BSce Advanced Studies Project
Semester 2 2004
Using Wavelet Transforms to Assist the Solution of van der Waals Forces

Brendon Lloyd Higgins
2091482

25th October 2004

1 Introduction

The interactions of and between electrons in matter are the cause of what is known as van der Waals forces [10]. These forces become significant in soft matter systems. To predict the properties of such systems equations involving the electron interactions must be solved, and by the many body nature of the problem it makes the solution of such equations necessarily numeric. The random phase approximation method accounts for electron correlation through approximation and produces good results, however it becomes computationally difficult to solve [4].

Wavelet theory, a relatively new tool in mathematics, has only recently found numerous applications in physics [5], and audiovisual processing and compression [9]. The concept of multiresolution analysis, a fundamental part of wavelet theory, has allowed for the generation of a limitless array of wavelet bases suited to all kinds of problems.

This project employed several simple wavelet bases in the transformation of the matrices involved in the random phase approximation with the aim of finding a basis that makes those matrices diagonal and sparse, and thus relatively easier to solve.

2 The Physical Problem

First-principle predictions of phenomena are important in physics. Cohesive energy of atoms in solids and molecules in solid state physics, materials science, and nanoscience generally depends on the quantum mechanical groundstate energy of the electrons in a system of interest. Prediction from first-principles of such a property of systems is made difficult by the fact that the interaction and correlation of the motion of electrons makes the calculation of this energy a quantum mechanical Many Body Problem (MBP), which is known to be insoluble.

A widely used technique to solve this problem is known as the Local Density Approximation (LDA). LDA works well for solids and large molecules where the correlations between electrons over more than a typical atomic spacing are insignificant, but for systems where such distant correlations are important the LDA fails [3]. These distant interactions lead to an energy known as the van der Waals energy, which causes soft van der Waals forces. These forces become significant in "soft" matter such as graphite. (Graphitic systems are currently an active area of research as they are being investigated as hydrogen storage systems.)

From Many Body Theory we have the adiabatic connection-fluctuation-dissipation theorem (ACF-FDT), which shows that the groundstate electron energy can be found by integrating a "density-density response" function $\chi(r, r', t, t')$. This function describes how the number density of electrons is changed at position r and time t when a small external potential is applied at position r' and time t'. This shows that the interactions between electrons can be found from the way the electrons respond to external influences. This is still an MBP, but a simple approximation known as the Random Phase Approximation (RPA) produces fairly good results, leading through ACF-FDT back to a prediction for the groundstate electron energy which includes the distant interactions between electrons that the LDA misses.

Random phase approximation works by splitting χ into two parts. The first part, χ_0, is the density-density response function of a theoretical system in which the electron interactions have been replaced with a single electron potential — the LDA potential. The density perturbation χ_0 is just the sum of the density perturbations of many independent single body Shrödinger solutions. A change in potential δV^{ext} will produce a change in electron density δn_0 at a point z that, through first order perturbation theory, is given by

$$\delta n_0(z, \omega) = \int \chi_0(z, z', \omega)\, \delta V^{\text{ext}}(z', \omega)\, dz'$$

For simplicity we work in a single spacial dimension considering "sheets" of charge at positions denoted by z and z', and instead of time delay $t - t'$ we use frequency ω by a Fourier transform. The physics of this system resembles layered compounds. A classic test problem for investigating

quantum many-electron theories is a layered system like this called a "jellium slab", where a slab of uniform density of positive charge "holds" the mobile electrons.

The calculation of χ_0, while messy, is achievable even for complex systems. χ_0 is known as the "independent-electron" or Kohn-Sham response function, and is sometimes alternatively denoted χ_{s} or χ_{KS}.

The second part of the χ function in the RPA is an approximation of the electron interaction. The density response due to electron interaction is caused by a potential generated by the perturbed density. This internal potential is given by

$$\delta V^{\mathrm{int}}(z,\omega) = \int V^{\mathrm{coul}}(z,z')\,\delta n(z',\omega)\,dz' \tag{1}$$

where V^{coul} is the Coulomb potential between electrons and $\delta n(z',\omega)$ is the change in density at z'.

We know the Coulomb potential between two uniformly charged sheets is

$$V^{\mathrm{coul}}(z,z') = -2\pi e^2\,|z-z'|$$

and has the property that

$$\frac{d^2}{dz^2}V^{\mathrm{coul}}(z,z') = -4\pi e^2\delta(z-z')$$

where $\delta(z-z')$ is the Dirac delta function. From the independent electron response function we now arrive at

$$\delta n(z,\omega) = \int \chi_0(z,z',\omega)\,\delta V^{\mathrm{tot}}(z',\omega)\,dz' \tag{2}$$

where

$$\delta V^{\mathrm{tot}}(z',\omega) = \delta V^{\mathrm{ext}}(z',\omega) + \delta V^{\mathrm{int}}(z',\omega) \tag{3}$$

is the total change in potential. This is an approximation since δV^{tot} represents the effect on an infinitesimal sheet of test charge that does not disturb the source charges. In reality the test charge is not infinitesimal and does affect the source in a way which the RPA does not account for, but most of the necessary physics is present in the ACF-FDT and the approximation does produce good results.

Combining Equations 1, 2, and 3 we have

$$\delta n(z) = \int \chi_0(z,z',\omega)\left(\delta V^{\mathrm{ext}}(z',\omega) + \int V^{\mathrm{coul}}(z',z'')\,\delta n(z'',\omega)\,dz''\right)dz' \tag{4}$$

This is one form of the "RPA screening equation". By solving this integral equation we can deduce χ.

Alternatively, we can take the second derivative of Equation 1 and turn the problem into a Poisson's equation, thus

$$\begin{aligned}\frac{d^2}{dz^2}\delta V^{\mathrm{int}}(z,\omega) &= 2\pi e^2\delta n(z,\omega)\\ &= 2\pi e^2\int \chi_0(z,z',\omega)\left(\delta V^{\mathrm{ext}}(z',\omega)+\delta V^{\mathrm{int}}(z',\omega)\right)dz'\end{aligned} \tag{5}$$

by Equations 2 and 3. This is another form of the RPA screening equation, and it is a differentio-integral equation in δV^{int}. From this we can generate δn and hence find χ. We could solve either Equations 4 or 5 and eventually result in, through the ACF-FDT, a groundstate energy that includes van der Waals type long-ranged electron interactions. [2]

For this project we consider using numerical techniques to solve Equation 5. The problem with this approach is that, while the $\frac{d^2}{dz^2}$ operator is very well localised, the χ_0 function is not. This makes the matrices used to solve the equation dense and thus difficult and time-consuming to invert, especially with large systems. In an effort to solve this problem we look at reinterpreting χ_0 and $\frac{d^2}{dz^2}$ using wavelet bases in the hope that they make those matrices easier to solve.

3 Introduction to Wavelets

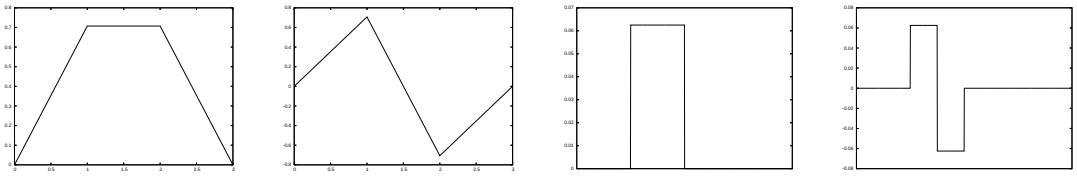
Wavelets are a family of bases. There are many different types of wavelets, all with different properties which suit different types of problems. The simplest and easiest to understand is the Haar wavelet.

3.1 Haar Wavelets

The most basic wavelet transform is the Haar transform. A simplified version of the Haar transform algorithm is to split an input signal into two output subsignals, one that records the average of successive pairs of samples in the input signal, and another that records their difference. As an example, derived from Walker [9, p. 3], say you have a signal sampled with the values 4, 6, 10, 12, 8, 6, 5, and 5. The averages of pairs of these samples are 5, 11, 7, and 5, and their differences are -2, -2, 2, and 0. So this simplified Haar transform would leave you with an output signal consisting of 5, 11, 7, 5, -2, -2, 2, and 0. This data completely represents the original signal, and the process used to create these samples can be easily reversed to obtain the original samples.

The real Haar transform extends this process slightly by taking, instead of straight averages and differences, the sum of pairs divided by $\sqrt{2}$ to give a "trend" subsignal, and the difference of pairs over $\sqrt{2}$ to give a "fluctuation" subsignal. This results in the transform signal having the same energy (sum of squares) as the original signal, but most of that energy has been shifted into the trend subsignal, or "compacted" [9, p. 6].

The Haar transform can be written in terms of inner products of vectors. If we have our signal in vector notation as $f = (4, 6, 10, 12, 8, 6, 5, 5)$, the Haar transform would have two sets of vectors, "scaling functions" $V_1^1 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0\right)$, $V_2^1 = \left(0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0, 0, 0\right)$, ..., corresponding to the sums part, and "wavelets" $W_1^1 = \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0\right)$, $W_2^1 = \left(0, 0, \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0, 0, 0\right)$, ..., corresponding to the differences part. All the scaling functions and wavelets can be produced from the first scaling function/wavelet by translating the entries $2n$ places to the right. It can be shown that the samples in the trend subsignal $a_n = f \cdot V_n^1$, and similarly for the samples in the fluctuation subsignal $d_n = f \cdot W_n^1$.



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 1: Haar Wavelet

3.2 Multiresolution Analysis

Multiresolution analysis (MRA) is the heart of wavelet techniques [9, p. 13]. The concept of MRA is that there are multiple fluctuation subspaces. Consider the example above. If we have the real Haar trend subsignal $\left(5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2}\right)$ we could take its Haar transform and split it into trend and fluctuation subsignals just like was done for the original signal. This would result in a complete representation of the original signal that involves a first level fluctuation subsignal (from 5th to 8th entry), a second level fluctuation subsignal (3rd and 4th entries), and a second level trend subsignal (1st and 2nd entries). This is known as a "2-level transform", and it further compacts the energy of the original signal.

Higher level transforms of signals can be taken. Due to sample nature, however, there is a limit to the resolution level at which a signal can be transformed. This limit depends on both the number of samples in the trend subsignal and the "support length" (described in the next section) of the wavelet basis used. Where n is the number of entries in the trend subsignal and s is the support of the wavelet basis, a transform can only be performed while $n > s$ and n is wholly divisible by 2. For example, a signal of 128 samples under a wavelet with support length 6 can only be transformed up to the 5th level. This limits the number of samples in a signal which is to be wavelet transformed to multiples of two, the size at which the highest resolution transforms can be taken being powers of two.

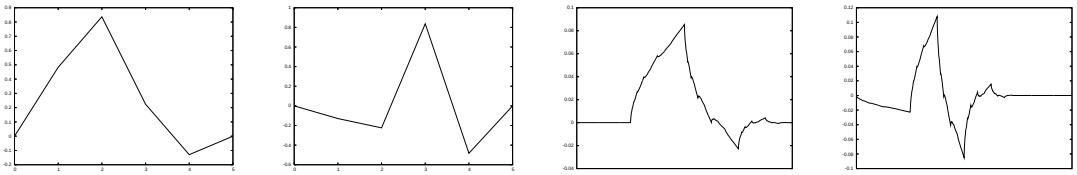
3.3 Beyond Haar Wavelets

The following figures and Figure 1 on the preceding page for the Haar Wavelet were generated from wavelet coefficients and via the method described in Goedecker [6, p. 16]. All the wavelets considered in this project have compact support (a finite number of coefficients) and are orthogonal.

Daubechies' Wavelets

Just like Haar wavelets, Daubechies wavelets split a signal into what could be considered running average and running difference subsignals [9, p. 29]. The main difference is that the Daubechies transform takes into account not just two adjacent samples, but four, six, ten, or even twenty adjacent samples (really, any multiple of two adjacent samples).

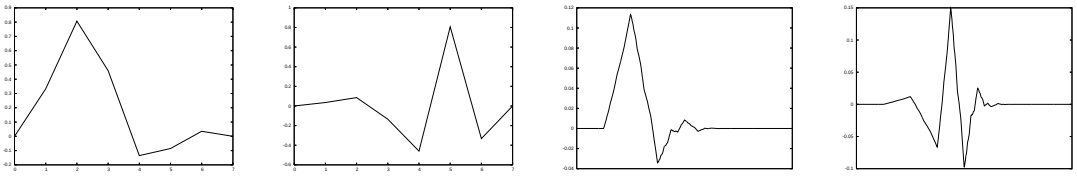
The process of performing the transform is the same as described for the Haar transform; V and W vectors are generated by translation of entries in each by two places (wrapping the entries where necessary), and the inner products are taken. The only difference between the Daubechies wavelet transform and the Haar wavelet transform is the way V and W are defined. The entries in the V (scaling) and W (wavelet) Daub4 vectors, for example, can be seen in Figure 2a and Figure 2b, respectively. The technical details of the definition of Daubechies wavelets are omitted here.



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

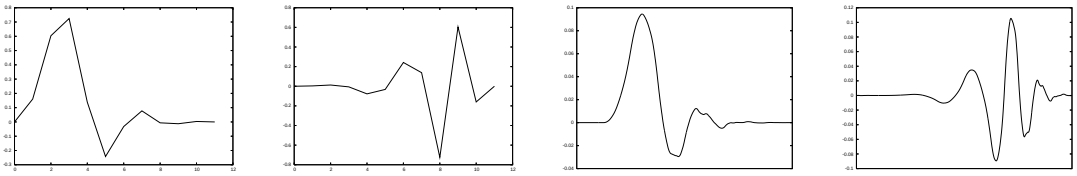
Figure 2: Daub4 Wavelet

Other Daubechies wavelets are defined similarly. The N in Daub$N$ indicates the number of adjacent samples in the signal that are considered, the number of entries in the V and W vectors, known as the wavelet's "support length".



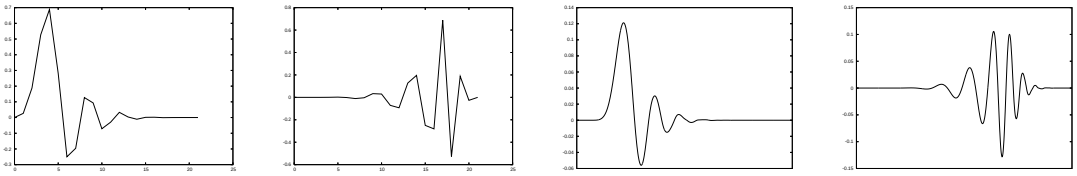
(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 3: Daub6 Wavelet



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 4: Daub10 Wavelet

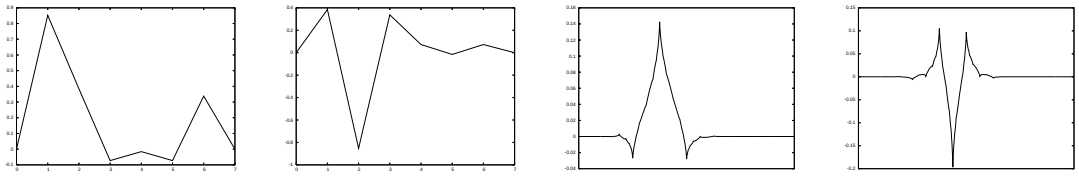


(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 5: Daub20 Wavelet

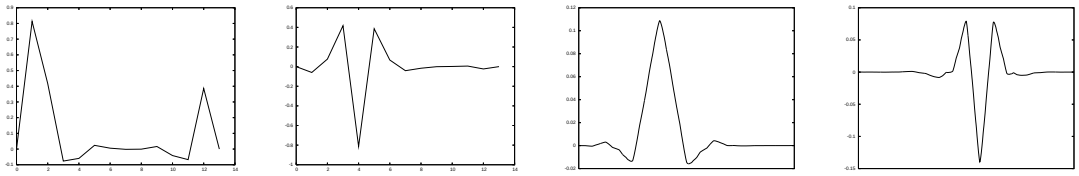
Coiflets

Coiflets are similar to Daubechies' wavelets except that the trend values from a coiflet transform are a much closer match to the original signal [6, p. 48]. The Coif$N$ family is constructed similarly to the Daub$N$ family.



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 6: Coif6 Wavelet

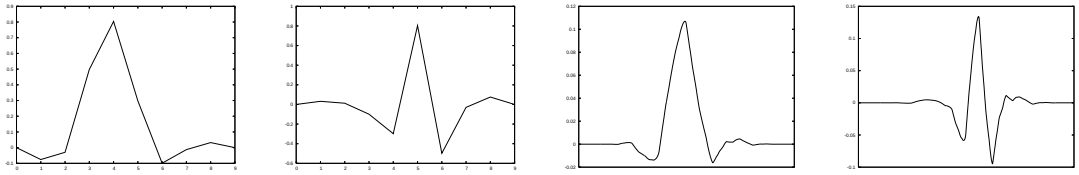


(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 7: Coif12 Wavelet

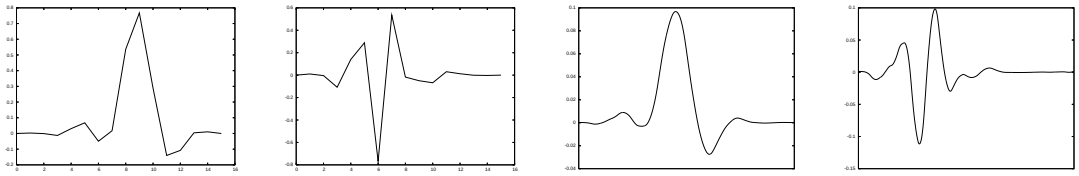
Symmlets

It has been proved that no compactly supported wavelet basis other than the Haar basis can be both orthogonal and symmetric [1, p. 252]. Symmlets are an effort to get as close to symmetry as possible [7].



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 8: Sym4 Wavelet



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 9: Sym7 Wavelet

More

There are many many other numbers, families and types of wavelets. For example, others in the Daubechies, Coiflet, or Symmlet families exist and could be considered. There are also more families of orthogonal wavelets that have not been shown here. Beyond that, there are biorthogonal, spline, second-generation, and other advanced wavelet families [1, 6, 8] that will not be examined in this project.

3.4 Two-Dimensional Signal Transforms

The wavelet transforms shown thus far have only considered one-dimensional signals akin to audio signals. Wavelet transforms can be easily expanded into two-dimensional signals akin to images or matrices. The wavelet transform of an image or matrix is simply a transform of each column of the image, and then a transform of each row of the resulting image [9, p. 67]. Transforming rows before columns is equivalent.

3.5 Transform Methods

There are three methods for wavelet transforms in two dimensions:

Standard

The normal transform described above.

