lists. Zero values are not allowed, and thus order and index keys must be maintained.

What has this to do with sparse matrix multiplication? The left matrix is transformed into a row chain and the right matrix is transformed into a column chain. Each value of the result matrix is simply a multiplication of the row vectors in the left matrix by the column vectors in the right matrix. With the chain structure, it simply row links times the column links. Only elements with matching keys are allowed to be multiplied together. The rest are assumed to be zero, and thus no action is taken. The sum of the multiplies is the result. The multiplication procedure is as follows:

1. Chain Multiply
   (a) Arguments: left matrix chain (A) and right matrix chain (B)
   (b) Results: Result matrix
   - $\forall i \in R.row$

- $\forall j \in R.col$
  - $R_{i,j} = CM(A[i], B[j])$
  - —- Note that CM is the chain multiply procedure.

2. Chain Multiply Element

   (a) Arguments: row link (r) and column link (c)

   (b) Output: Double result: total

   - rlimit = r.size;
   - climit = c.size;
   - k = l = 0;
   - jlow = 0;
   - total = 0.0
   - BnotExhausted = true
   - while both (k ¡ rlimit) and (BnotExhausted)
     - $\forall l \in [jlow, climit)$ if ($A_i^c.getkey(k) \equiv B_j^c getkey(l)$) then lmatch = l
     - if ($l \equiv climit$) BnotExhausted = false
     - otherwise
       - temp += $A_i^c[k] * B_i^c[lmatch]$
       - jlow = lmatch

For the Matrix Chain Multiply the complexity is $O(N^2)$. For the chain multiply, the complexity is $O(M)$ where M is the larger length of the two links. Thus total complexity is $O(N^2M)$ since M's largest size is N. This a general and simple algorithm for sparse matrix multiplication of matrix

chains. Of course the chore of loading the matrix chains is $O(N^2)$ per matrix. Thus total cost is on the order of $O(3N^2M)$. So long as M is significantly less than N (on the order of a $\frac{1}{3}$), then the wavelet matrix multiplication has a reasonable advantage.

3.3 Practical Implementation

There are few questions about the practical implementation that must be answered:

1. Are the matrices square matrices, and same size?
2. If not, are the dimensions suitable for multiplication?
3. If so, what is the maximum resolution for each matrix? The lesser maximum is the limit for both.
4. Is the wavelet transform performed on each matrix the same?

A yes answer to the first matrix, allows for an relatively easy transform, and matrix multiply. If the matrices are not square, then practical problems exist. If the wavelet transform is the same on each, then the linear proof is sound. However, if they are not then the proof is bogus and the multiplication is too.

The general wavelet based matrix multiply is as follows:

1. Arguments:
   - A : a $m \times p$ matrix
   - B: a $p \times n$ matrix

2. Results: C : a $m \times n$ matrix

3. Procedure:

   - $A \xrightarrow{\psi} \alpha$
   - $B \xrightarrow{\psi} \beta$
   - $\alpha \overset{ChainRow}{\longrightarrow} \alpha^c$
   - $\beta \overset{ChainColumn}{\longrightarrow} \beta^c$
   - Chain Multiply $(\alpha^c, \beta^c) \rightarrow C$

3.3.1 Chain Row and Chain Column Setup

Setting up the the chain-link structure for row or column orientation is relatively simple. Each one requires proper addressing of the hooks. The procedures are as follows:

Row Chain-Link Setup

- $\forall i \in \alpha.rows$
  - $\forall j \in \alpha.columns$
    - if ($\alpha[i][j] \approx 0$) $a^c.hook_i.addlink(j, \alpha_{i,j})$

Column Chain Link Setup

- $\forall i \in \alpha.rows$
  - $\forall j \in \alpha.columns$

$*$ if ($\alpha[i][j] \approx 0$) $a^c.hook_i.addlink(j, \alpha_{i,j})$

One of the big questions is how to optimize wavelet packets for multiplication applications. A straight multi-resolution wavelet on a square matrix is seemingly trivial. In the square matrix, straight multi-resolution case the number of resolutions is dictate by the size of the matrices being multiplied. For wavelet packets on square matrices, it is also a size issue. One can apply wavelet packets to maximum extent that the matrix size allows. However, some of the packet levels may be unnecessary since the amount energy shuffling is small in comparison to the number of operations. In the case of non-square matrices, best fits must be applied to ensure that the same wavelet transform

The Chain-Link Structure

The chain link structure is derived from 2 other classes (hooks and links). Each one of these classes are relatively simple and can be implemented with either arrays or pointers. Due to the anticipated size, the array mechanism is chosen for speed and efficiency.

Class Chain Link members

- length
- hooks

Class Hook members:

- length
- l2norm

- links

Class Link members:

- id
- value

Recursive Wavelet Packets The algorithm for optimal wavelet packets is easiest to describe by recursive means. The stop condition either when the maximum resolution has been reached, or energy shuffle metric is satisfied.

procedure wavepack (matrix A, integer res) return μ

- if ($res \equiv 0$) stop
- if ($A.rowmodulo2 \equiv 0$ and $A.colmodulo2 \equiv 0$) stop
- if (optimum) stop
- otherwise
  - $A \stackrel{\psi}{\rightarrow} \mu$
  - $\mu \stackrel{topleft}{\rightarrow} \alpha$
  - $\mu \stackrel{lowleft}{\rightarrow} \beta$
  - $\mu \stackrel{topright}{\rightarrow} \gamma$
  - $\mu \stackrel{lowright}{\rightarrow} \delta$
  - wavepack (α , res -1)

- wavepack (β , res -1)
- wavepack (γ , res -1)
- wavepack (δ , res -1)
- insert ($\alpha \overset{topleft}{\rightarrow} \mu$)
- insert ($\gamma \overset{topright}{\rightarrow} \mu$)
- insert ($\beta \overset{lowleft}{\rightarrow} \mu$)
- insert ($\delta \overset{lowright}{\rightarrow} \mu$)
- stop

The determining of the optimum condition is matter that is not well defined.

Chapter 4

PDE via Wavelets

Wavelets are used to condition the difference equation matrix, and yield a sparser matrix. What this means is that a wavelet enhanced matrix is easier to solve. How is this accomplished? The condition number is an indication of how quickly a solution will converge. Sparseness refers to the large concentration of zeroes in the matrix. Consequently, a sparse matrix has fewer elements to be used in a computation. The point of this chapter is show how wavelets can be applied to yield a well conditioned and sparse matrix, how conventional methods of solving PDEs work, why a wavelet transformed matrix works for the solution. Also included is a comparison between conventional and wavelet based methods.

4.1 PDE in General

What are partial differential equations (PDE) good for? PDE are used to solve equilibrium, diffusion states, oscillitory systems. Most conventional algorithms are costly for performing these tasks.

Therefore the challenge is how to compute the solution of PDE faster.

4.1.1 Classic Methods

In the classic sense computational scientists agree that there are three basic categories of partial differential equations: elliptical, parabolic, and hyperbolic. Some scientist consider the hyperbolic to be made of a fourth category, ultra-hyerpbolic. Similar properties exist for these groups of problems. Properties include their boundary conditions and the physical phenomena the PDE models.

Elliptical PDE represent heat expansion, electrostatic charge, and other source expansion problems. Parabolic PDE model diffusion of gases or fluids. Oscillating states such as electromagnetic fields, some control theory, vibrating strings, and electronic communication systems all map to the model of hyperbolic PDE.

The general form of difference in partial differential equations is defined

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G$$

For case where $B^2 - 4AC > 0$, then the PDE is hyperbolic, and the solution has an oscillatory nature to it. In cases, where $B^2 - 4AC < 0$, then the PDE is elliptical. If there is an equivalence to zero, then the PDE is parabolic. The effect this has on classic methods (analytical and numerical) is how the PDE is represented and eventually solved.

In addition to the above classifications, their limits or boundaries also have a classification. Boundary conditions are defined either on the dependent variable or the gradient of the dependent variable.

If the boundary conditions leaves the region open, then the problem is an initial value problem.

Each solution (analytic or numerical) is tailored to the equation; however, the three main categories will have similar solutions due to similarities in the problem. In numerical terms there are two general mechanisms for solving these equations: explicit and implicit. Explicit tends favor initial value problems, and has difficulties with round off errors. Implicit handles error quite well; however, complexity exists increases depending on the number points to be solved for. A third method also exists, Monte Carlo. However, the Monte Carlo method is more for "simulating the results of differential or integral equations.

Elliptical Differential Equations

Elliptical PDEs can be represented with Poison's formula. In the homogenous case, and Laplace's equation is the representing equation.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

Also, it is typical that a polygon is chosen to define the boundary conditions. Typically, the polygon is defined by either constants either in the first derivative or normal space. Simplest is a rectangle specified in the first derivative for the right and top edges and constants for the lower and left edges. Also, the central difference formula is used to setup a linear set equations to be solved. [5]. The trick is to set up the matrix, and then solve it by any means (classic or wavelet based).

Boundary conditions for elliptical PDEs consist of three catagories: Dirichlet, Neumann, and Robbin's. Dirichlet Boundary conditions are specified by either functions or constants on the solution function itself. Neumann Boundary conditions specify against the derivative of the function based

on some either independent or dependent variable. The combination of the two is called Robbin's Boundary Conditions. What does mean?

Boundary conditions define the starting point and limits for solving PDEs. This is necessary to have enough constants in the solution for a solution to be calculated. Different boundary condition types have different method of solution to account for the boundary conditions.

Parabolic Differential Equations

Parabolic type of differential equations usefully describe diffusion and fluid mechanics. Some analytical methods useful for solving parabolic equations include substitution of variables, Laplace and Fourier Transforms. Examples of Parabolic equations include heat diffusion, diffusion - convection, and Navier-Stokes.

Boundary conditions for parabolic PDEs are less formal than elliptical PDEs. Boundary conditions are specified either across the boundary, on the boundary, or a hybrid of the two. The boundary conditions are typically spelled by example in temperature equations. However; it is conceivable that other diffusion problems have similar issues.