Packet

When performing MRA, instead of only transforming the trend subsignal when transforming to a higher level, the wavelet packet transform also transforms the fluctuation subsignal. An n-level packet transform will thus generate 2^n subsignals, instead of the usual $n + 1$ subsignals. This method can also be applied to one-dimensional signals.

Non-Standard

This involves the artificial insertion of zeros into the result of a transform between successive level transforms. It is best considered in the context of a wavelet transform of an operator matrix. Most of the entries in such a matrix are zero, except for some near or on the diagonal. With such a matrix each successive level is "coupled". Take for example the Daub4 transform of the $\frac{d^2}{dz^2}$ operator matrix shown below.

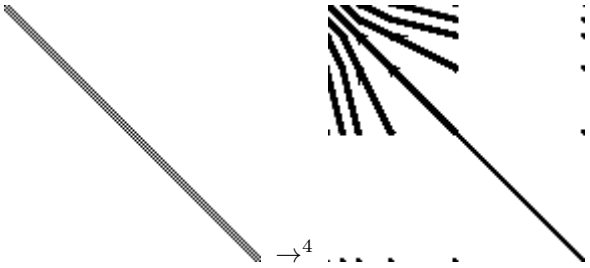
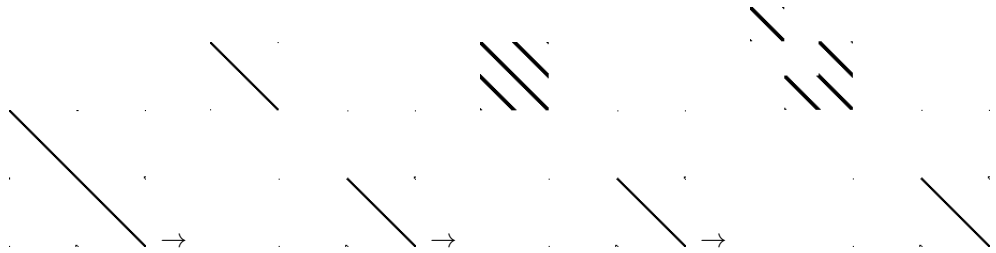


Figure 10: Coupled Resolution Levels in the Daub4 Transform of $\frac{d^2}{dz^2}$

The coupling effect is apparent in the top left corner of the 4-level transform shown in Figure 10. For each level the fluctuation values for the horizontal transform are used in the next vertical transform. By artificially inserting zeros between the fluctuation and trend signals in each level we can completely decouple different resolution levels [6, p. 40]. The process is illustrated in Figure 11.



(a) Transform (b) Insert Zeros (c) Transform (d) Insert Zeros

Figure 11: Decoupled Resolution Levels in the Daub4 Non-Standard Transform of $\frac{d^2}{dz^2}$

This decoupling of levels produces matrices that are sparser and more diagonal than the standard transform, thus making them easier to invert. The drawback, however, is that the matrices are now larger than the original, and there is some redundancy introduced. It is important to consider comparing the efficiency of inversion gained from the decoupling of resolutions to the efficiency lost due to the matrix growing larger.

4 Applying Wavelets to the χ_0 and $\frac{d^2}{dz^2}$ Matrices

4.1 Aim

For this project we are given a χ_0 matrix for a quasi-1-dimensional system. This matrix is diagonal, but quite dense with significant values over almost the entire matrix (see Section 4.3). The aim of this project is to investigate a number of different wavelet transforms of the χ_0 and $\frac{d^2}{dz^2}$ matrices in order to find a transform that makes both these matrices sparse and diagonal, and thus easier to solve.

4.2 Measuring Success

We need a way to measure the solubility of a transformed matrix in order to determine if a particular transform has made our problem any easier. One way this can be achieved is to determine the "density" of the matrix. This is done by counting the number of entries in the matrix whose absolute value is above some small threshold value. (This is technically not a density — it's more akin to flux — but for the most part this is irrelevant.) This method effectively gives a count of how many significant entries are in the matrix — the higher the number, the more entries in

the matrix are significant, and the more difficult the matrix will be to solve. The code to do this can be found in Appendix B1. The method used in this code also considers entries less than the threshold by adding to the result the ratio of the entry to the threshold. For this project, the threshold was chosen to be 10^{-4}.

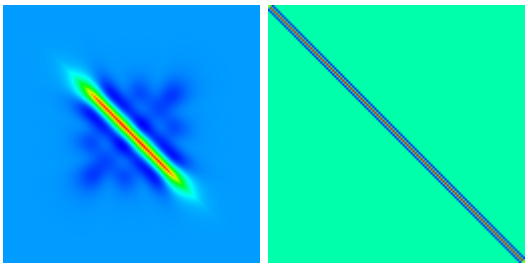
There is a flaw with this method. Matrices with significant values away from the diagonal are considerably harder to solve than matrices with significant values near the diagonal. This fact is not taken into consideration by the density function, and thus two matrices which have major differences in the ease of which they can be solved, owing to the location of significant values, may have the same density.

For this reason, visual inspection is still important. To this end we will also consider a "significance image", a pictorial representation of a matrix whose elements are the absolute of the respective elements of the measured matrix, and whose grey-scale-map will be in the range from 0 to a small threshold (10^{-4}), 0 being white and the threshold, and any value greater than that threshold, being black. The code for this can be seen in Appendix B10. These images allow us to see where in a matrix the significant entries appear. This qualitative measurement, coupled with the quantitative density function, should give a reasonable idea as to how easy a particular matrix will be to solve.

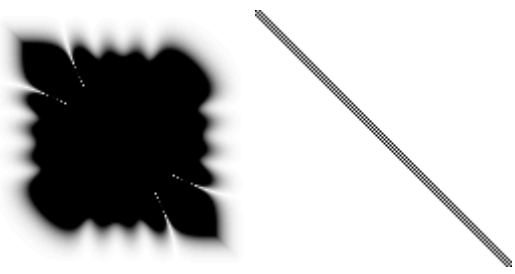
4.3 The Original Matrices

The original χ_0 and $\frac{d^2}{dz^2}$ matrices are plotted here for comparison to the transforms that will be shown in the next section. It should be noted that the original χ_0 matrix has 129^2 data points due to the nature of the algorithms used to generate it. The last row and column, which consist entirely of zeros, were discarded to achieve a size of 128^2 to suit wavelet transforms of multiple resolutions (see Section 3.2).

The colours in signal images range from red (minimum) to blue (maximum) with the remainder of the visible spectrum in between. Do note that the same colour will likely have a different corresponding value between images. The significance map (Sig. Map), however, has white corresponding to 0 and black corresponding to $|M_{x,y}| \geq 10^{-4}$ where $M_{x,y}$ is the matrix entry at x and y, and thus they are consistent between images and can be compared.



(a) χ_0 (b) $\frac{d^2}{dz^2}$



(c) χ_0 Sig. Map (d) $\frac{d^2}{dz^2}$ Sig. Map

Figure 12: Original Signals

By visual inspection we can see that the $\frac{d^2}{dz^2}$ signal is already very diagonal and sparse, while the χ_0 signal is also diagonal, but is very dense. It would be accurate to suggest that a wavelet basis that does well at making the χ_0 matrix sparse runs the risk of making the $\frac{d^2}{dz^2}$ matrix dense. We will try a number of different wavelet bases in the hope of finding one or more that may be suitable. The following sections show these transforms visually, followed by comparisons of density functions of all the transforms.

4.4 The Haar Transform

Standard

In the figure on this page, each successive resolution is pictured next to each other, increasing from left to right, with a corresponding significance image underneath each. Not all possible resolutions are listed — only the first four are given — though all possible resolutions were considered and their densities appear in the graphs in Section 4.8.

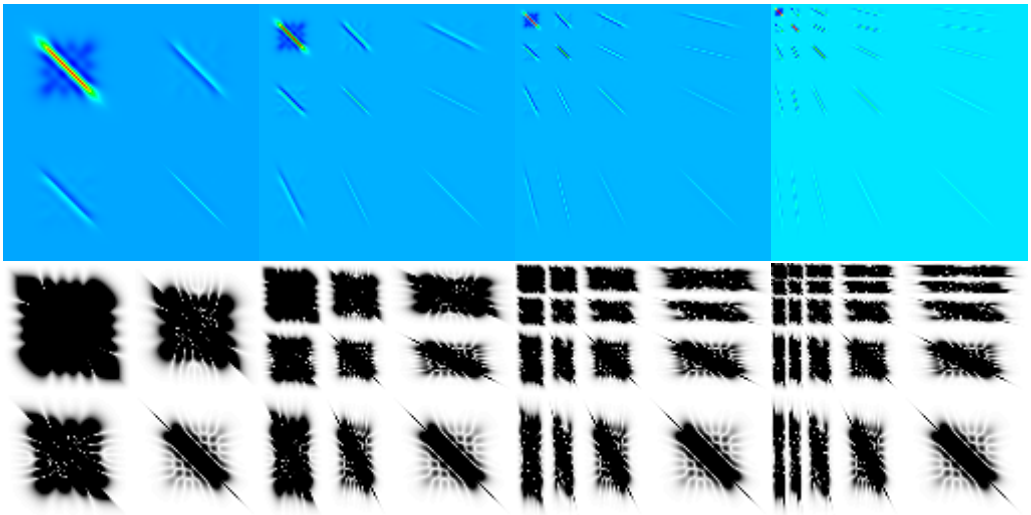


Figure 13: Haar Standard Transform of χ_0

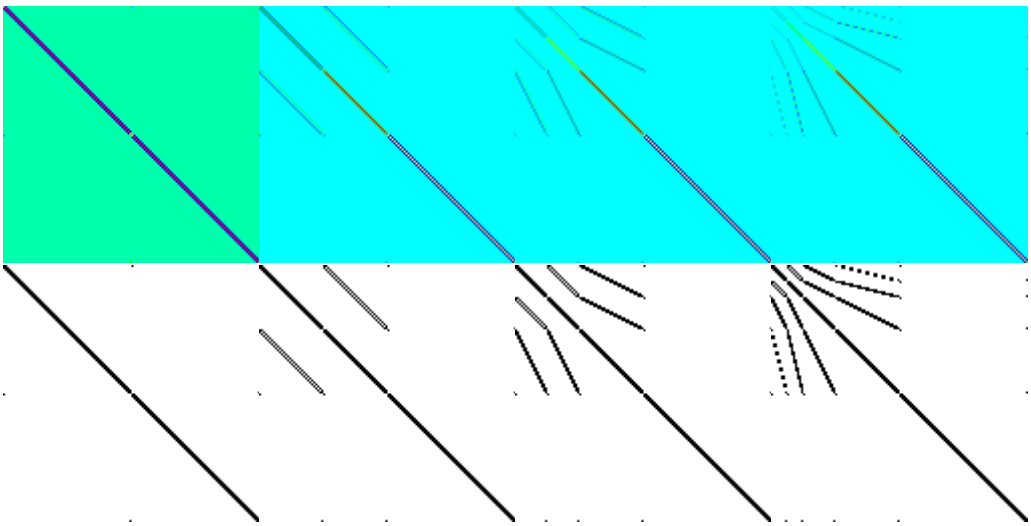


Figure 14: Haar Standard Transform of $\frac{d^2}{dz^2}$

The standard Haar transform does not do particularly well with either the χ_0 or the $\frac{d^2}{dz^2}$ matrices. It seems to make both matrices less diagonal, especially the $\frac{d^2}{dz^2}$ matrix.

Packet

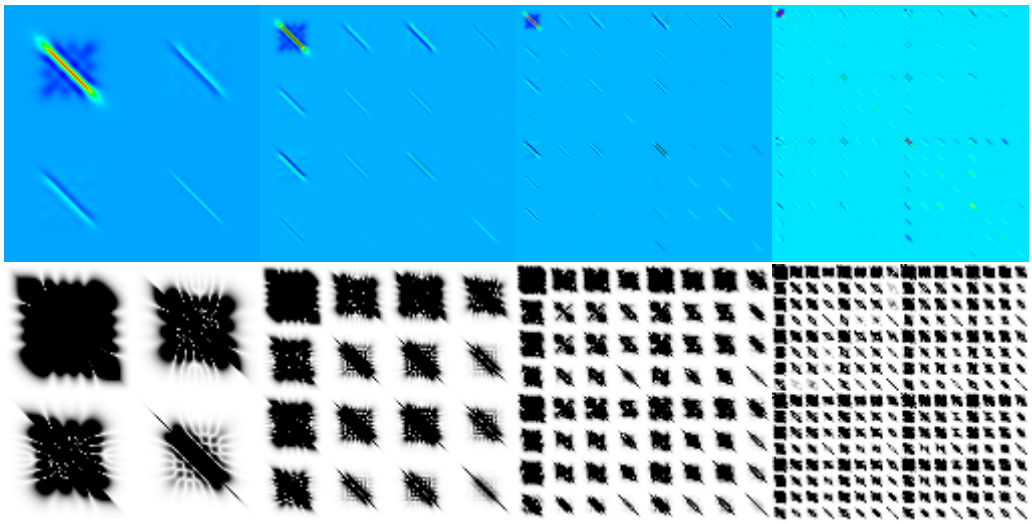


Figure 15: Haar Packet Transform of χ_0

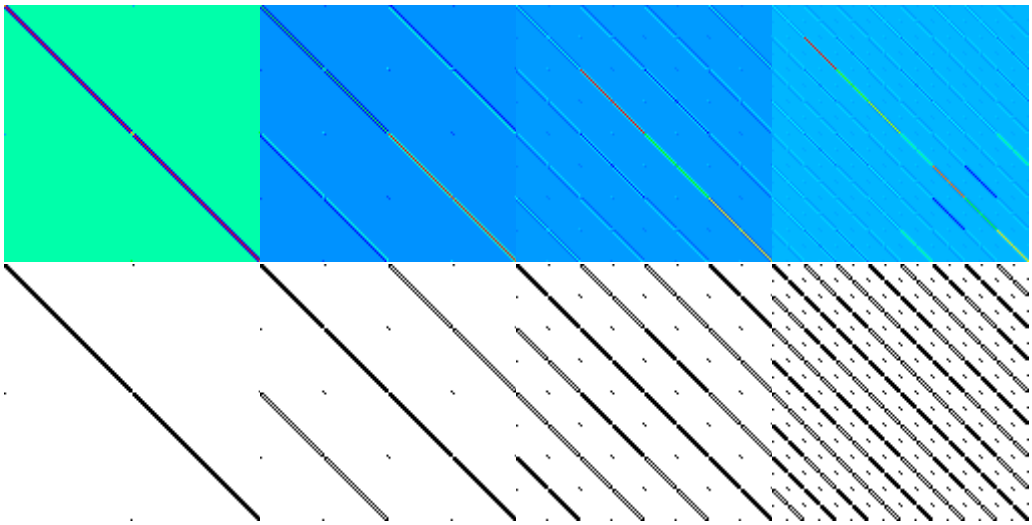


Figure 16: Haar Packet Transform of $\frac{d^2}{dz^2}$

The Haar packet transform is also no good. It spreads significant values away from the diagonal.

Non-Standard

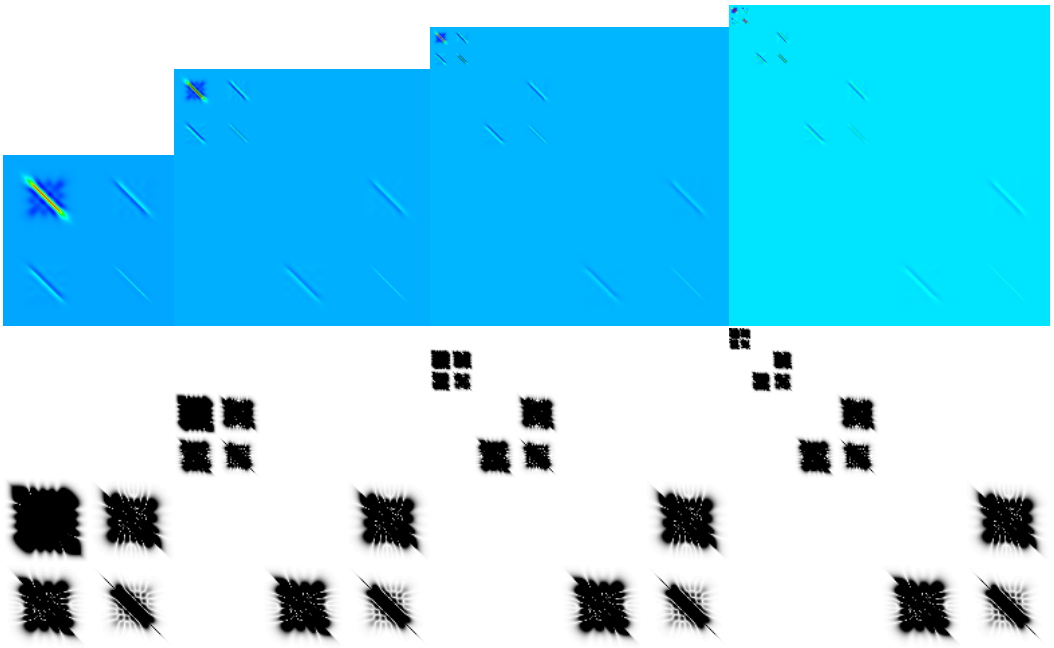


Figure 17: Haar Non-Standard Transform of χ_0

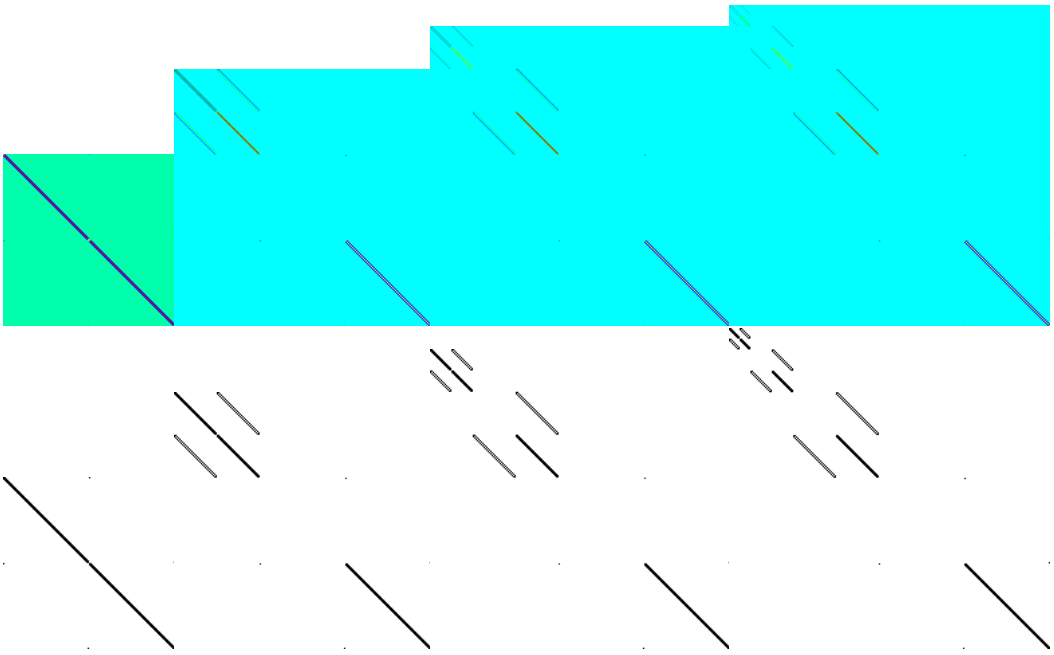


Figure 18: Haar Non-Standard Transform of $\frac{d^2}{dz^2}$

The non-standard Haar transform is quite good, especially for the $\frac{d^2}{dz^2}$ matrix, which remains diagonal and reasonably sparse. The χ_0 behaves similarly, though is still a bit dense.