Hyperbolic Differential Equations

Hyperbolic type of differential equations are prevalent in equations for oscillating phenomena. This includes electromagnetic fields and vibrating strings. In its analytical form, the D'Alembert Solution is an textbook example method. The D'Alembert Solution is an example the canonical form and it use in solving PDE problems.

In some literature canonical form is used to generate a sparse domain. The general rule for using canonical form comes from the D'Alembert Solution which is as follows:

1. Replace (x,t) by new canonical coordinates.
2. Solve the transformed equations.
3. Transform the solution into the original coordinates
4. Substitute the general solution into the IC's the acquire the constants.

One set of boundary conditions to watch out for are ones that can vary either their strength or position. Examples:

- $u_n + \lambda u = g(t)$
- $u_n = g(t)$
- $u = g(t)$

Some the boundary conditions that Farlow [11] refers to are controlled end points, force specified boundaries, and elastic attachments. In the case of force specified points, the boundary point itself can move in position. In the elastic attachment problem, the force can change on each of the boundary points. While, controlled end points are straight forward functions or constants. In each case, the force, position, and function of the boundary conditions must be accounted for through out the solution of the PDE.

Difference Equation Formulae

Central Difference Formulae:

1. $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$
2. $f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$

Backward Difference Formula:

- $f'(x) \approx \frac{f(x)-f(x-h)}{h}$

Forward Difference Formula

- $f'(x) \approx \frac{f(x+h)-f(x)}{h}$

Explicit Methods/ Iterative Methods

Explicit methods compute the solution of PDE by using the previous result to compute the current result. The first step is to acquire the solution is transform the PDE into its difference equation equivalent. Any of the difference equations can be used to acquire this form. What is important? The one line system of eqaution, and initial value formula are critical for setting.

The basic method is as follows:

1. Start with the index value of 0 (i=0 at the initial value)
2. Find the solution for all of x for $t = t_{i+1}$ by explicit formula.

3. Establish the boundary conditions with respect for $u_{t_{i+1},x}$ by boundary condition approximation formula.

4. Repeat steps 2 through 4 until i = n (where n is the maximum index).

The explicit method is used to solve a PDE where parts of it have been transformed into its central difference equivalent. The boundary conditions must also be computed in cases where they are not constants. In cases where the boundary conditions are specified on a derivative, those conditions must be placed in their difference equation form.

The strength of the explicit method is its speed. It is a top-down dynamic algorithm which uses only a few previous results to compute the next result. Thus each result is computed quickly. However, the explicit form is unstable, and tends to yield results inconsistent with the boundaries. In order for this form to be stable a reasonable Δx and Δt must be chosen.

Implicit Methods

One area where explicit and implicit methods differ is the arrangement of the linear equations used to solve the system. While in a sense both linear equations, implicit methods are computed as a system of equations.

An example of an implicit method is the Crank-Nicolson Method. A system of equations arrived at by converting the PDE into a system of difference equations.

Method:

1. Pick some value for λ such that $\lambda \in [0, 1]$

2. Pick Δx and Δt and assign grid points

3. Use computational molecule to generate equation.

4. Solve the matrix

Galerkin

The Galerkin method supposes that a complete orthonormal system $\{v_j\}_j$ is defined on $L^2([0,1])$ and every v_j is C^2 on [0,1]. The boundary conditions of the v_j are defined. The solution approximation is then defined on the span of this orthonormal system. For example, $u_s = \sum_{k\in\Lambda}(x_k v_k) \in S$ such that S is a span of $v_j : j \in \Lambda$, Λ is a subset of the natural numbers and x_j is a scalar. The catch is that u_s should behave as true solution a system of linear equations, i.e. a vector itself. The linear equations are the implicit set of equations for solving a PDE or ODE.

Frazier takes this one step further to show a parallel from Galerkin to a conventional implicit form. If L is a linear operation then

- $\langle Lu_s, v_j\rangle = \langle f, v_j\rangle \ \forall j \in \Lambda$ such that $\langle f, g\rangle = \int_0^1 f(t)\bar{g(t)}dt$
- Furthermore: $\langle L(\sum_{k\in\Lambda} x_k v_k), v_j\rangle = \langle f, v_j\rangle \ \forall j \in \Lambda$ leading to
- $\sum_{k\in\Lambda}\langle Lv_k, v_j\rangle x_k = \langle f, v_j\rangle \ \forall j \in \Lambda$

The final connection is that each element of a matrix A defined $A = (a_{j,k})_{j,k\in\Lambda}$ is a scalar defined by $\langle Lv_k, v_j\rangle$. This yields the following equality

$(a_{j,k})_{j,k\in\Lambda} = \langle Lv_k, v_j\rangle$ and $\sum_{k\in\Lambda} a_{j,k}x_k = y_j \ \forall j \in \Lambda$

- x is a vector $(x_k)_{k\in\Lambda}$
- y is a vector $(y_k)_{k\in\Lambda}$
- A is a matrix with rows and columns indexed by Λ
- $a_{j,k}$ is an individual element of A

With Galerkin, for all subsets in Λ we obtain an approximation $u_s \in S \rightarrow u$. This is done by solving $Ax = y$ and using x to determine u_s. One of the tricks is finding the the v_j's and x_j's such that the equations are satisfied.

4.1.2 Problems

The previous subsection discussed PDEs in general and how to solve them. Listed in this subsection are some common PDE problems. The selection has at least one problem each of the three PDE categories. In this section, the classic method of solution is applied. However, the wavelet methods are saved for the next section. Implicit and Galerkin solutions are generally chosen where accuracy is required, and explicit methods are shown when speed and complexity are necessary.

In order to make these solutions a few formulae need to be defined. These formulae are the central difference, forward difference, and backward difference formulae.

Semi-Infinite String Problem

The semi-infinite string problem is a typical resonance problem. Newton's physical laws derive the equation based on external forces, friction forces, restoration forces, and net forces. In the simplest

form, the problem is solved for net forces only. The problem is defined mathematically as:

- PDE $u_{tt} = c^2 u_{xx}$ $\forall x \in (0, \infty)$ and $\forall t \in (0, \infty)$
- BC $u(0, t) = 0$
- IC $u(x, 0) = f(x)$
- $u_t(x, 0) = g(x)$
- general solution: $u(x, t) = \frac{1}{2}[f(x - ct) + f(x + ct)] + \frac{1}{2c} \int^{x+ct} x - ctg(\zeta)d\zeta$
- $c^2 u_{xx}$ is the net force due to the tension on the string.
- u_{tt} represents the longitudinal or torsional vibrations on the string.

There is a conventional solution that comes from the central difference, and forward difference formulae. The rest is rather simple algebra. One key issue is the boundary conditions. Boundary values must be solved to establish the constants in the matrix. Once the boundary conditions are established, conventional methods can produce the solution. The conventional algebra is as follows:

- $u_{xx} = u[i, j + 1] - u[i, j] + u[i, j - 1]$
- $u_{tt} = u[i + 1, j] - u[i, j] + u[i - 1, j]$
- $u_t = (u[i + 1, j] - u[i, j])$
- $u[t, 0] = 0$
- $u[0, x] = f(x)$
- $u_t[0, x] = g(x) = u[1, x] - u[0, x]$

- $u[1, x] = g(x) + f(x)$
- $u[i + 1, j] + (c^2 - 1)u[i, j] + u[i - 1, j] - c^2u[i, j + 1] - c^2u[i, j - 1] = 0$

Heat Diffusion

The heat diffusion or heat conduction equation defines heat in a solid at any point and any time within the domain. Diffusion comes from a heat source, and may come from an artifical constant. In the case of heat diffusion, there is a constant α which is defined as the diffusitivity constant. Diffusitivity is defined by the following:

$\alpha = \frac{K}{\tau\sigma}$

such that K is the thermal constant, τ is the density and σ is the specific heat. The heat diffusion problem is defined as follows:

- $u_t = \alpha^2 u_{xx}$ such that
- u_t is the change in temperature with respect to time.
- u_{xx} is the concavity of the temperature

Diffusion - Convection

The big idea for diffusion - convection is that convection currents effects on the diffusion process. In particular, this addresses substances diffusing in areas where currents exists. One case is a gaseous substance being released in an air conditioned room or wind area. Another case is the release of a liquid substance into a river.

- $u_t = \alpha^2 u_{xx} - v u_x$ such that
- u_t is the change in time
- $\alpha^2 u_{xx}$ is the diffusion component
- $v u_x$ is the convection component
- v is the velocity of the convection current
- α is the diffusitivity constant

The solution to this problem is best done by considering the general solution, and then filling in the constants with the boundary conditions. The general solution is as follows:

- $u_t = \alpha^2 u_{xx} - v u_x$
- $u_t = u[i+1, j] - u[i, j]$
- $u_x = u[i, j+1] - u[i, j]$
- $u_{xx} = u[i, j+1] - 2u[i, j] + u[i, j-1]$
- $u_t - \alpha^2 u_{xx} - v u_x = 0$
- $u[i+1, j] - u[i, j] - \alpha^2 u[i, j+1] + \alpha^2 2u[i, j] - \alpha^2 u[i, j-1] - vu[i, j+1] + vu[i, j] = 0$
- $u[i+1, j]\alpha^2 u[i, j+1] - \alpha^2 u[i, j-1] - vu[i, j+1] + (2\alpha^2 + v - 1)u[i, j] = 0$

4.2 Related Work

4.2.1 Gregory Beylkin

One example applied mathematician in the field of wavelets is Gregory Beylkin of the University of Colorado. He has published several papers on the topic, and few have similar work to this paper. Some of his quotes deserve some explaination, and other do not. The main point is to use the wavelet transform to diagonalize the matrix for fast sparse methods to work.