4.5 Daubechies' Wavelets

Standard

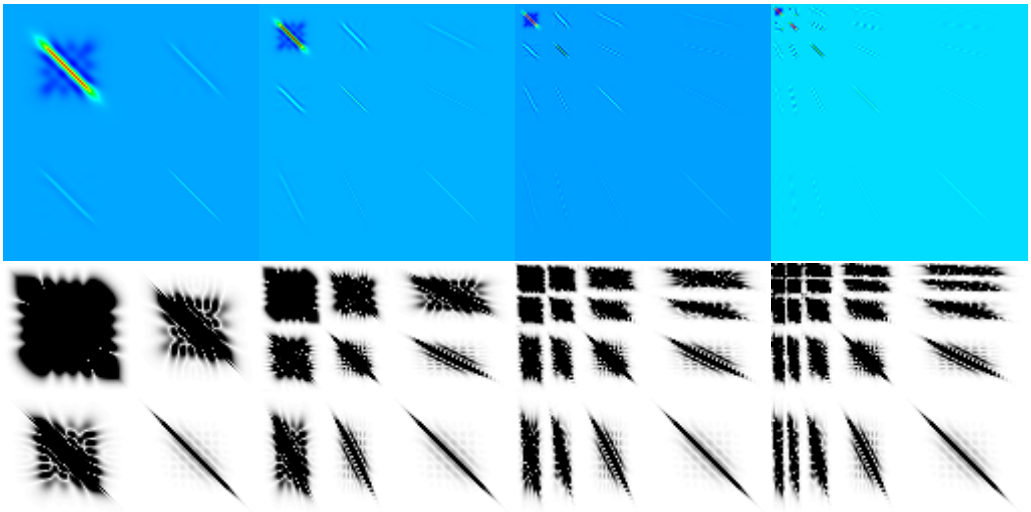


Figure 19: Daub4 Standard Transform of χ_0

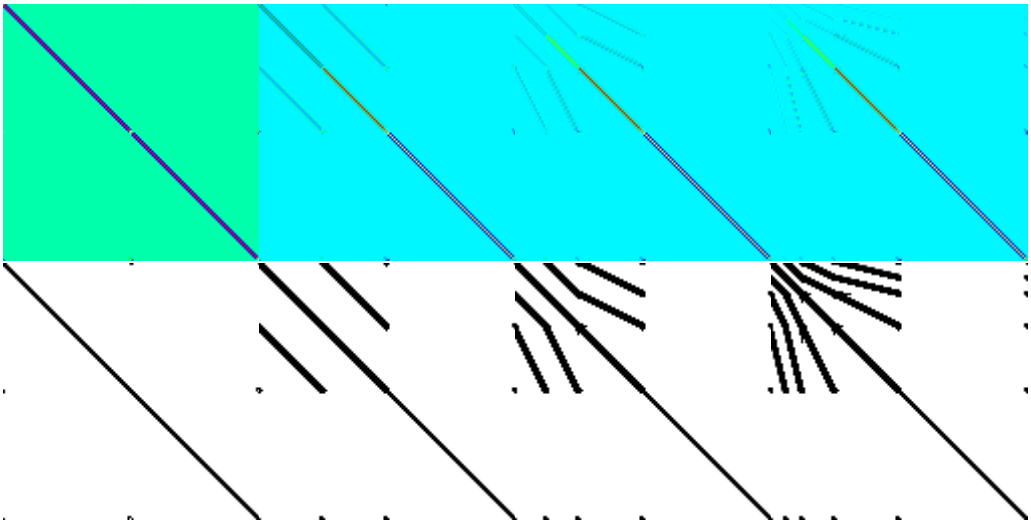


Figure 20: Daub4 Standard Transform of $\frac{d^2}{dz^2}$

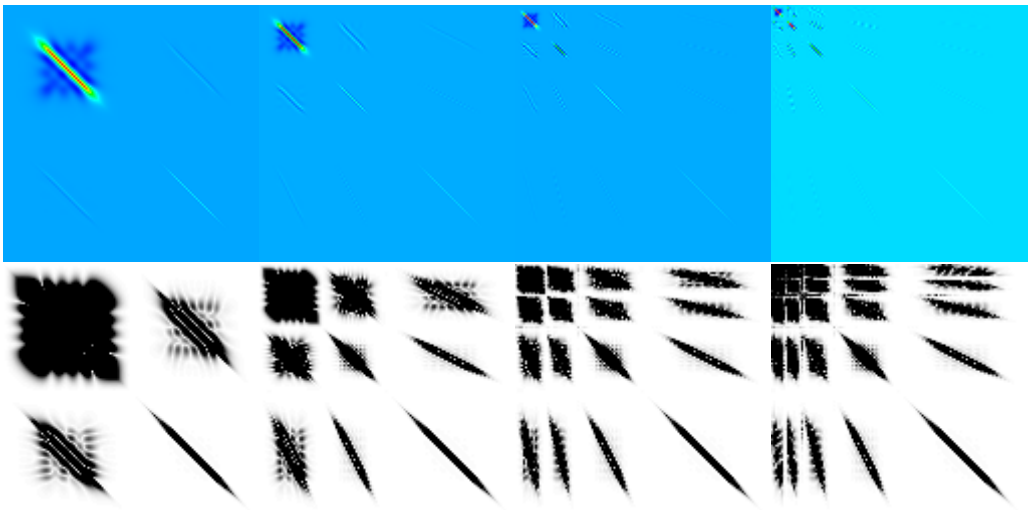


Figure 21: Daub6 Standard Transform of χ_0

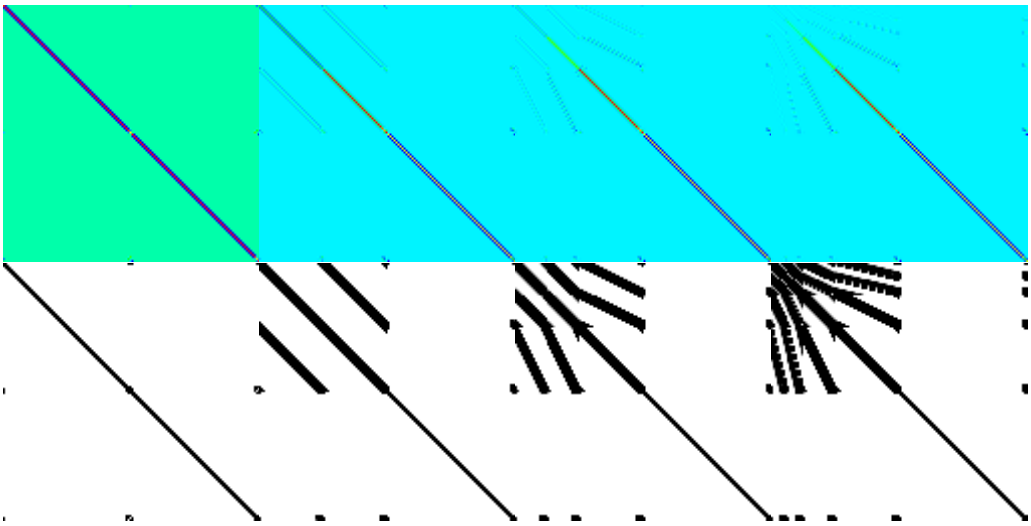


Figure 22: Daub6 Standard Transform of $\frac{d^2}{dz^2}$

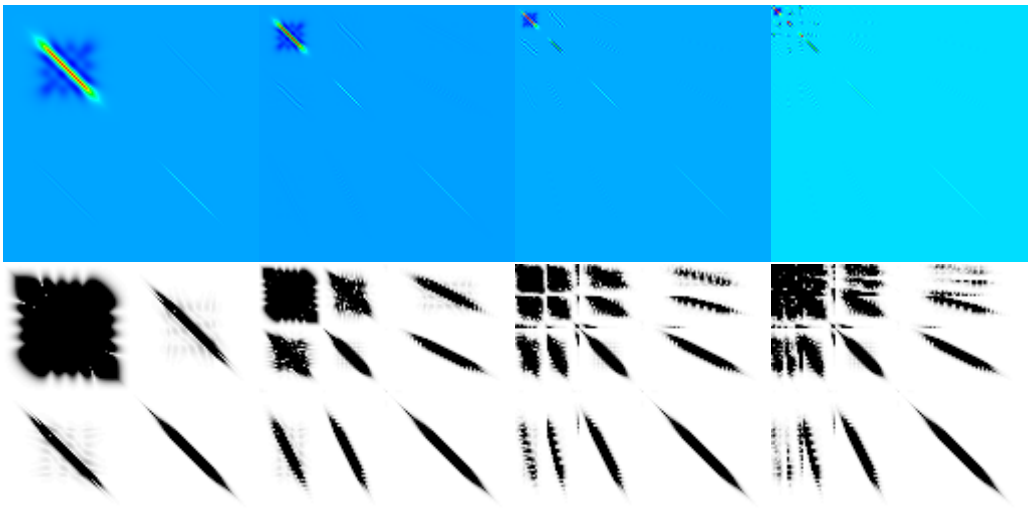


Figure 23: Daub10 Standard Transform of χ_0

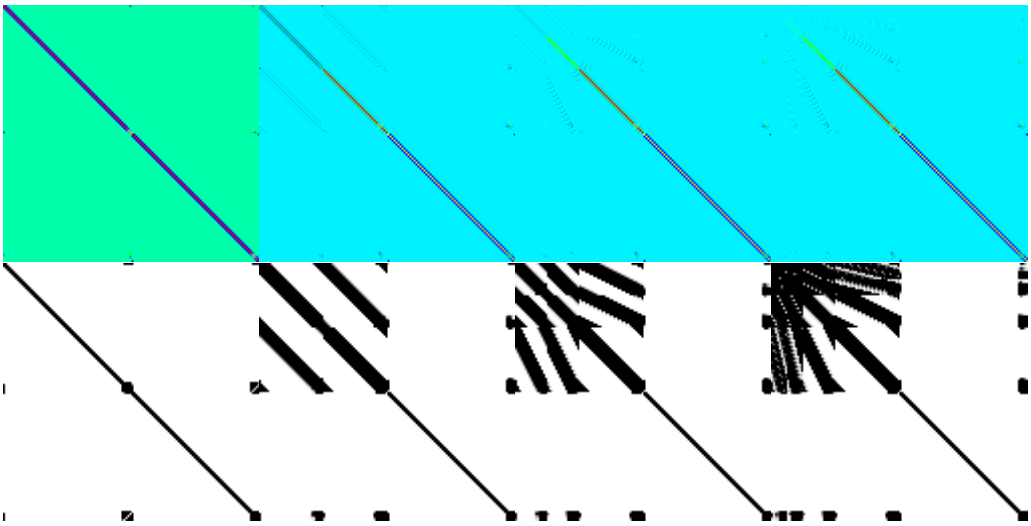


Figure 24: Daub10 Standard Transform of $\frac{d^2}{dz^2}$

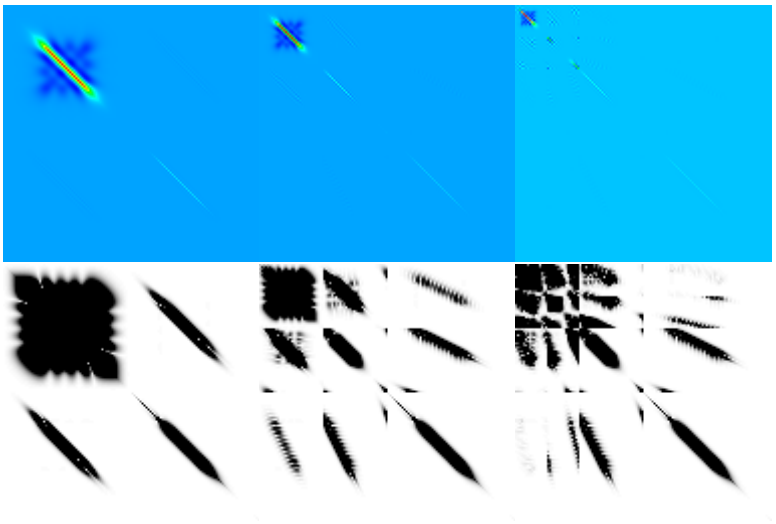


Figure 25: Daub20 Standard Transform of χ_0

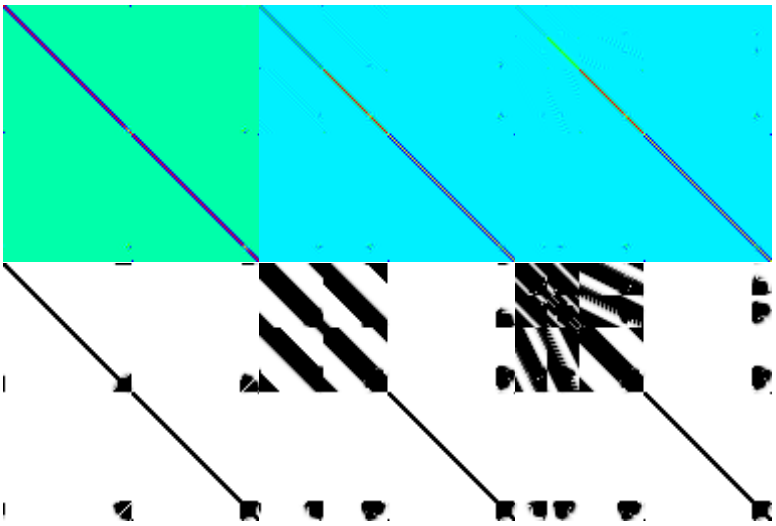


Figure 26: Daub20 Standard Transform of $\frac{d^2}{dz^2}$

We begin to see how the χ_0 tends to prefer transforms with wider support, but the $\frac{d^2}{dz^2}$ prefers shorter support. As the length of the Daubechies wavelet increases the χ_0 transforms become more sparse, but at the same time the $\frac{d^2}{dz^2}$ transforms become more dense. The trick will be to find a good compromise between the two.