One quote, "Fast algorithms for applying these operators to functions, solving integral equations. The operators which can be efficently treated using representations in the wavelet bases include Calderon-Zygmund and pseudo-differential operators."[7] This quote references two mathematical forms introduced in the early half of the 20th century. Calderon-Zygmund was difference operator that divided the vector in two each time it operated. The more generic version, the psuedo-difference operator was introduced shortly there after. Both introduced methods of numerically differentiating equations. Beylkin used kernel K to illustrate the use of the wavelet transform:

$$T(f)(x) = \int K(x,y)f(y)dy$$

This kernel allows for the mathematical explanation of how wavelets fit in scheme of solving PDE's. The bottom line is best said in Beylkin's own words, "If our starting point is a differential equation with boundary conditions then the wavelet system of coordinates there is a diagonal preconditioned which allows us to perform algebraic manipulations only with the sparse the sparse matrices. ... The wavelets play an auxiliary role in that they provide a system of coordinates where the condition numbers of the sparse matrices (involved in the computations) under control. "

Indeed, in Beylkin's examples the wavelets diagonalize the matrices for which it works on. Beylkin's multi-resolution scheme is unique in the way it provides the diagonalization process. It performed successively on each average term. However, the averaging and differencing is applied to portions in the vertical and horizontal components. Thus some of these parts are also forced two zero.

Belykin claims in another paper that wavelets support rapid application of dense matrix. In some cases, Beylkin claims to achieve operations in $O(N^2)$ operations. In general it is clear that wavelets offer the ability to reduce the overall size of the input, and therefore reduce the time necessary to compute the result. It is important to note that the input is an equivalent form to the original, and the new form is simply more compact.

On both integral applications, it is know that the algorithms have hidden recursive components that tend to make the algorithm grow exponentially. In cases described by Farlow [11], integral equations are solved in a similar manner to PDE's by using there numerical equivalence forms.

Lastly, Beylkin points out that wavelets have several advantages over the Fast Fourier Transform. Amongst them is the cost of the transform itself. The wavelet transform in 1-D space has a cost of nearly $O(N)$. where the FFT is of order $O(N^2)$ Another is translation invariance. The wavelet transform does not require translation invariance, where as the FFT does. This gives the wavelet transform some adaptability to numerical applications.

4.2.2 Turbulance and Navier Stokes

Jaffard, Meyer, and Ryan showed exploration into the basis functions useful for solving PDE problems such as Navier-Stokes and other turbulance problems. Amongst the first attempts for solving Navier-Stokes used the Battle-Federbush and Shannon basis. One of the keys for this problem was

the absense of boundary conditions.

The idea was to apply the Battle-Fererbush basis with the Gelerkin method to solve Navier Stokes. This concept was first published by Zenin in 1981. One of the difficulties is getting the matrix solution to diagonal form. The Shannon basis does not cause the non-diagonal elements to decay rapidly. In some cases, neither does Battle-Federbush.

The main issues for wavelets and PDE solutions in some opinions is to address complicated geometries. Others see that local refinements can enhance or be used in place of multi-grid algorithms in some cases. Again the key is to make the solution matrix converge to diagonal and sparse system.

4.2.3 KL Transform Approach

In the solution of the PDE in implicit form, the bottom line is to solve a linear equation. One approach is to use the Karhunen-Loeve transform. As stated by Wickerhauser [14] the overall goal this solution method is to acquire the singular values, eigenvalues, and eigenvectors. From these answers, the solution to the matrix is trivial. Methods for accomplishing this are factor analysis, principle component analysis, singular value decomposition, and the KL transform.

The point of wavelets with the KL-Transform was well stated "The approximate factor analysis algorithm is the search through a library of orthonormal basis for the one whose H is closest to that of the Karhunen-Loève basis, and cases of fast search methods the result is a fast approximate Karhunen-Loève algorithm. "[14]

So what problem is solved by the KL transform? "Two problems solved by principle orthogonal decomposition. First, distinguishing elements from a collection by making d measurements. Second,

inverting a complicated map from a p-parameter configuration space to d-dimensional measurement space. " [14] What does this mean? Usual cases for the extraction of singular values come measurements taken. The parameters necessary to get these values by certain formulae are extracted by their singular values which the KL transform can acquire. In the case of PDE's, these parameters represents a mapping to eigenvalues, eigenvectors, and singular values. As stated, "the Karhunen-Loève basis eigenvectors are also called principle orthogonal components or principal factors, and computing them for a given ensemble X is also called factor analysis."[14]

Where do wavelets come in? "It is possible to build a library of more than 2^d fast transforms U of R^d to use for "x" points.' Question: What does this mean? This means that wavelets can be used to construct the fast transforms U which make up the "x". From these basis, a particular wavelet basis can be selected as the best choice. Or so this idea is implying.

The method of using wavelets to acquire the KL transform are stated as follows:

- Expand N vectors $\{X_n \in R^d : n = 1, 2, \ldots, N\}$ into wavelet packets coefficients: $O(Nd \log d)$
- Summing squares into the variance tree: $O(d \log d)$
- Searching the variance tree for a best basis: $O(d + d \log d)$
- Sorting the best basis vectors into decreasing order $O(d \log d)$
- Diagonalizing the auto-covariance matrix of the top d' best basis vectors $O({d'}^3)$

Total complexity of the Approximate Karhunen-Loève basis: $O(Nd \log d + {d'}^3)$. Since $d' \ll d$, it safe to say the approximate solution is faster than the conventional one.

The approximate Karhunen-Loève transform of one vector

- Computing the wavelet packet coefficients of one vector $O(d \log d)$
- Applying the $d' \times d'$ matrix K'^{*} : $O(d'^{2})$

Updating the approximate Karhunen-Loève basis

- Expanding one vector into wavelet packet coefficients
- Adding the coefficients into the means tree
- Adding the squared coefficients into the squares tree
- Forming the variance tree and computing the new information costs
- Searching the variance tree for the joint best basis

The classification in large data sets apply to both rogues' gallery problem, fingerprint classification problem, and rank reduction for complex classifiers.

4.2.4 Galerkin Approach

4.2.5

Appendix A

Wavelets Implemented in General

A.1 Haar Wavelet Transform Class:

The Haar Wavelet Transform class has the purpose of taking in a signal and outputing the signal processed by the Haar Wavelet Transform. It is dependent on the convolution function for computation. It is also dependent on the haar function generator to establish a Haar Scaling and Wavelet vector. For practical I/O, a result plotter is also necessary.

All of the above can be accommodated within one class. One other class that is necessary is the myVector class. This class provides a simple, yet practical data type for the computation. It can be allocated and deallocated such that its size is appropriate for the task.

It is projected that the two-dimensional version will simply add a matrix manipulator. What such a manipulator extracts rows and columns into myVector structures such that the computation to

procede on each row and column..

A.1.1 One Dimensional Wavelet Transform

The primary function in the Wavelet transform class is the one dimensional wavelet transform (hWaveXform). More than one form of this function could possible exist; however, for simplisty only one was produced. It takes two myVector classes (input and output). It produces a haar difference and scaling vector, as well as a R_A, R_D, T, F and S myVector classes for use within the function. The algorithm is as follows

$S = input$

$l =$ is the size of S (or the input)

$\forall i \in [0, l)$

$R_A = S * H_A$

$R_D = S * H_D$

T = join (R_A,R_D)

F = evenSplit (T)

ForceInsert (F, output)

end = floor ($\frac{l}{2^{i+1}}$)

Extract (output, S, 0, end)

The Extract procedure is to take values from output up to the index end, and make S a copy of that vector. Thus S is called the resolution vector. T is always twice the size of the working S vector. F is always half. The resolution vector decreases by half each iteration, and starts at the same size as the input.

A.1.2 Join Procedure

The join procedure is rather simple. Produce a result whose size is the size of the two sources combined, and whose elements are those of the two input vectors.

Input:

myVector left, right

Output:

myVector Result

Algorithm

result.deallocate

$s_l = size(left)$

$s_r = size(right)$

$s = s_l + s_r$

result.allocate(s)

$\forall i \in [0, s_l)$

result[i]=left[i]

$\forall i \in [s_l, s)$

result[i]= right[$i - s_l$]

A.1.3 Procedure: Even Split

This procedure is simply a special type for condensing procedure. Simply put, this procedure makes the result half the size of the original input. Both the input and result are myVector classes. The characteristic of the elements of the result is:

result[i] = input[2*i]

Input:

myVector s

Output:

myVector r

Algorithm

l = ceil (s.size/2)

r.allocate(l)

$\forall i \in [0, l)$

$r_i = s_{i*2}$

A.1.4 Procedure: Force Insert

Purpose: To insert one myVector ,s, into a larger myVector, r. The constraint is that r be larger than s. As long as this is the case, then forced insertion can occur. Special cases can include start and end points. In which case, the start and end points must be within the specified range.

Input:

myVector s, r

Output:

myVector r

Algorithm:

$l_1 = s.size$

$l_2 = r.size$

if $(l_1 < l_2)$ and (start ¡ end)

$\forall i \in [start, end]$

$r_i = s_i$

A.1.5 Procedure: Extract

Purpose: To produce a new or replace a myVector whose length is that of the section to copied and extracted. The point behind this procedure is provide allow a wavelet transform to be performed on a segment of data, and keep the procedure the same each resolution.

Input:

The inputs for the wavelet Xform extraction procedure are as follows:

myVector S,

integer a

integer b

These names are arbitray. The variables a and b specify the start and end points respectively. Obviously, there is an inequality relationship here.

$0 \leq a \leq b \leq S.size$

Output:

The output of the wavelet Xform ¿ extraction procedure is simply:

myVector R

The elements of R are a copy of the elements of S from index a to index b.

Algorithm

if (0 ¡a ¡ b ¡ S.size)

R.deallocate

l = b - a +1

R.allocate(l)

$\forall i \in [a, b]$

$R_{i-a} = S_i$

A.1.6 Procedure: Haar Wavelet Inverse (Left Side)

Purpose: These procedures are specific case toward the 2-element Haar Wavelet. They take in an array of even length and return an array of equal size which is the Inverse Haar Wavelet Transform of the original. The format of the original is assumed to be $(A|D_1|D_2|...|D_n)$.

Input

A "MyVector" class which is an array with simple operations associated with it. The object name is source. The source is of the form $(A|D_1|D_2|...|D_n)$.