Packet

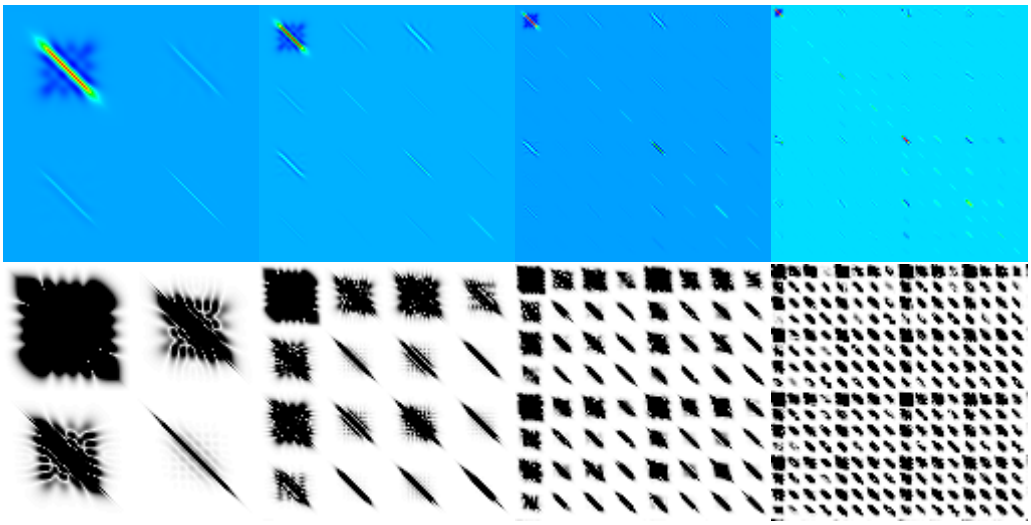


Figure 27: Daub4 Packet Transform of χ_0

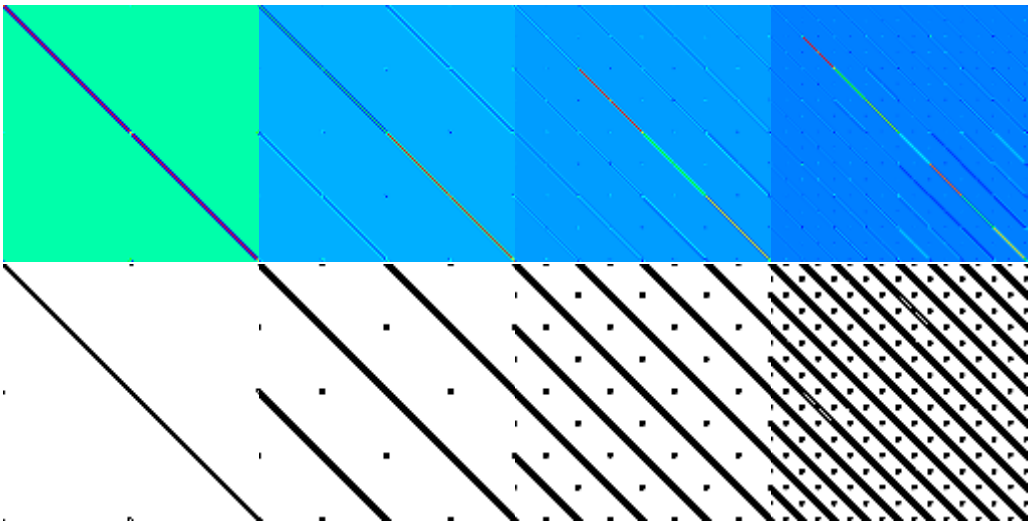


Figure 28: Daub4 Packet Transform of $\frac{d^2}{dz^2}$

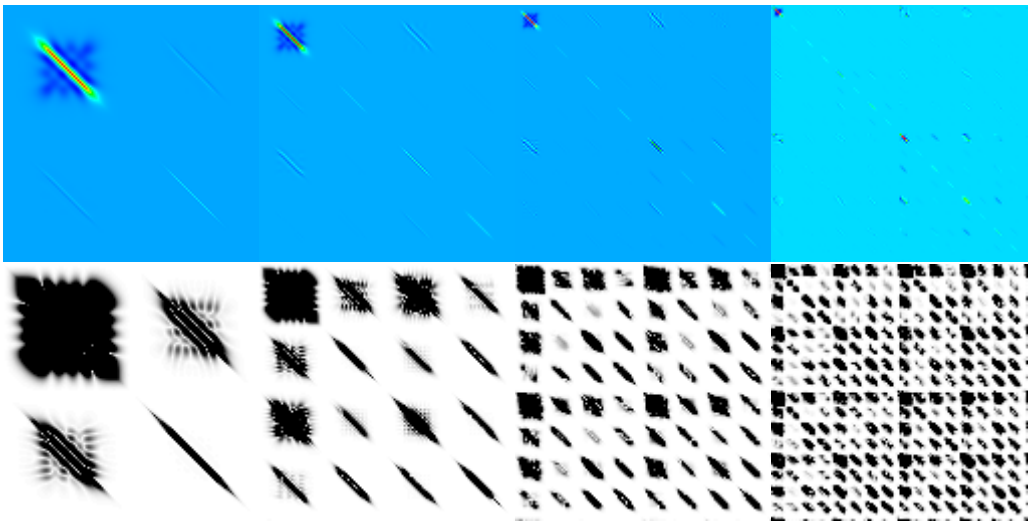


Figure 29: Daub6 Packet Transform of χ_0

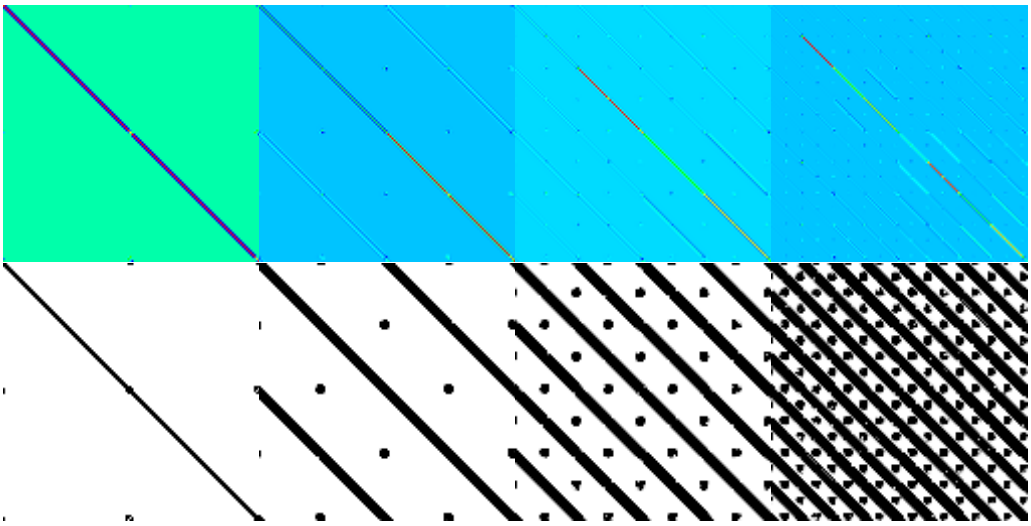


Figure 30: Daub6 Packet Transform of $\frac{d^2}{dz^2}$

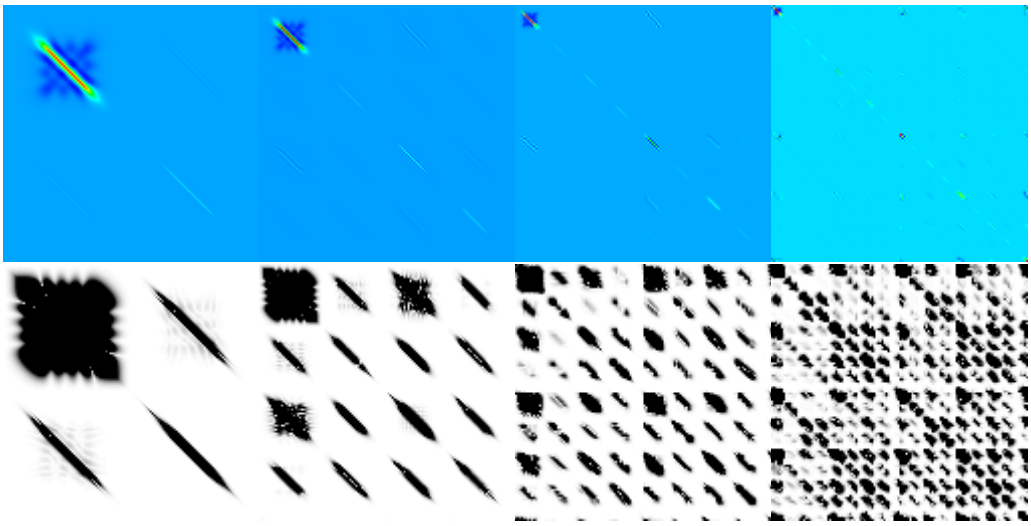


Figure 31: Daub10 Packet Transform of χ_0

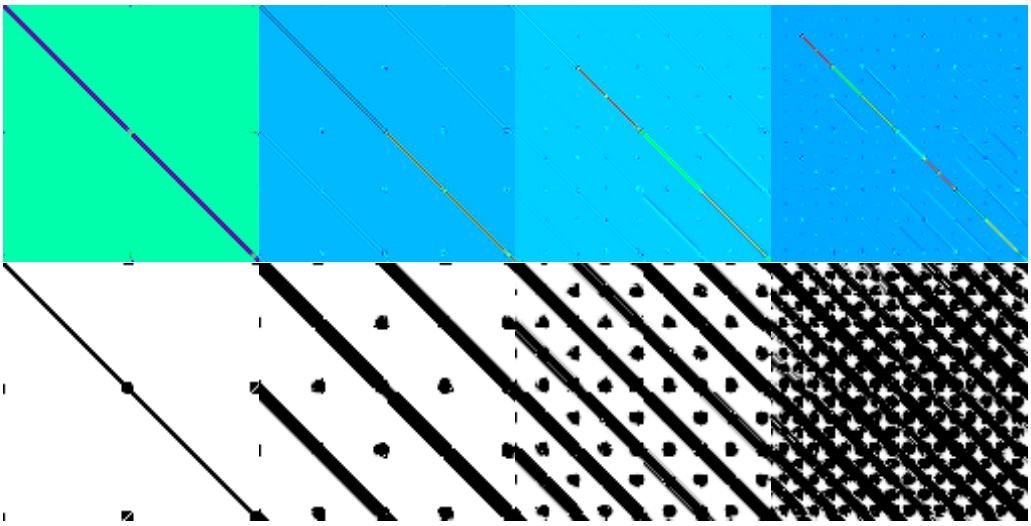


Figure 32: Daub10 Packet Transform of $\frac{d^2}{dz^2}$

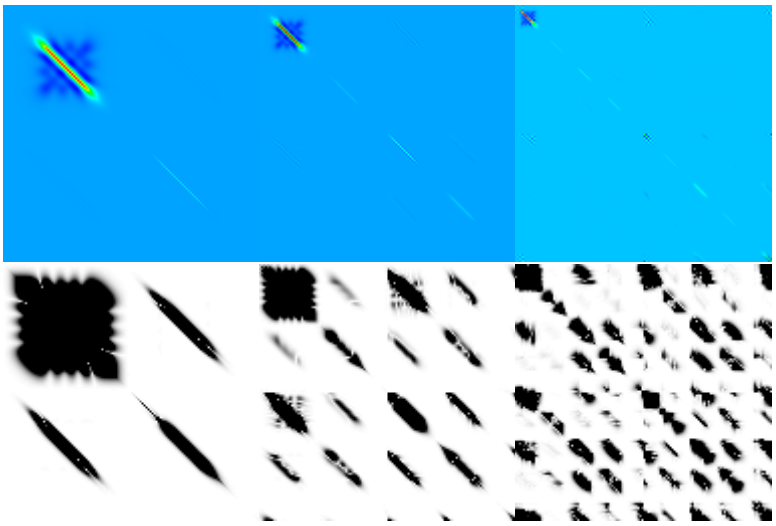


Figure 33: Daub20 Packet Transform of χ_0

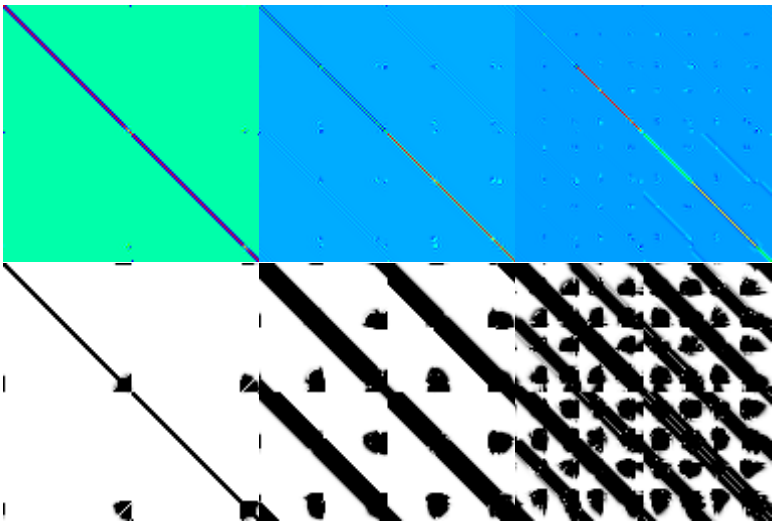


Figure 34: Daub20 Packet Transform of $\frac{d^2}{dz^2}$

Though it makes for some interesting patterns in the significance maps, the packet transform here seems quite inappropriate for what we want to achieve. All packet transforms thus far have taken something reasonably sparse and turned it into something very dense. This is the exact opposite of what we are aiming for.