Output

A "MyVector" class with an object name of result.

Algorithm

There is a difference between the current algorithm and the ideal algorithm which may be the source of error. This algorithm uses the following symbols to aide in its description:

- S is the source myVector
- A is the average term for which $A_i = S_i$
- D is the difference term for which $D_i = S_{i+l/2}$

- l is the length of S

The first version has the following algorithm:

1. Initialize the result, R.
2. $\forall i \in [0, \frac{l}{2}]\ R_{2i} = (A_i + D_i)\sqrt{\frac{1}{2}}$
3. $\forall i \in [0, \frac{l}{2}]\ R_{2i-1} = (A_i + D_i)\sqrt{\frac{1}{2}}$

There is a problem with this method. Let us start with the even values: $R_0 = (A_0 + D_0)\sqrt{1/2}$. Now the odd, not that $R_{-1} = (A_0 + D_0)\sqrt{1/2}$ which of course does not exist. Next, $R_1 = (A_1 - D_1)\sqrt{1/2}$ is valid. Lastly, $R_{l-1} = R_{2*(l/2)-1} = (A_{l/2} - D_{l/2})\sqrt{1/2}$. The problem is that $A_{l/2}$ and $D_{l/2}$ do not exist. Thus, R_{l-1} does not exist, either.

A.2 2-D Wavelet Transform Class

This class provides three functions. The column wavelet provides a transform solely on the columns. The row wavelet likewise, provides a wavelet transform on each of the rows. The 2-D wavelet transform is simple a row wavelet followed by a column wavelet transform.

It should be noted that these wavelet transforms are Haar Wavelet transforms.. Multiple resolutions functions can be made, but again it is still the Haar Wavelet at the core. Other class clones may be produced to use other wavelet basis such as Daubachie or Coeflet.

Another support class required for support of the two dimensional wavelet is the image reader/ writer. Such a class of functions provide means of acquiring data for a matrix to be computed. Granted this matrix could have been populated by other means. Generating such a class consumed some time on this project. The reason is that there a few mechanisms for doing this. One is to use raw image types. These types are known as by their extensions: pgm and ppm. Another popular mechanism is Image Magick. The reason for Image Magick is that it is available on most UNIX platforms (Linux, IRIX, Solaris, and Mac OSX). Otherwise, schemes such as Apple's Quicktime would be used. The other reason for the use of other software to acquire the image is that the objective of this class is not produce image translators for each image type imaginable.

A.2.1 Method: Column Wavelet Transform

The column wavelet transform is provides a source matrix, and returns a result. In order to yield this result, each column is extracted into a vector and that vector is fed into a one-dimensional transform. The result of the one dimensional transform is placed the corresponding row of the result matrix. Mathematically, this algorithm is as follows:

$\forall j \in col$

n = 0

$\forall i \in row$ in reverse order

S_{n++} =source$_{i,j}$

$S\ W \Rightarrow R$

n = 0

$\forall i \in row$ in reverse order

result$_{i,j}$ $= R_{n++}$

Note that the need for the n index is keep the order straight for this wavelets convention.

A.2.2 Method: Row Wavelet Transform

Like the column wavelet transform, the row wavelet transform takes a source matrix and returns a resulting matrix. The only two significant differences are first the items being operated on (rows not columns). Second, the lack of need for the n index.

$\forall i \in row$

$\forall j \in col$

S_j =source$_{i,j}$

$S\ W \Rightarrow R$

n = 0

$\forall j \in col$ in reverse order

result$_{i,j}$ $= R_j$

A.2.3 Method: Self Row/Column Wavelet Transform

There is a significant difference between the proof of concept version and self row/column wavelet transforms. The self versions use a matrix convolution.

A.2.4 Method:Wavelet Transform

As stated above, the two dimensional wavelet transform is simply row wavelet transform followed by a column wavelet transform. It could have been done in reverse order. However, the result would be the same.

Just of note, this class lacks for the moment an inverse transform function. Not that such a thing does not exist mathematically. It simply was not implemented at the time of this document.

A.2.5 Method: Row Wavelet Inverse Transform

The row wavelet inverse transform uses the one-dimensional form to transform each row of the matrix. The algorithm is as follows:

1. $\forall i \in [0, k)$
   (a) Assign S the values of row i in such that $S_i = M_{i,j}$
   (b) Perform inverse wavelet transform on S: $S \rightarrow R$
   (c) reintegrate R into result such that $N_{i,j} = Rj$

The above algorithm is defined with the following notation

- S is the source vector for use in the inverse wavelet transform.
- R is the result vector used in the inverse wavelet transform.
- M is the source matrix
- N is the result matrix

A.2.6 Method: Column Wavelet Inverse Transform

The column wavelet inverse transform is nearly identical to the row wavelet inverse transform. The exception is of course that the source vector S is assigned to equal the columns. Another significant issue is that the column indices and source vector indices are in reverse order.

$S_j = M_{i,l-j}$ and $N_{i,l-j} = R_j$

A.2.7 Method: Wavelet Inverse Transform v1

The wavelet inverse transform simply calls the row wavelet inverse transform first, and uses the results to in column wavelet inverse transform. The result of the column wavelet inverse transform is used as the result for the wavelet inverse transform.

A.2.8 Method: Self Wavelet Inverse Transform

This method is used to save on memory leaks by only allocating the temporary matrix and result only once. This method uses two other methods, the Self Column Inverse Wavelet Transform and the Self Row Inverse Wavelet Transform.

Given

The two items provided are references to the source matrix and the result matrix.

Algorithm

- $\forall j \in W.columnsSelfColumnInverseXform(W, R, j)$
- $\forall j \in W.rowsSelfRowInverseXform(W, R, i)$

A.2.9 Method: Self Column Inverse Wavelet Transform

The code name for this method is selfColumnInverseXform, and it takes three arguments. This particular method performs a column inverse wavelet transform on a particular column, designated j. The return value is placed in R, in the correct column.

Given

Two references are given. One to the source matrix, and the other to the result matrix. The third item is an integer, j. This integer identifies the column to be transformed.

Notation

Two symbols are used to simplify the writing of the algorithm.

- k is number of rows in source matrix minus 1.
- k2 is the half the number of rows in the source matrix.
- W is the source matrix
- R is the result matrix.

Algorithm

$\forall i \in [0, k2)$

- $R_{2i,j} = (W_{i,j} + W_{i+k2,j})\sqrt{\frac{1}{2}}$
- $R_{2i+1,j} = (W_{i+k2,j} - W_{i,j})\sqrt{\frac{1}{2}}$

A.2.10 Method: Self Row Inverse Wavelet Transform

The code name for this method is selfRowInverseXform, and it takes three arguments. This particular method performs a column inverse wavelet transform on a particular column, designated i. The return value is placed in R, in the correct column.

Given

Two references are given. One to the source matrix, and the other to the result matrix. The third item is an integer, j. This integer identifies the column to be transformed.

Notation

Two symbols are used to simplify the writing of the algorithm.

- l is number of columns in source matrix minus 1.
- l2 is the half the number of columns in the source matrix.
- W is the source matrix
- R is the result matrix.

Algorithm

$\forall i \in [0, l2)$

- $R_{i,2j} = (W_{i,j} - W_{i,j+l2})\sqrt{\frac{1}{2}}$
- $R_{i,2j+1} = (W_{i,j+l2} + W_{i,j})\sqrt{\frac{1}{2}}$

Bibliography

[1] Howard L. Resnikoff. and Raymond O. Wells, Jr. *Wavelet Analysis: The Scalable Structure of Information* copyright Springer-Verlag New York, Inc. New York, NY 10010, USA, 1998

[2] James Walker *A Primer on Wavelets and Their Scientific Applications* copyright Chapman & Hall/CRC : Boca Raton, FL, USA, 1999

[3] Gavin Tabor *Wavelets - The Idiots Guide* http://monet.me.ic.ac.uk/people/gavin/java/waveletDemos.html

[4] Amara Graps *Introduction to Wavelets* http://www.amara.com/current/wavelet.html

[5] Singiresu S. Rao *Applied Numerical Methods for Engineers and Scientists* published Prentice Hall Upper Saddle River, NJ 07458 copyright 2002

[6] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery *Numerical Methods in C* Published by the Press Syndicate of the University of Cambridge The Pitt Building, Trumpington Street, Cambridge CB2 1RP 40 West 20th Street, New York, NY 10011-4211, USA Copyright Cambridge University Press 1988, 1992

[7] G. Beylkin *On wavelet-based algorithms for solving differential equations*

[8] G. Beylkin *Wavelets and Fast Numerical Algorithms*

[9] G. Beylkin, R. Coifman, and V. Rokhlin *Fast Wavelet Transforms and Numerical Algorithms I, Article in Communications on pure and applied mathematics* copyright 1991 Wiley, New York

[10] G. Beylkin, D. Gines and L. Vozovoi *Adaptive Solution of Partial Differential Equations in Multiwavelet Bases*

[11] Stanly J. Farlow *Partial Differential Equations for Scientists and Engineers* copywrite 1993 Dover Publications, Inc Mineola, N.Y. 11501

[12] Murray R. Spiegel, *Theory and Problems of Advanced Mathematics for Engineers and Scientist* copy-write 1996 McGraw-Hill

[13] Stephane Jaffard, Yves Meyer, and Robert D. Ryan *Wavelets: Tools for Science and Technology* copyright 2001 by the Society for Industrial and Applied Mathematics Philadelphia, PA 19104-2688

[14] Mladen Victor Wickerhauser *Adapted Wavelet Analysis from Theory to Software* copyright 2001 by the Society for Industrial and Applied Mathematics Philadelphia, PA 19104-2688

[15] n-Bing Lin and Zhengchu Xiao *Multiwavelet Solutions for the Dirichlet Problem* Department of Mathematics, University of Toledo; Toledo, OH 43606, USA

[16] ian-Xiao He *Wavelet Analysis and Multiresolution Methods (Lecture Notes in Pure and Applied Mathematics* published Marcel Dekker, Inc. New York, NY 10016 copyright 2000