Non-Standard

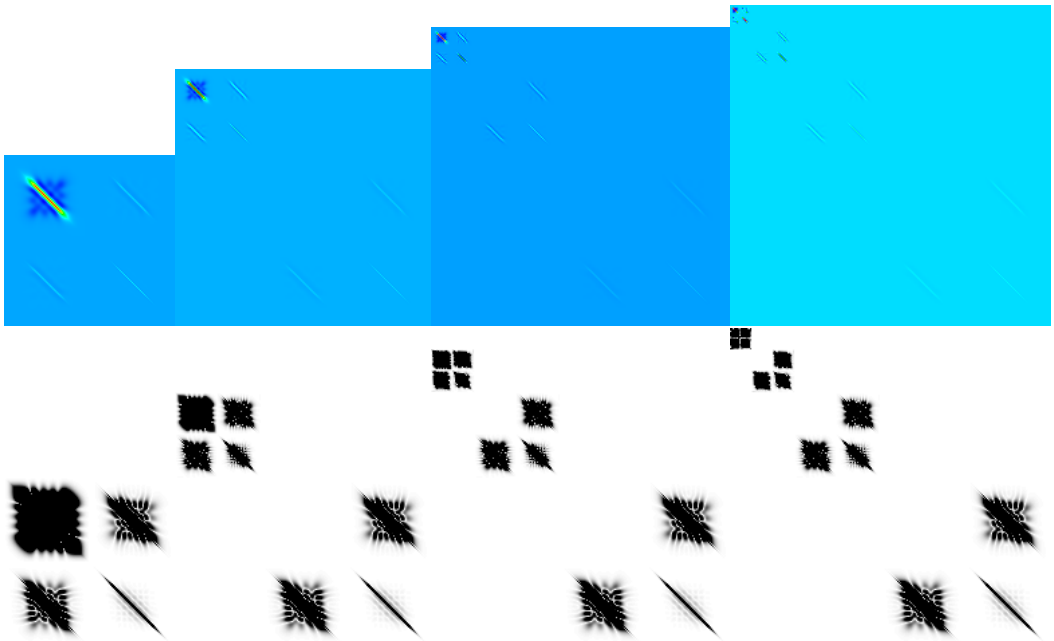


Figure 35: Daub4 Non-Standard Transform of χ_0

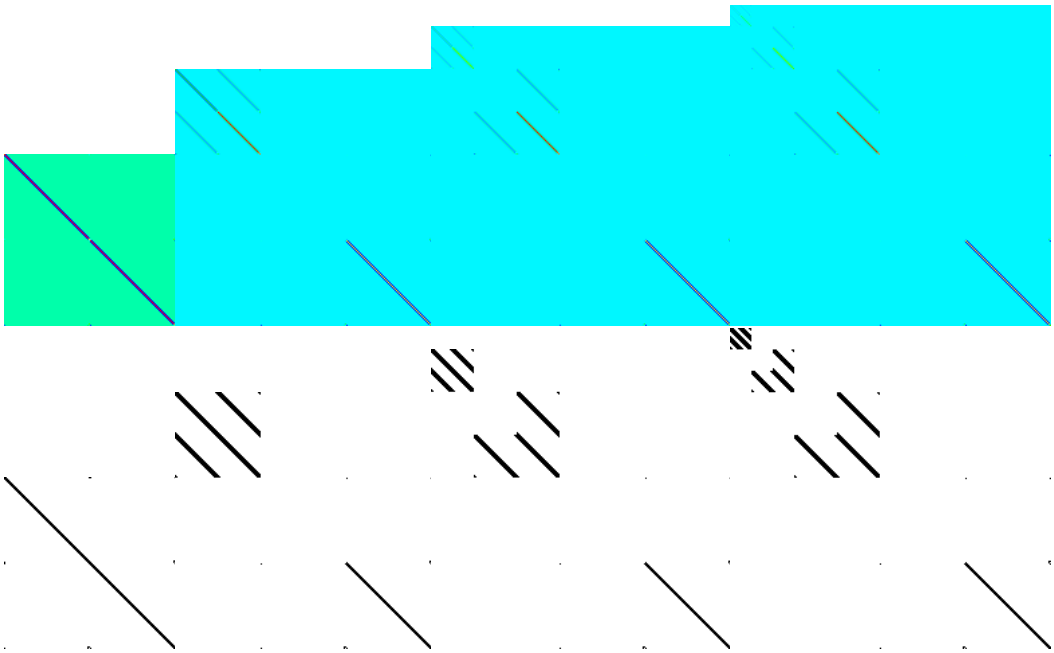


Figure 36: Daub4 Non-Standard Transform of $\frac{d^2}{dz^2}$

Figure 37: Daub6 Non-Standard Transform of χ_0

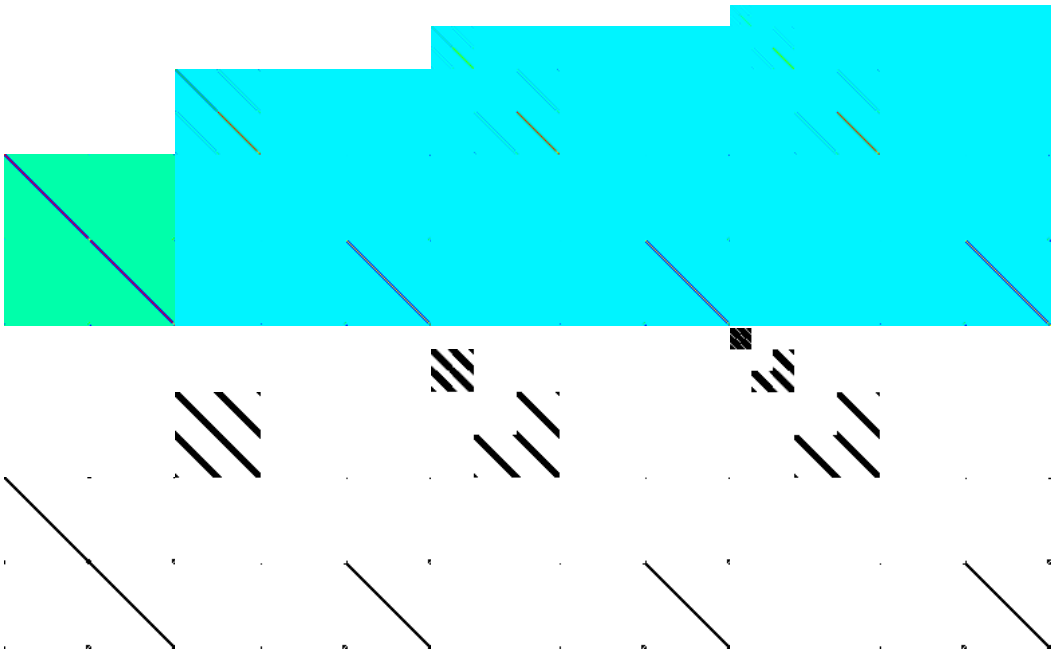


Figure 38: Daub6 Non-Standard Transform of $\frac{d^2}{dz^2}$

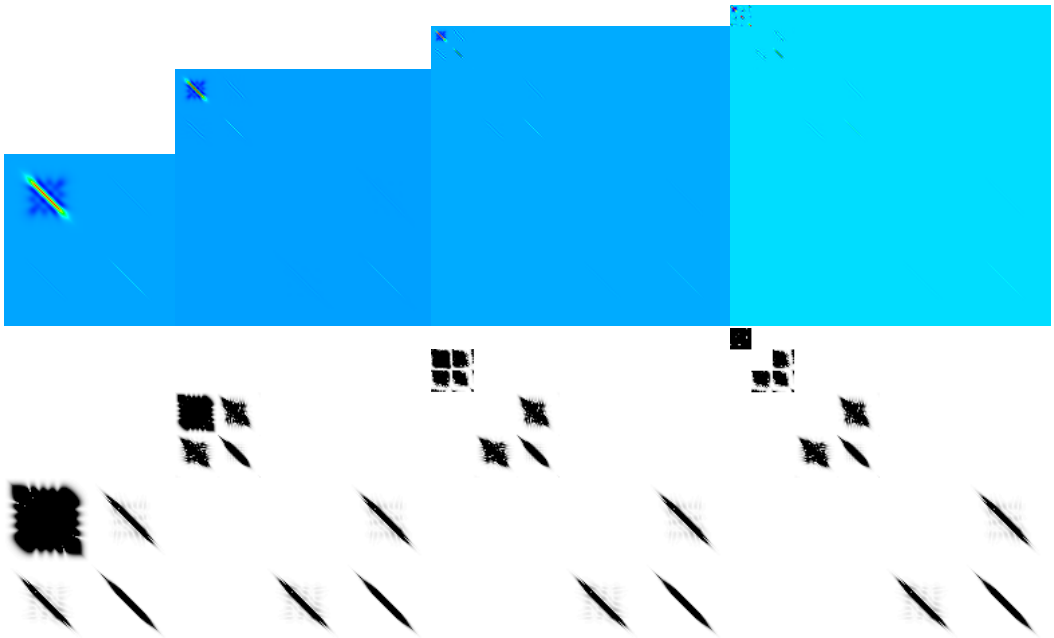


Figure 39: Daub10 Non-Standard Transform of χ_0

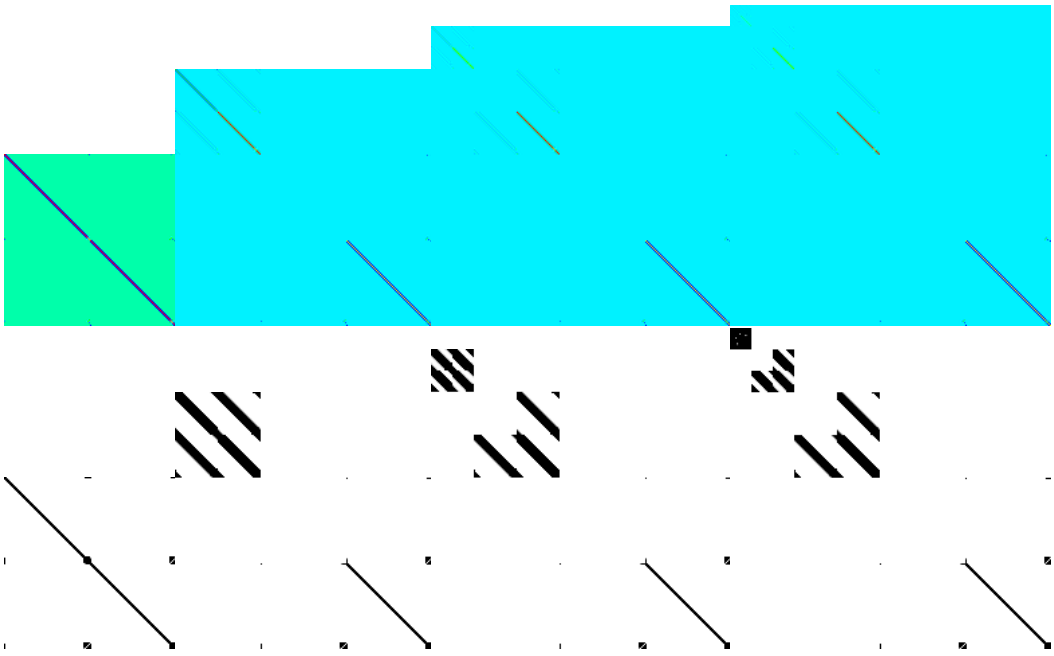


Figure 40: Daub10 Non-Standard Transform of $\frac{d^2}{dz^2}$

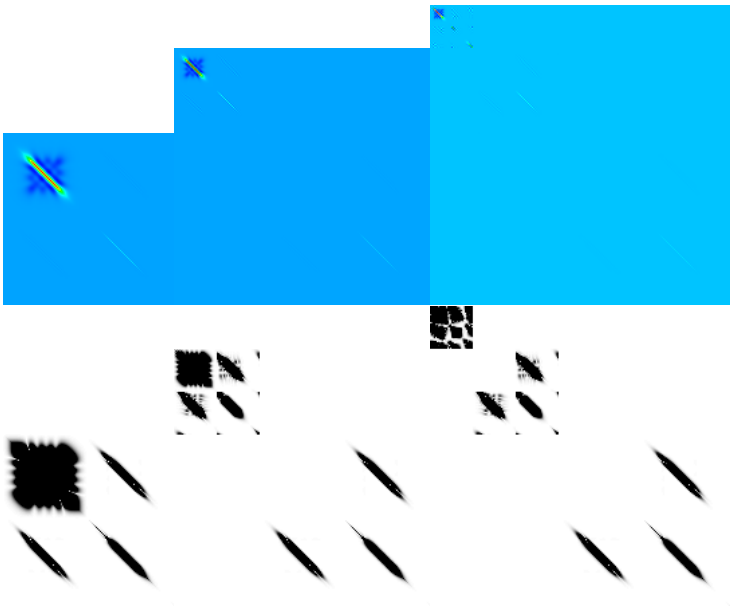


Figure 41: Daub20 Non-Standard Transform of χ_0

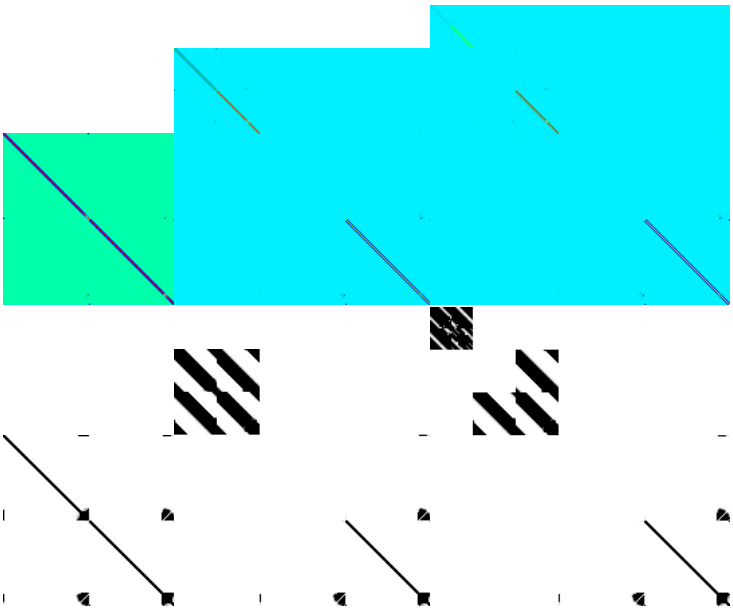


Figure 42: Daub20 Non-Standard Transform of $\frac{d^2}{dz^2}$

The non-standard transform, on the other hand, is looking promising. The matrices remain diagonal and, while the $\frac{d^2}{dz^2}$ does become more dense particularly with long support, the χ_0 becomes quite less dense.

4.6 Coiflets

Standard

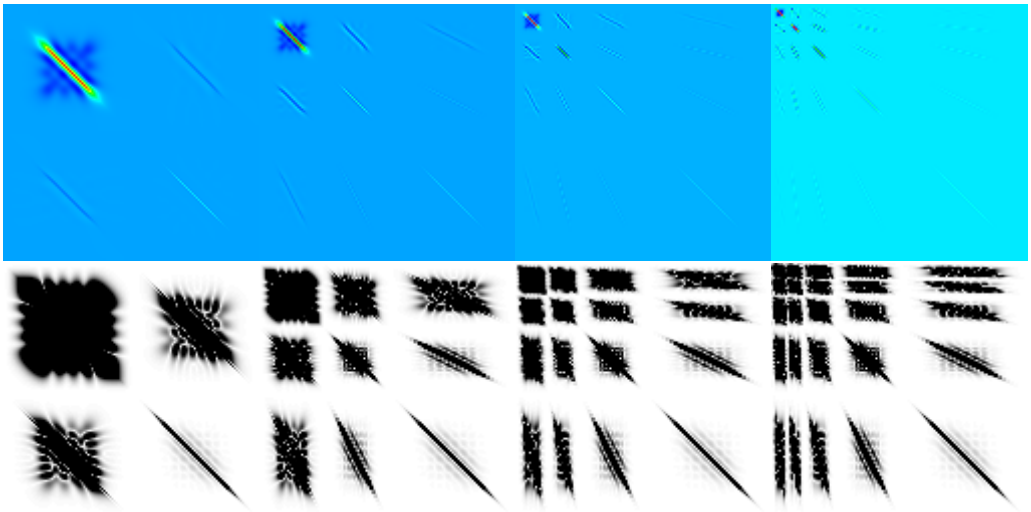


Figure 43: Coif6 Standard Transform of χ_0

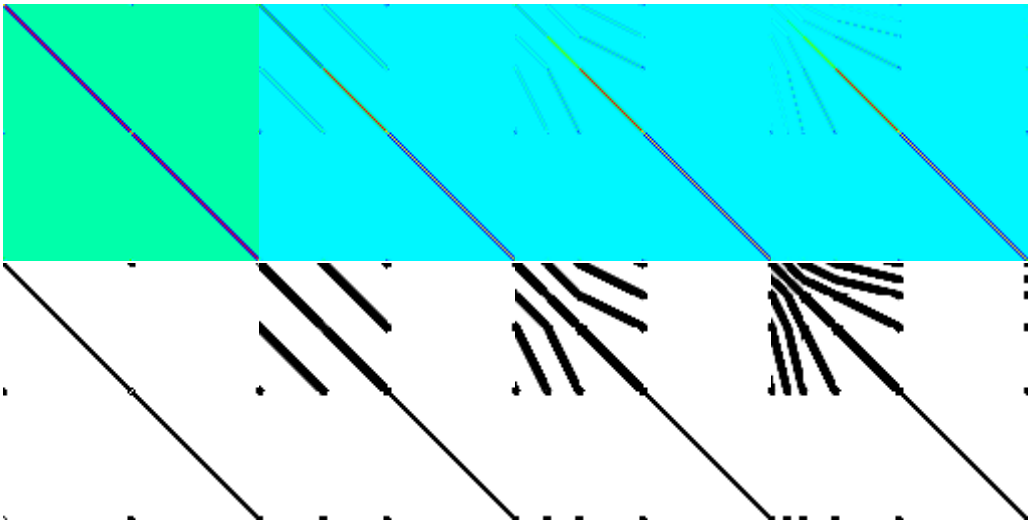


Figure 44: Coif6 Standard Transform of $\frac{d^2}{dz^2}$

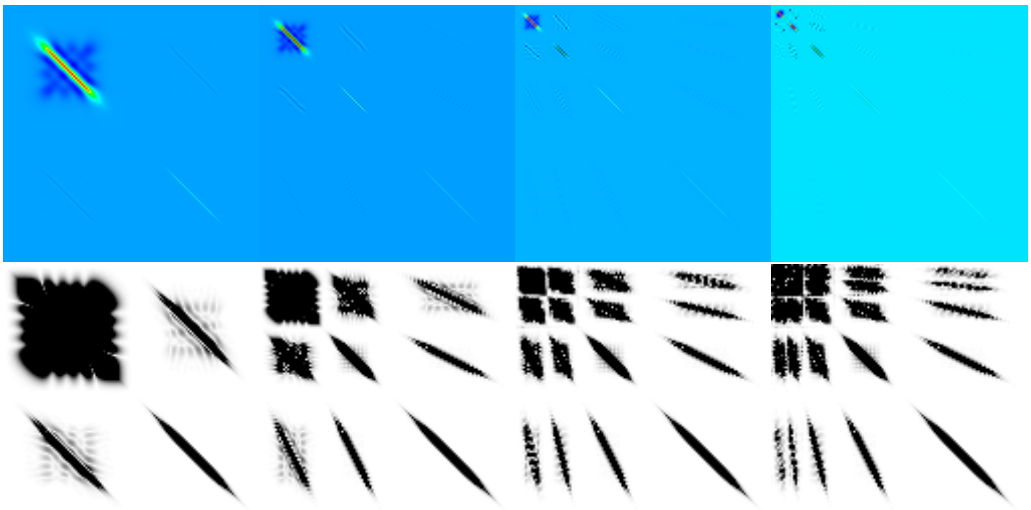


Figure 45: Coif12 Standard Transform of χ_0

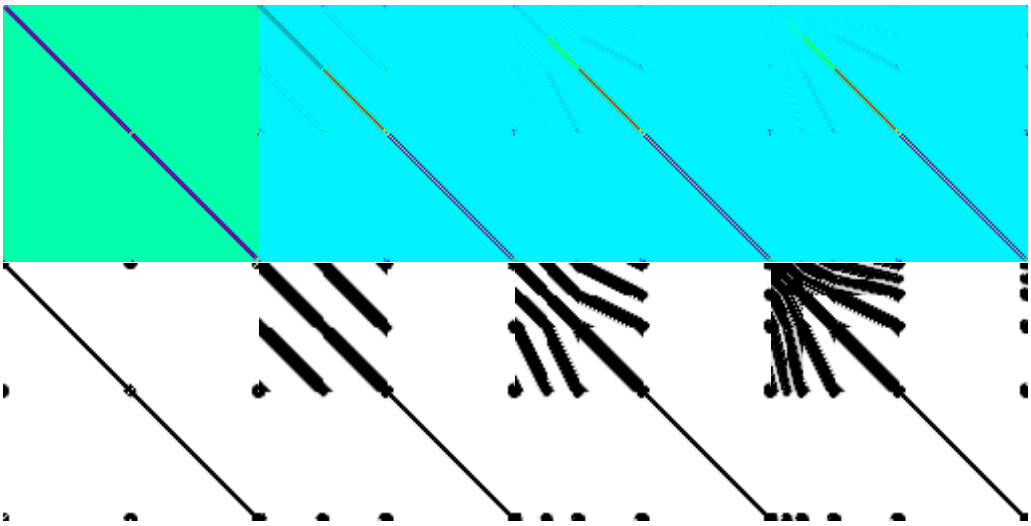


Figure 46: Coif12 Standard Transform of $\frac{d^2}{dz^2}$

The standard transform seems to move significant values towards the top and the left, and spreads them around. Because of the coupled nature of resolutions in the standard transform this is to be expected (see Section 3.5). This does make such a transform less desirable.

Packet

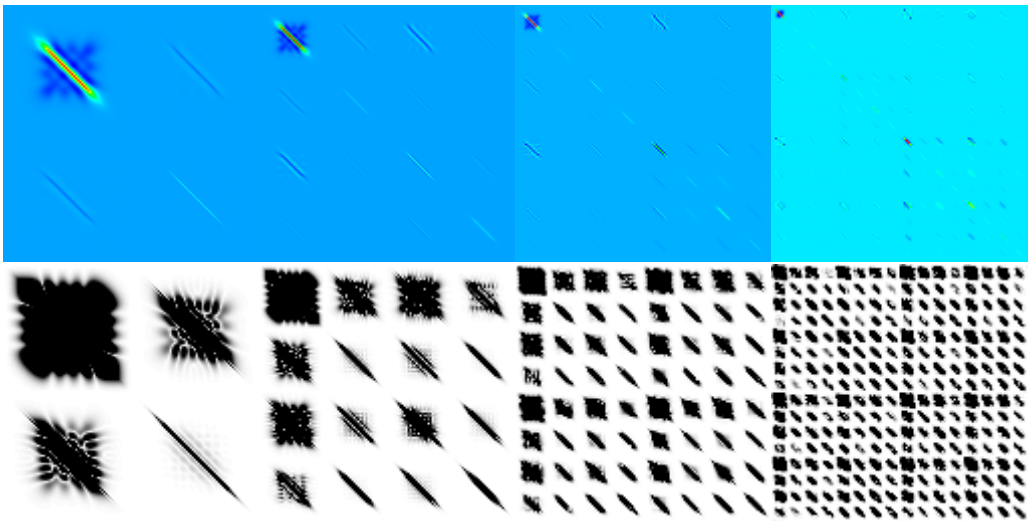


Figure 47: Coif6 Packet Transform of χ_0

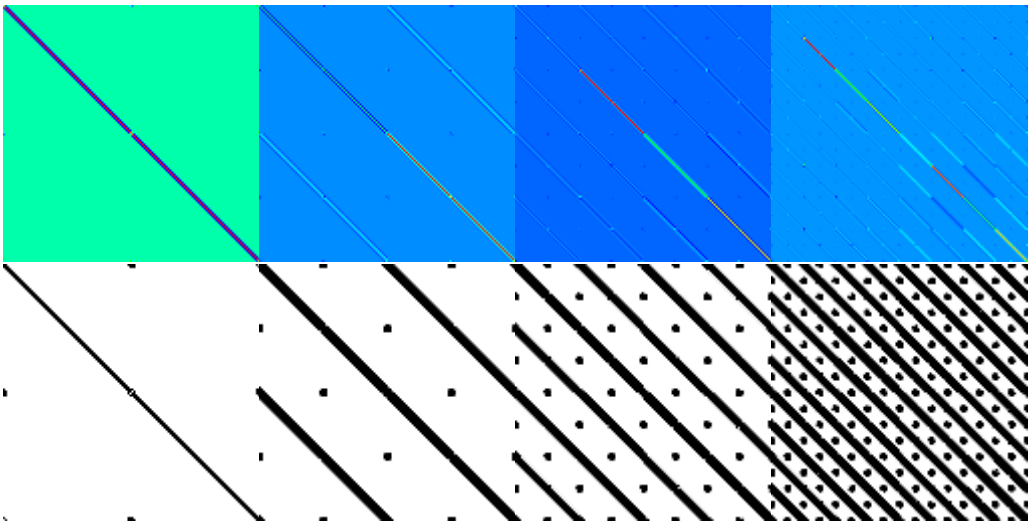


Figure 48: Coif6 Packet Transform of $\frac{d^2}{dz^2}$

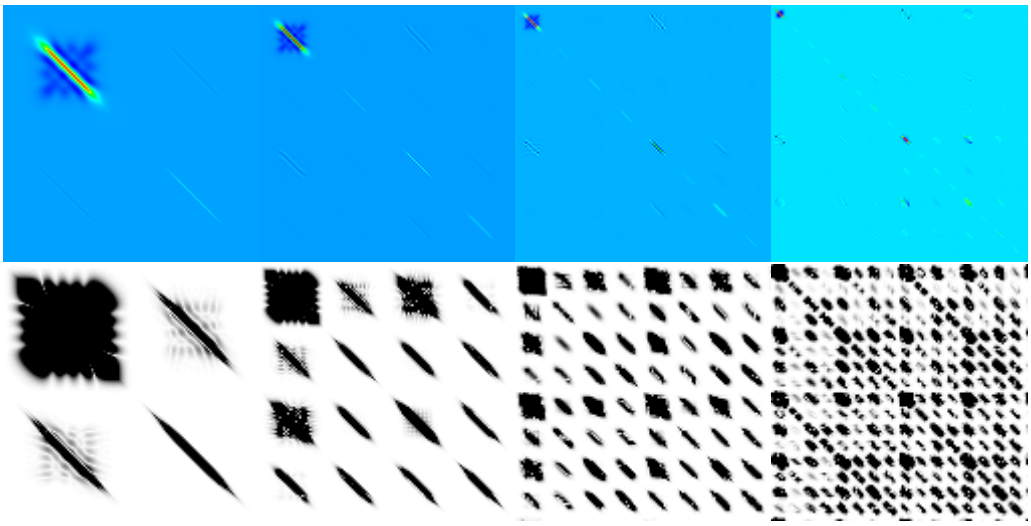


Figure 49: Coif12 Packet Transform of χ_0

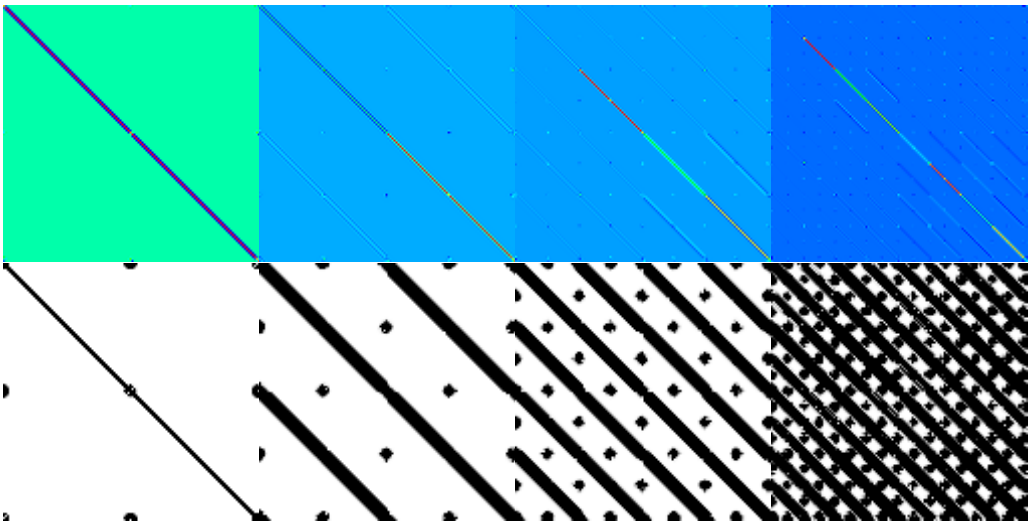


Figure 50: Coif12 Packet Transform of $\frac{d^2}{dz^2}$

Packet transforms seem to send significant values all over the resulting matrix. Packet transforms would be a very poor approach to this problem.

Non-Standard

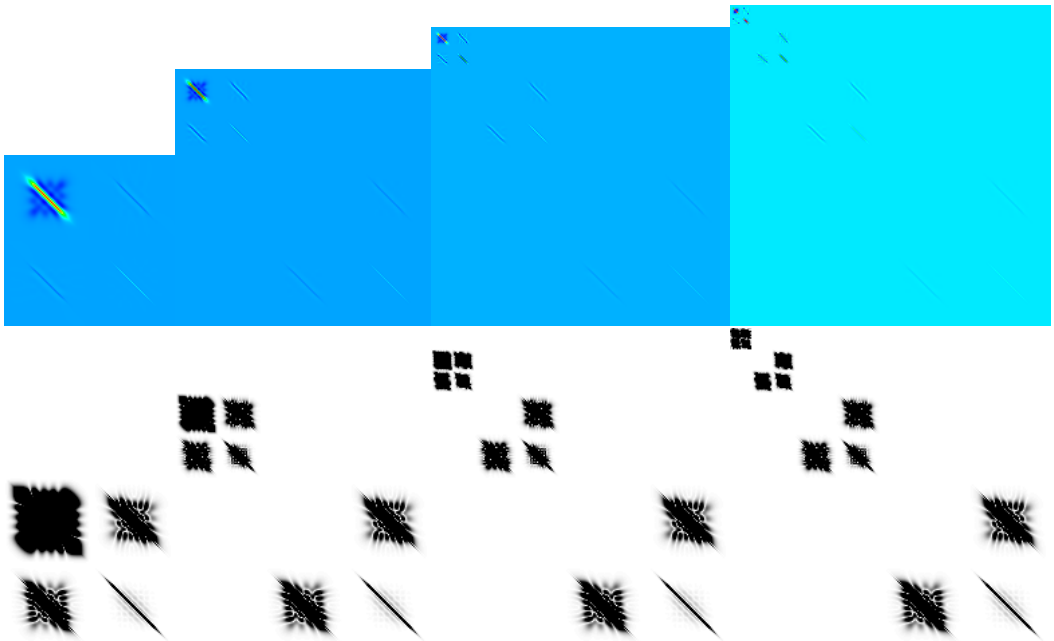


Figure 51: Coif6 Non-Standard Transform of χ_0

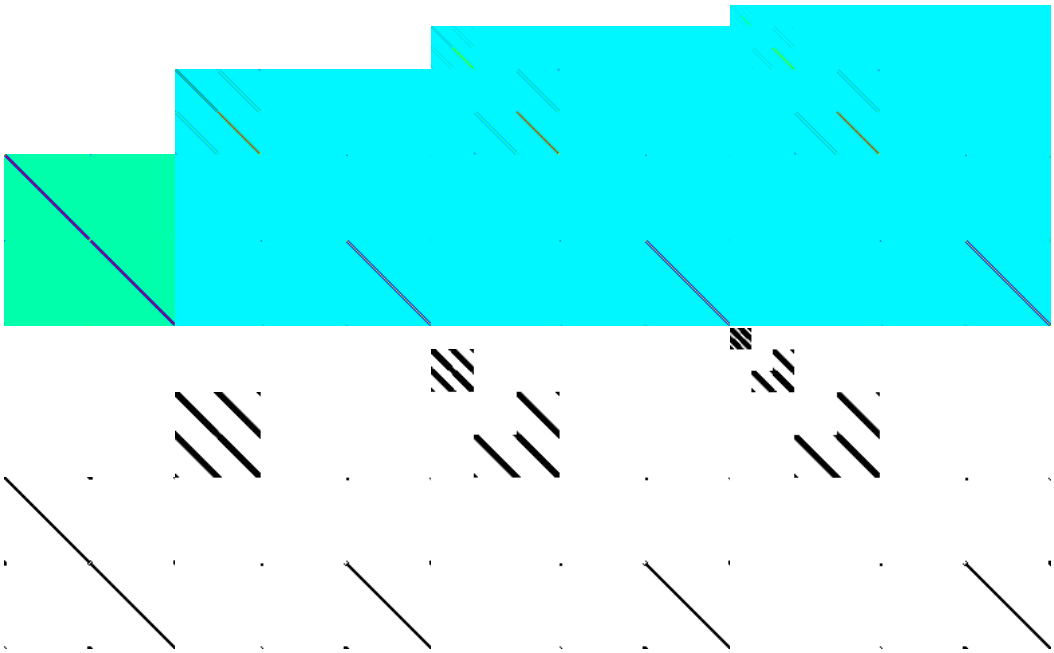


Figure 52: Coif6 Non-Standard Transform of $\frac{d^2}{dz^2}$

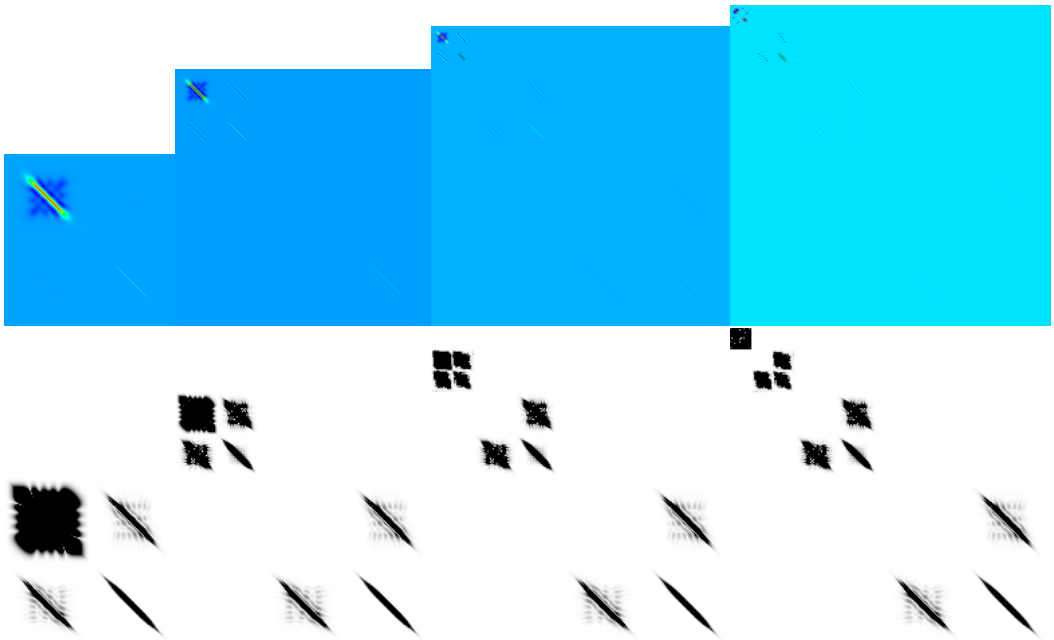


Figure 53: Coif12 Non-Standard Transform of χ_0

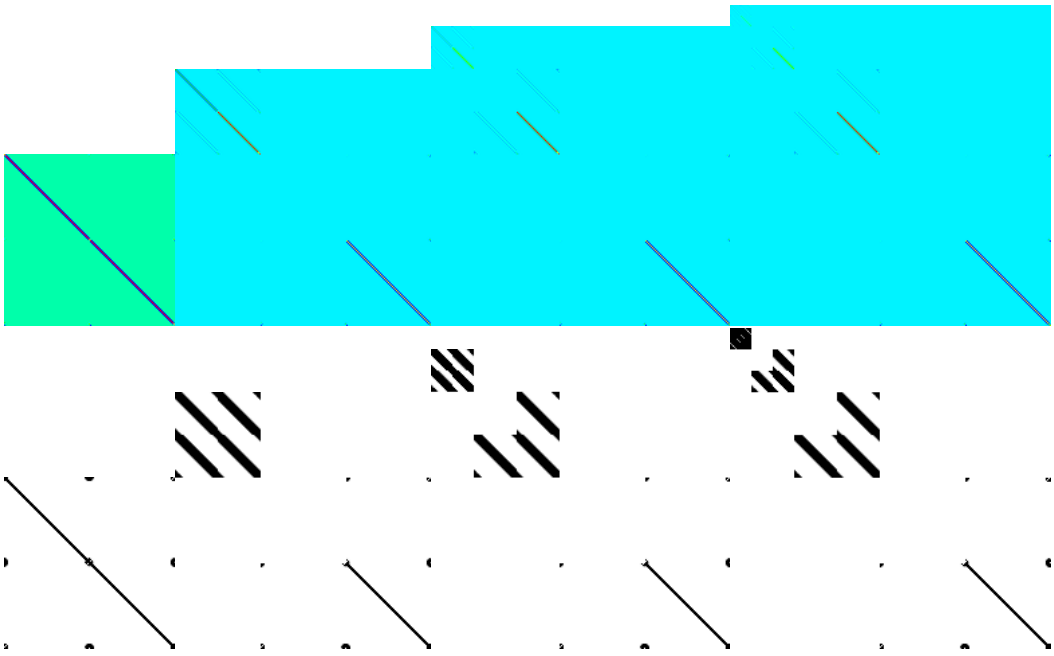


Figure 54: Coif12 Non-Standard Transform of $\frac{d^2}{dz^2}$

Again the non-standard transform looks promising.

4.7 Symmlets

Standard

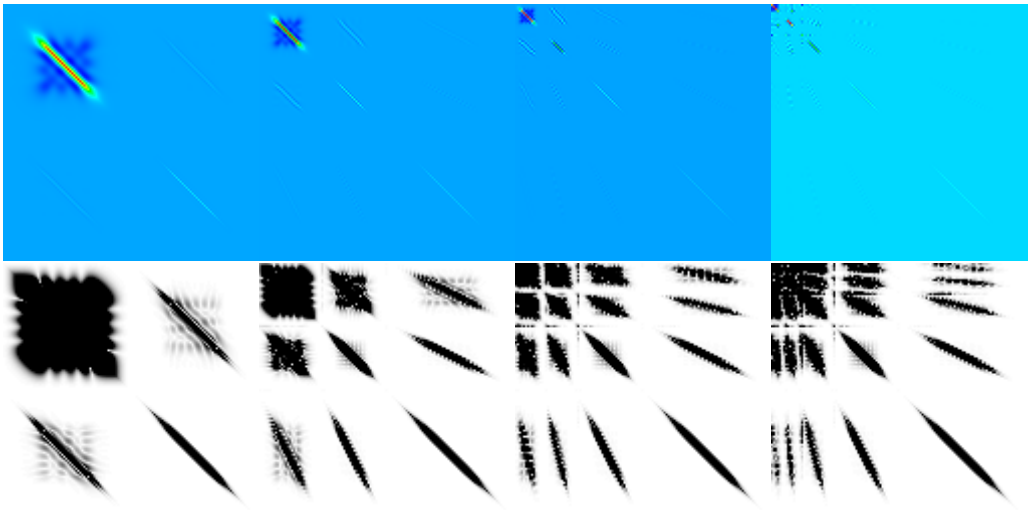


Figure 55: Sym4 Standard Transform of χ_0

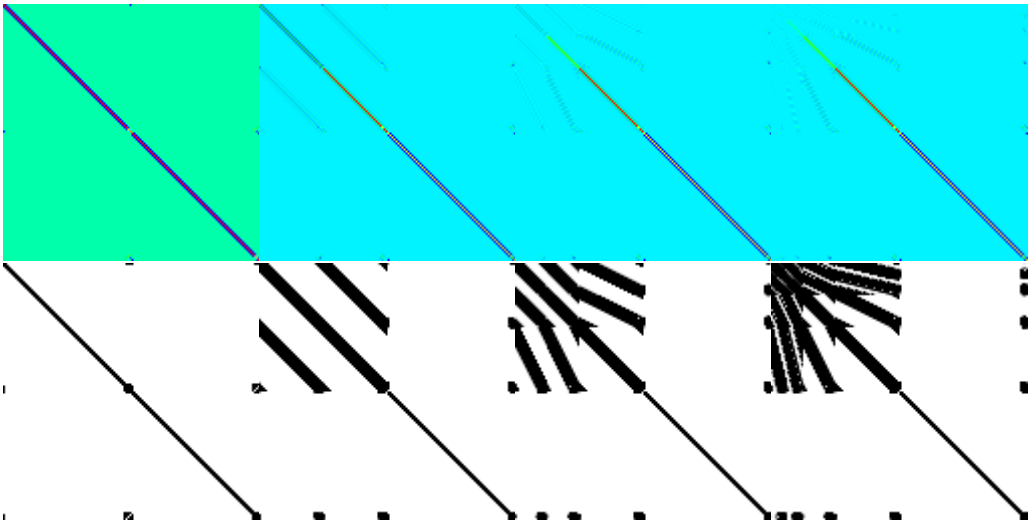


Figure 56: Sym4 Standard Transform of $\frac{d^2}{dz^2}$

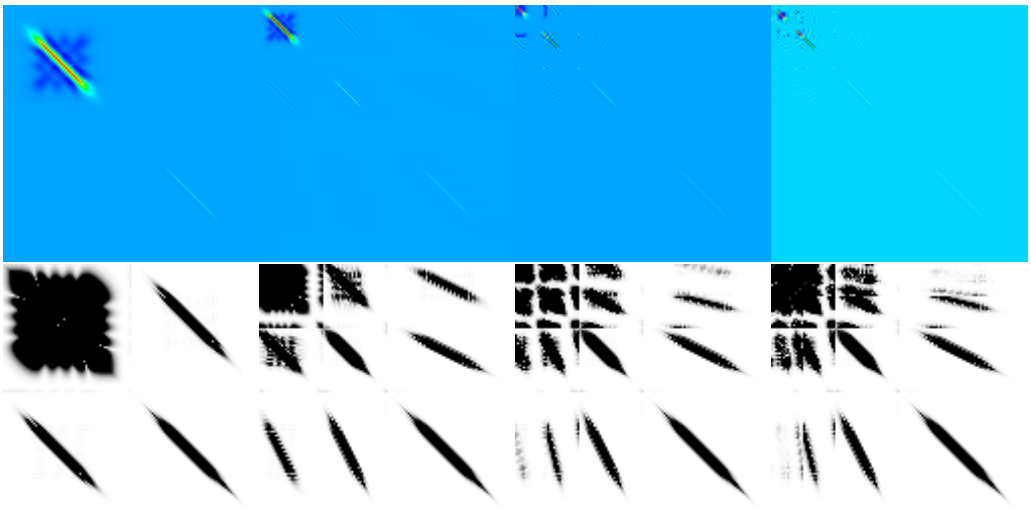


Figure 57: Sym7 Standard Transform of χ_0

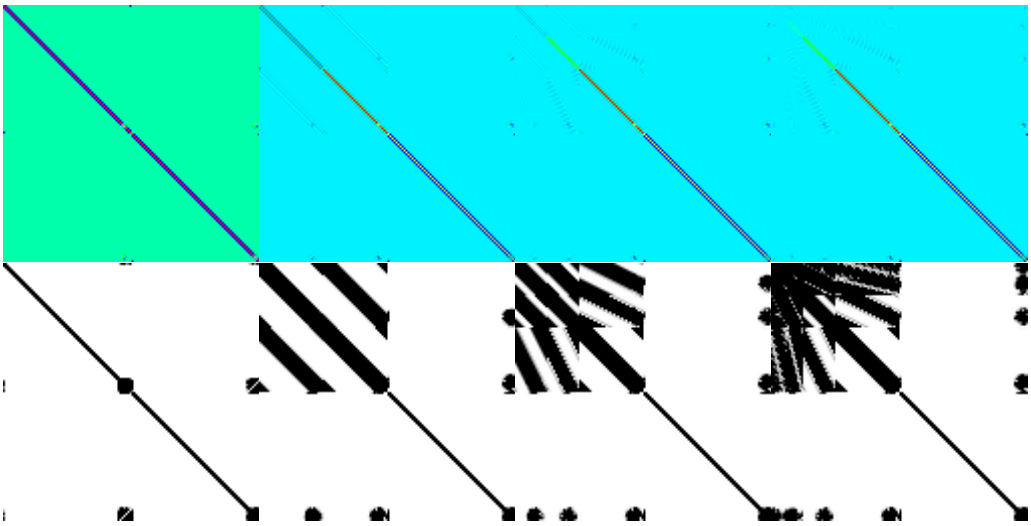


Figure 58: Sym7 Standard Transform of $\frac{d^2}{dz^2}$

The Sym4 seems to do quite well on the χ_0 signal, slightly better than the Sym7. The $\frac{d^2}{dz^2}$ matrix continues to transform poorly.

Packet

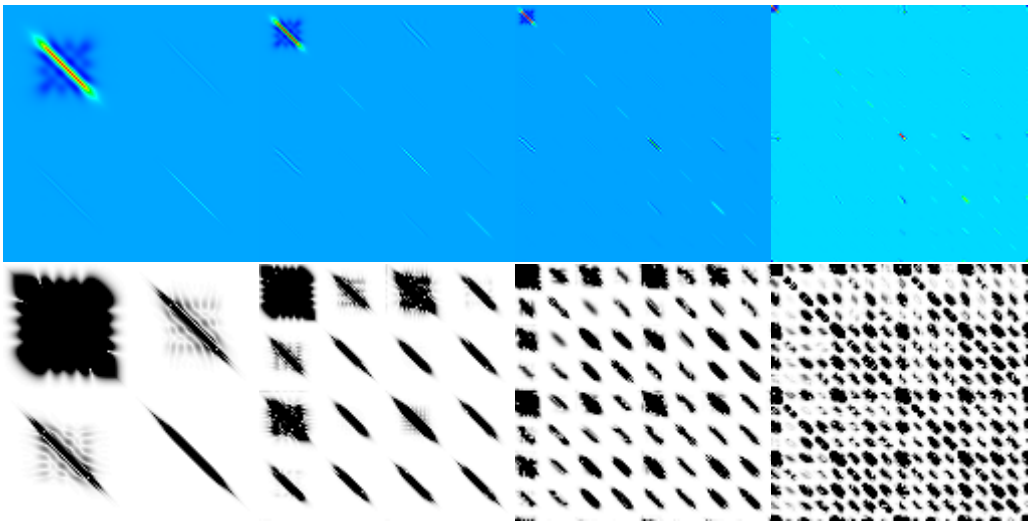


Figure 59: Sym4 Packet Transform of χ_0

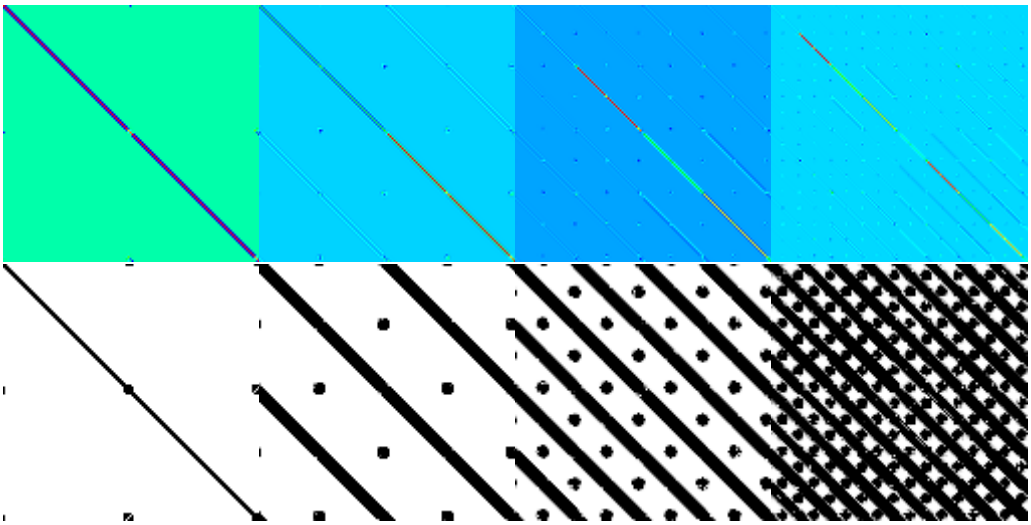


Figure 60: Sym4 Packet Transform of $\frac{d^2}{dz^2}$

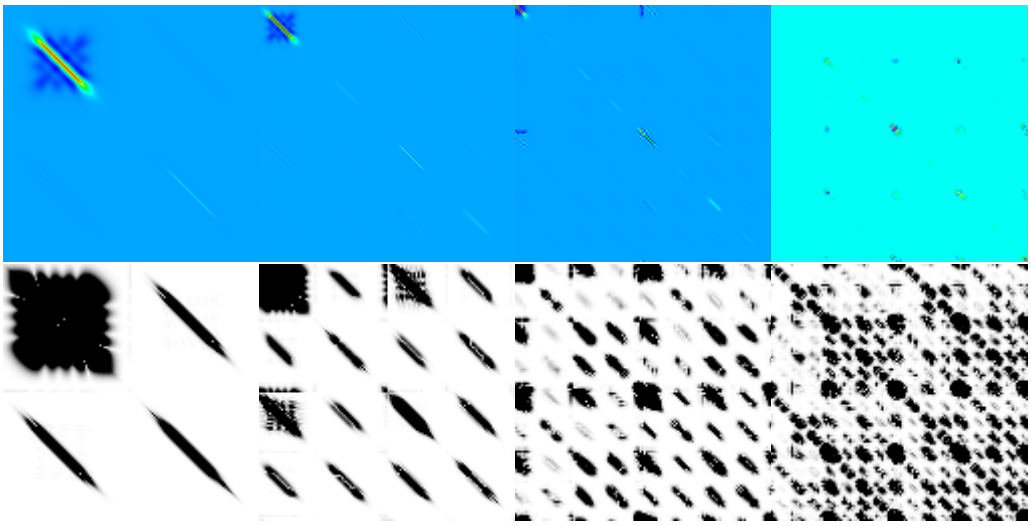


Figure 61: Sym7 Packet Transform of χ_0

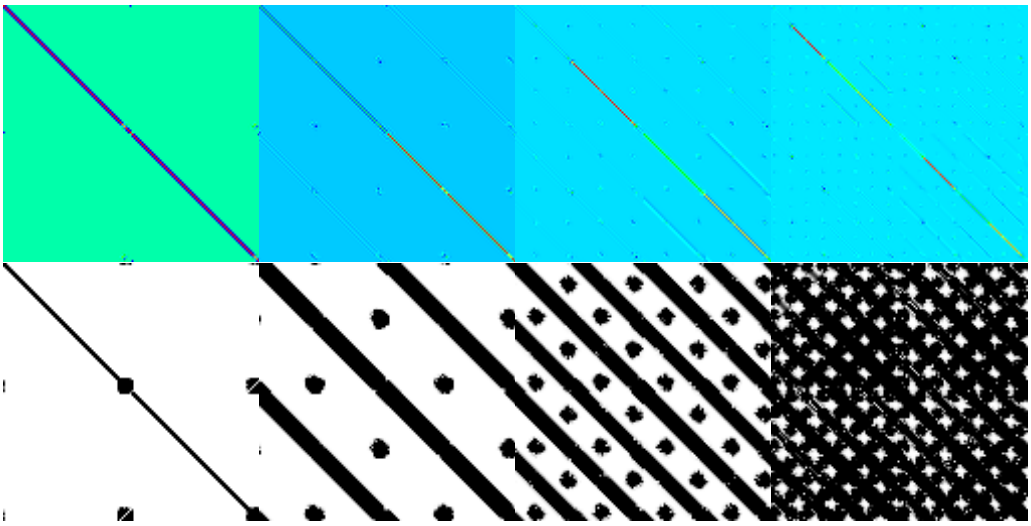


Figure 62: Sym7 Packet Transform of $\frac{d^2}{dz^2}$

The symmlet packet transforms are not so bad on the χ_0 matrix, compared to other packet transforms, but the transforms make the $\frac{d^2}{dz^2}$ matrix so dense that this transform should be rejected.

Non-Standard

Figure 63: Sym4 Non-Standard Transform of χ_0

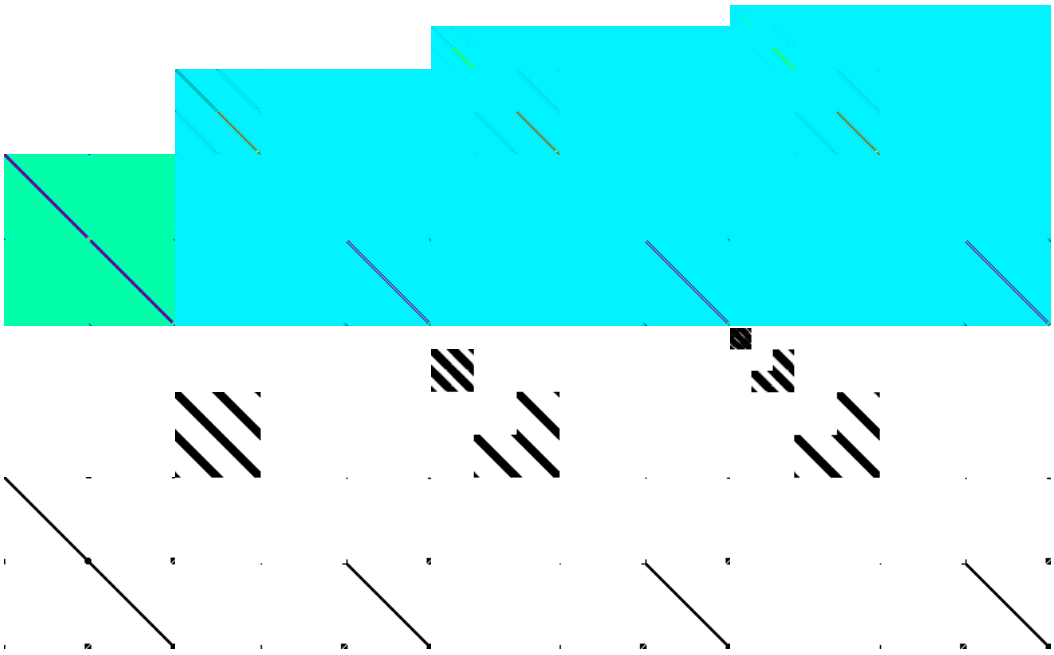


Figure 64: Sym4 Non-Standard Transform of $\frac{d^2}{dz^2}$

Figure 65: Sym7 Non-Standard Transform of χ_0

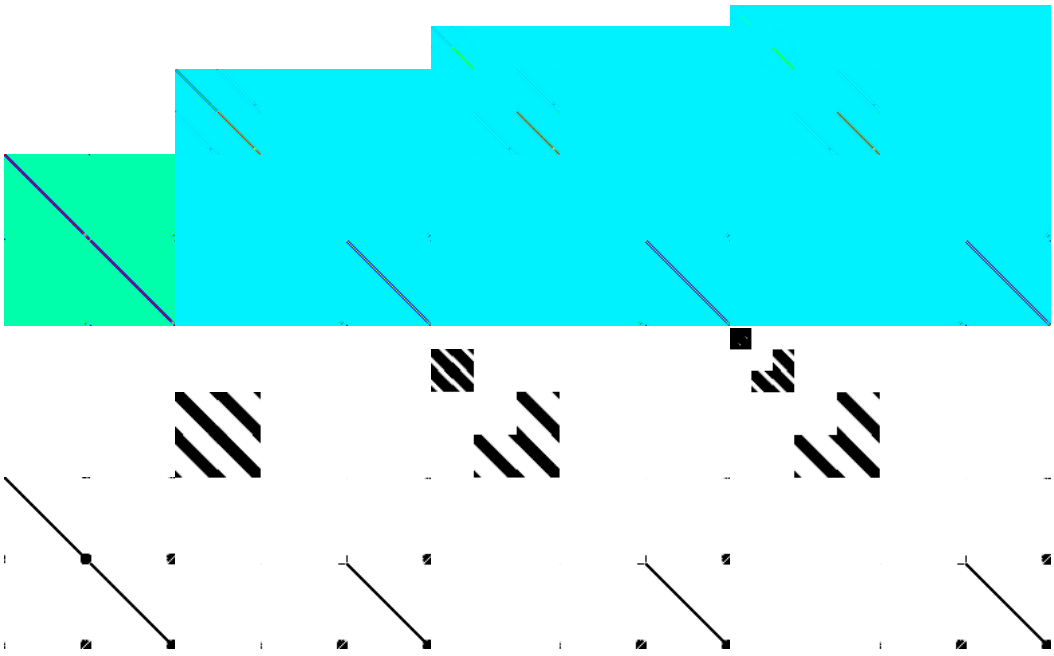


Figure 66: Sym7 Non-Standard Transform of $\frac{d^2}{dz^2}$

As expected, non-standard transforms are the best of this bunch.

4.8 Densities of Transforms

It is difficult to compare each of the figures listed above. To aid comparison, graphs of measured density (see Section 4.2) for each wavelet type as the resolution level increases are listed below.

Standard

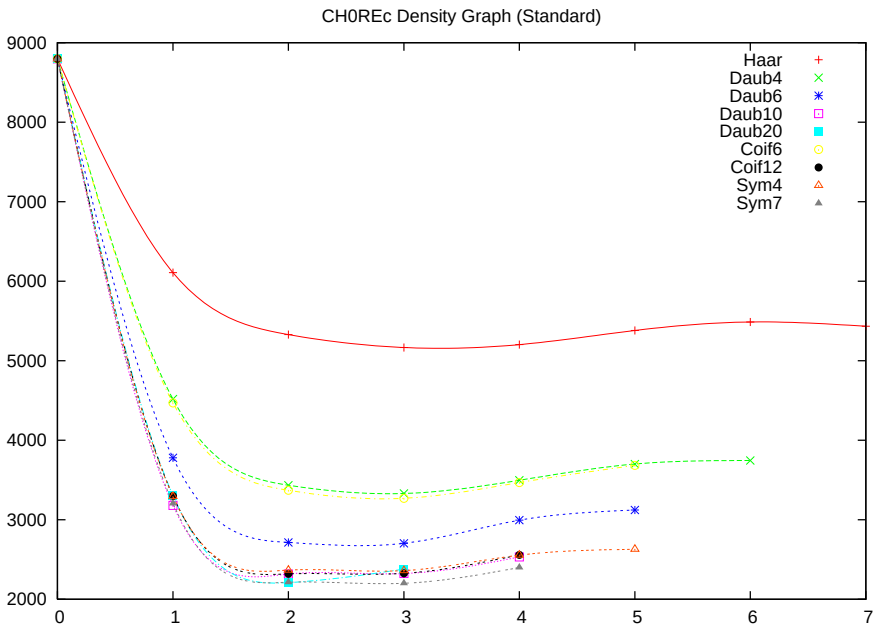


Figure 67: Density Graph for Different Resolution Standard Transforms of χ_0

This quantitatively shows how wavelets of longer support generally better suit the χ_0 matrix. You can see here that the best standard transform is the Sym7, though the Sym4, Daub20, Daub10 and Coif12 are all not far behind. Also, it is evident that the best resolution for all of these transforms is 2 or 3. Higher resolutions tend to increase the density.

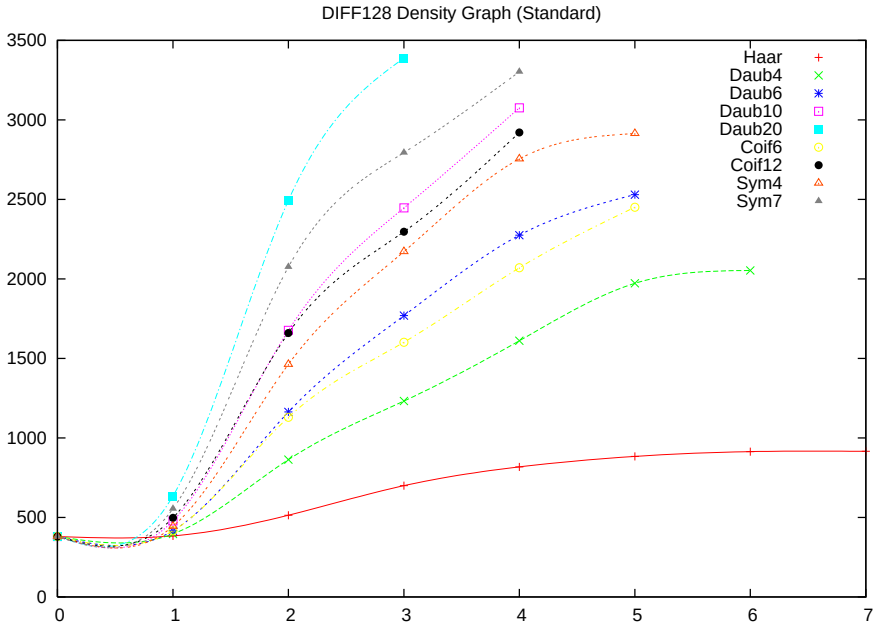


Figure 68: Density Graph for Different Resolution Standard Transforms of $\frac{d^2}{dz^2}$

Here we see how the $\frac{d^2}{dz^2}$ matrix reacts badly to the transform, becoming more dense. As opposed to the χ_0 matrix, longer support here makes things worse for every resolution level.

Packet

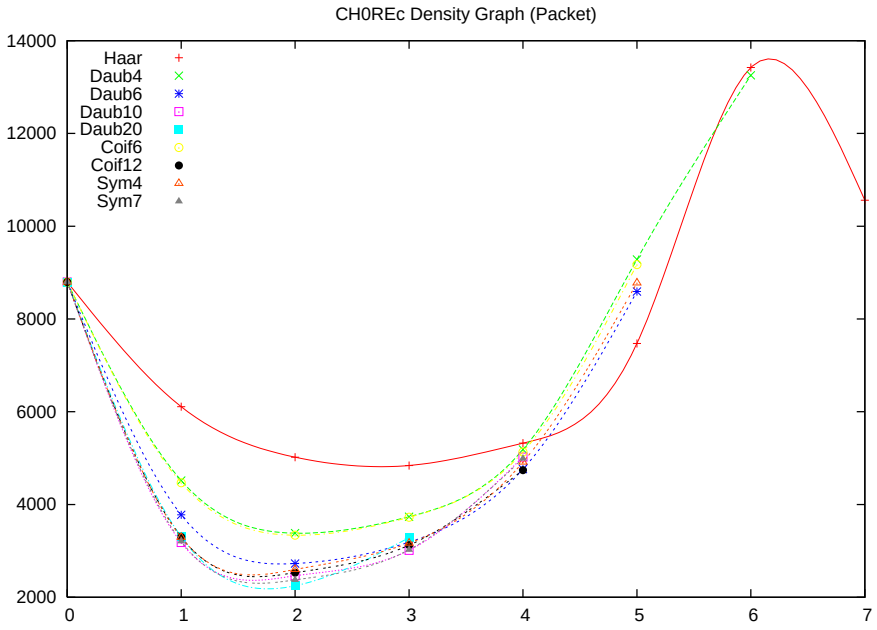


Figure 69: Density Graph for Different Resolution Packet Transforms of χ_0

We can see here that the densities don't dive as sharply as they did for the standard transform. Also, transform resolutions above 2 make the density much worse, and high resolutions have densities about 150% of the original matrix's density.

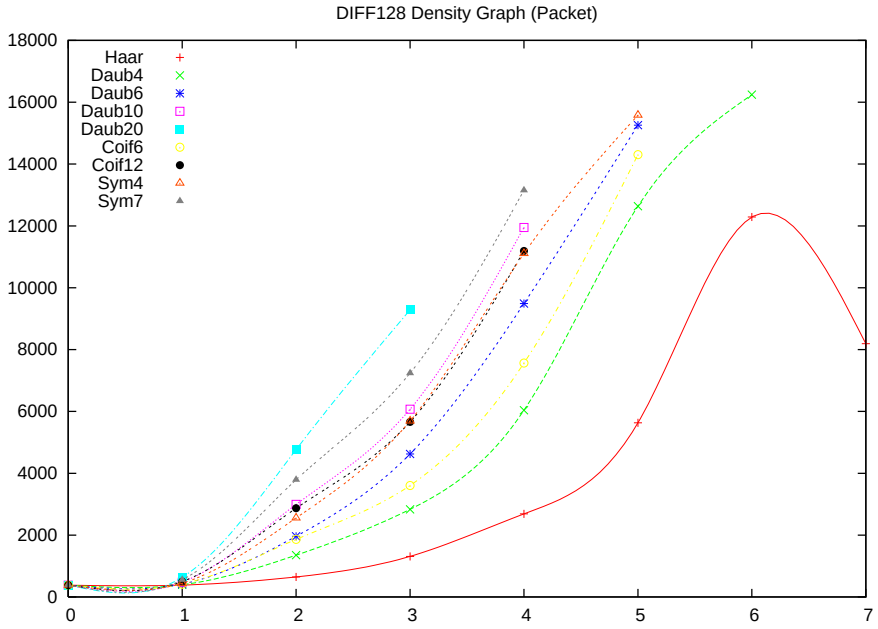


Figure 70: Density Graph for Different Resolution Packet Transforms of $\frac{d^2}{dz^2}$

As before, the $\frac{d^2}{dz^2}$ matrix gets more dense, but the numbers here are five times larger. The packet transform is not acceptable for our problem.

Interestingly, the Haar packet transform's density seems to dip slightly at its maximum resolution for both the χ_0 and the $\frac{d^2}{dz^2}$ matrices. Why this is the case is unclear.

Non-Standard

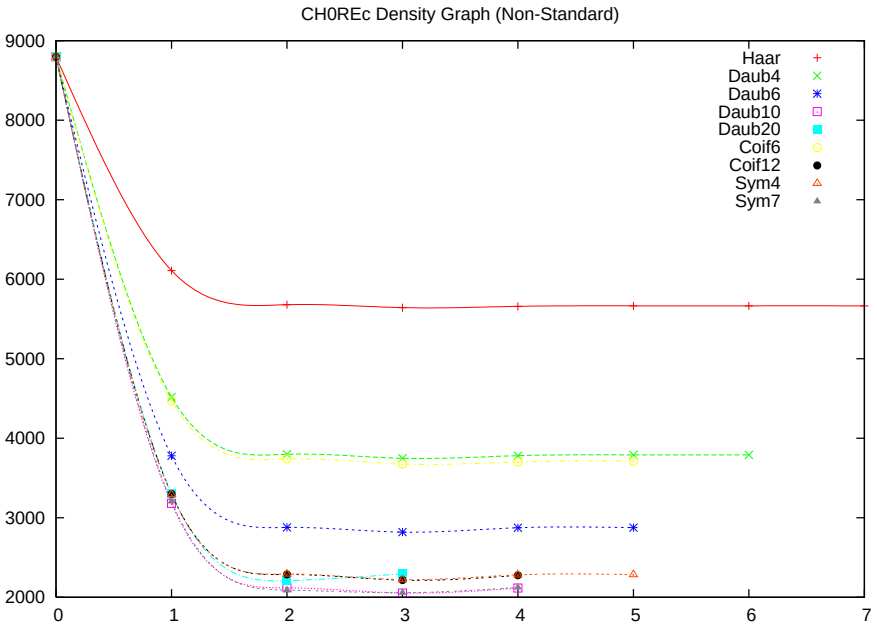


Figure 71: Density Graph for Different Resolution Non-Standard Transforms of χ_0

The non-standard transform seems to behave similarly to the standard transform for χ_0, except at higher resolution levels where the densities of standard transforms slightly increase but the densities non-standard transforms stay reasonably constant.

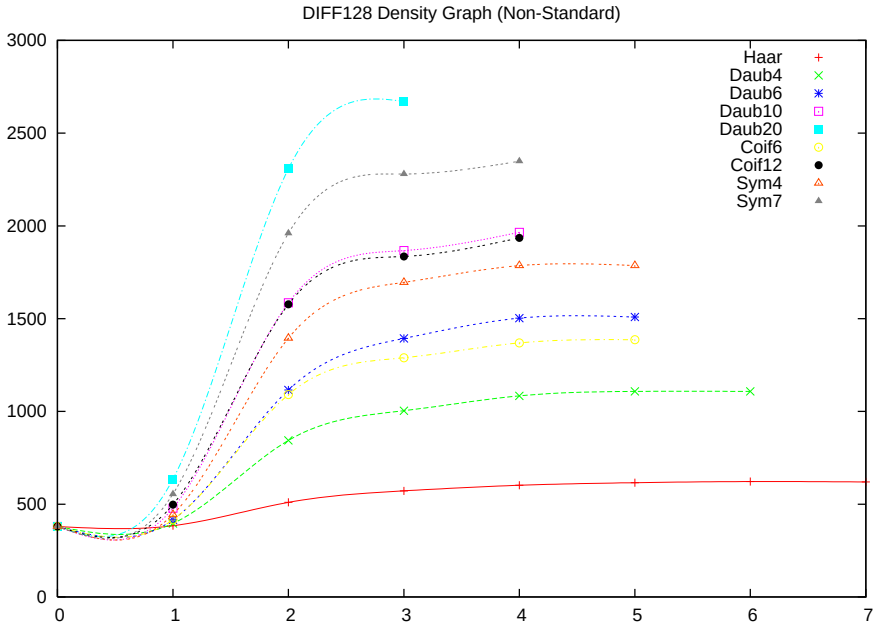


Figure 72: Density Graph for Different Resolution Non-Standard Transforms of $\frac{d^2}{dz^2}$

The non-standard transform of $\frac{d^2}{dz^2}$ is similar in this respect, but slightly better than the standard transform due to the extra levelling-off of densities by decoupling the resolutions.

Considering that the size of the matrix changes for each resolution in the non-standard system it may be better to consider the ratio of the density function to the number of entries in the matrix (see Section 4.2).

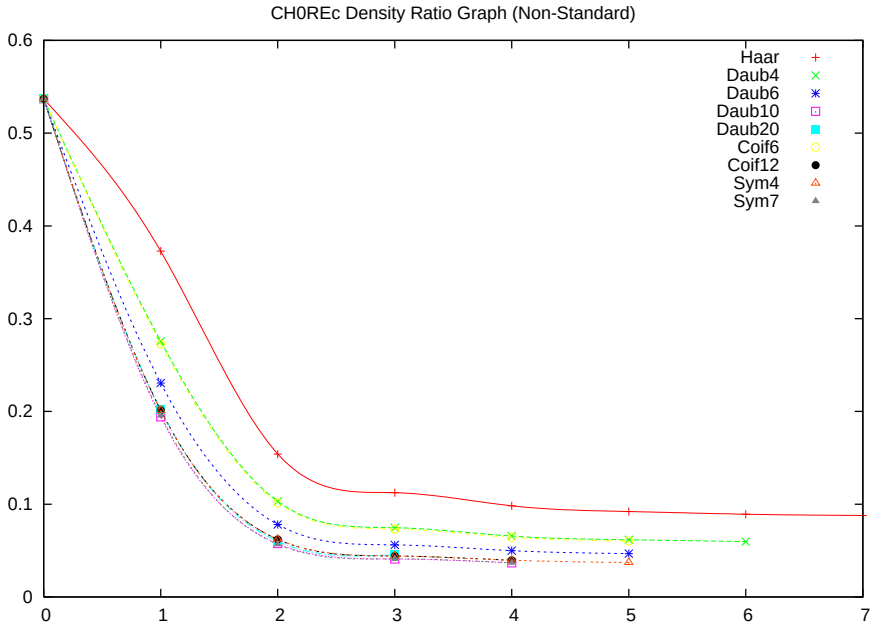


Figure 73: Density Ratio Graph for Different Resolution Non-Standard Transforms of χ_0

In this graph the decreasing nature of the χ_0 matrix is more pronounced. We can see how the lower levels decrease rapidly, due to the matrix expanding a greater amount at those levels.

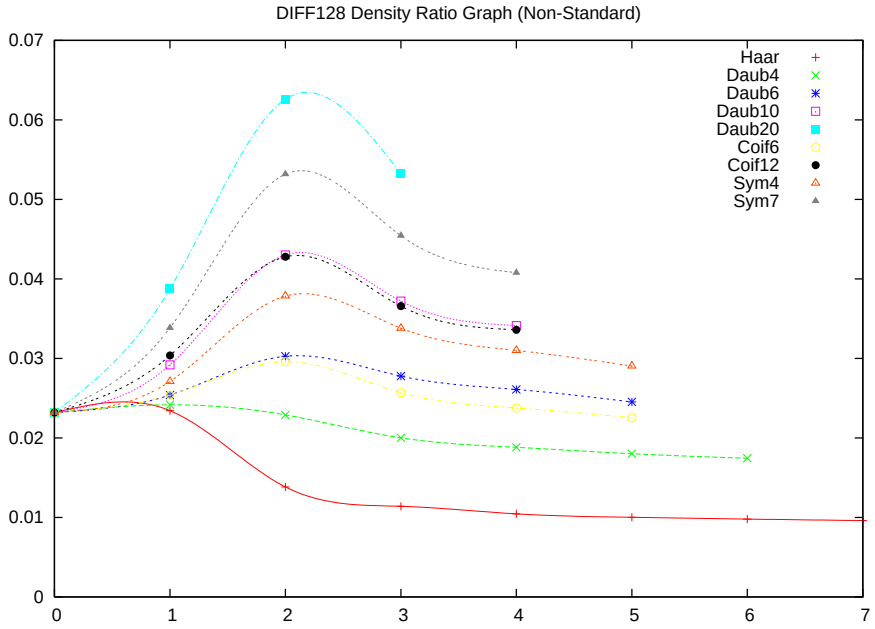


Figure 74: Density Ratio Graph for Different Resolution Non-Standard Transforms of $\frac{d^2}{dz^2}$

This is much more encouraging. It looks like after a small bump in relative density at the first couple of levels, higher levels generally taper down and in some cases go below the relative density of the original. The Daub4 and Haar both do this. The Coif6 and Daub6 slightly increase the relative densities, and the other transforms markedly increase densities.

5 Discussion

In the $\frac{d^2}{dz^2}$ transform "dots" of significant values will commonly appear at the side and corners of the region being transformed in any particular resolution level. These values are likely due to the periodic nature of the wavelet coefficients interacting with the non-periodicity of the $\frac{d^2}{dz^2}$ matrix. It is possible that a specially constructed $\frac{d^2}{dz^2}$ matrix might alleviate this problem.

There is a relationship between support length and the response of χ_0 and $\frac{d^2}{dz^2}$, evidenced by the density graphs in the previous section. χ_0 transforms better with longer support, and $\frac{d^2}{dz^2}$ transforms better with shorter support. This competing behaviour makes it difficult to chose a suitable basis — we need one that fits somewhere in between.

For every wavelet type, standard transforms work well on χ_0, with minimal densities at resolution levels 2 and 3, but poorly on $\frac{d^2}{dz^2}$, with densities rapidly increasing for each resolution level. Packet transforms of $\frac{d^2}{dz^2}$ are much worse than standard transforms, with densities about five times greater. Packet transforms of χ_0 have similar densities for 2 and 3 level resolutions, but higher resolutions increase densities dramatically, even beyond the original matrix density. Packet transforms also tend to spread significant values further from the diagonal, thus increasing the difficulty to invert the matrix.

Non-standard transforms are the best. They keep the density of the $\frac{d^2}{dz^2}$ matrix reasonably stable for wavelets of shorter support, especially when compared to other transforms, while making

the χ_0 matrix notably sparse. Non-standard transforms of $\frac{d^2}{dz^2}$ are at least on par with standard transforms, and are much better than packet transforms of $\frac{d^2}{dz^2}$.

The Daub4 non-standard transform with a resolution ≥ 2 seems to be the best of both worlds. Its support is relatively short, but it transforms the χ_0 on par with the Coif6, and not far from the Daub6 non-standard. Also, the Daub4 transforms χ_0 to less than half of its original density, solving one part of our problem. $\frac{d^2}{dz^2}$, the other part of our problem, is also treated reasonably well with the Daub4 non-standard transform.

6 Future Investigation

6.1 Actual Solution of the Problem

This project stops short of actually solving the original equation using the transforms described. The logical next step is to go ahead and carry out this solution. From there, χ and then the groundstate electron energy can be calculated.

6.2 Optimisation of Code

As this project is more a "proof-of-concept" the code used is not as optimised as it could or should be to generate a real solution. The code currently employs matrix multiplication to perform wavelet transforms, but a customised iterative algorithm would likely be much more efficient. Other areas that could be investigated to improve code performance include parallelisation of the algorithms used, and the use of the Fast Wavelet Transform algorithm [6, p. 13].

6.3 Advanced Wavelet Theory

The aim here was to find a wavelet transform that would make both the χ_0 and the $\frac{d^2}{dz^2}$ matrices less sparse. We have seen that there are wavelet bases that can do this quite well for the χ_0 signal, but they do not perform so well for the $\frac{d^2}{dz^2}$ signal. It should be noted that a useful transform of a $\frac{d^2}{dz^2}$ matrix has already been accomplished [5], so by the fact that we know it is feasible that a wavelet transform can produce sparser χ_0 matrices we can be optimistic about the possible uses of the advanced wavelet theory used by Goedecker in his solution of the Coulomb problem [5]. There is a bulk of wavelet theory that was not covered in this project, including biorthogonal, lifted, or "second-generation" wavelets, wavelets that are not compactly supported, spline wavelets [1], continuous, and even truly 2- or 3-dimensional wavelets [8]. There is also the possibility of designing new wavelets to specifically solve this problem.

6.4 Expanding the Use of Wavelets Into Systems of Higher Dimensions

If wavelet theory proves to be useful, and the transforms performed in this project indicates that it is, expanding the use of wavelets to solve real world systems will become the ultimate goal. Such an expansion will likely require 3-dimensional wavelets.

7 Conclusion

The solution of van der Waals forces through the adiabatic connection-fluctuation-dissipation theorem employing the random phase approximation method produces matrices that are not easy to invert. This project has shown that, through the use of wavelet bases, some of the sparseness of the $\frac{d^2}{dz^2}$ matrix necessary in the solution can, in a sense, be sacrificed to make the dense χ_0 matrix, also necessary, more sparse. The resulting densities for the Daub4 non-standard transform are such that the pair of resultant matrices appears to be in total less dense, thus easier to solve, than the original matrices. Even on its own this is a useful result, but it is also a strong indicator

of the power of more advanced wavelet theory and transformation techniques. Investigating these appears as though it would be of value. Though it is too early to tell for sure, the results of this project suggest that wavelet analysis may have found yet another application in physics. If so, the solution of van der Waals forces in soft matter may become more practicable.

References

[1] Ingrid Daubechies, *Ten lectures on wavelets*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1992.

[2] John F. Dobson, *Screening for B. Higgins*, Personal Communication, August 2004.

[3] John F. Dobson and Jun Wang, *Successful test of a seamless van der waals density functional*, Physical Review Letters **82** (1999), no. 10, 2123.

[4] John F. Dobson, Jun Wang, Bradley P. Dinte, and Hung M. Le, *Soft cohesive forces*, International Journal of Quantum Chemistry **100** (In press 2004).

[5] S. Goedecker and O. V. Ivanov, *Linear scaling solution of the coulomb problem using wavelets*, Solid State Communications **105** (1998), no. 11, 665–669.

[6] Stefan Goedecker, *Wavelets and their application*, Presses polytechniques et universitaires romandes, 1998.

[7] Wolfgang Härdle, Gerard Kerkyacharian, Dominique Picard, and Alexander Tsybakov, *Wavelets, approximation, and statistical applications*, Springer-Verlag, New York, 1998.

[8] J. C. van den Berg (ed.), *Wavelets in physics*, Cambridge University Press, 1999.

[9] James S. Walker, *A primer on wavelets and their scientific applications*, CRC Press LLC, 1999.

[10] Angela White, *Honours thesis*, School of Science, Griffith Uni., 2003.

Appendices

Appendix A: The Mysterious Wavelet

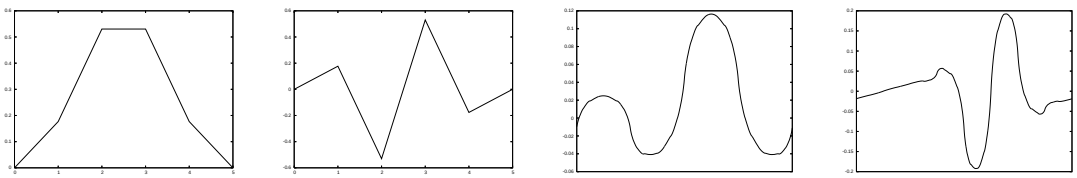
During the execution of this project a "wavelet" was discovered that seemed to produce highly sparse transforms of both the χ_0 and $\frac{d^2}{dz^2}$ matrices. The coefficients were found at `http://cvs.sourceforge.net/viewcvs.py/snd/cvs-snd/snd-fft.c?rev=1.217&view=markup`. It has since become evident that the coefficients used were not what they claimed to be. Originally labelled a "sym3" wavelet, it is now known that it is not a symmlet as originally thought. Early investigation suggests that it is in fact a biorthogonal spline wavelet with $\tilde{N} = 3$ and $N = 1$ [1, pp. 271-278]. The results of its transform are displayed here.

Note The following data is suspect. More investigation into the coefficients used in order to determine if they form a true wavelet basis is required before this particular transform might be taken seriously. The effect of biorthogonal wavelets to this problem has yet to be determined, and the mechanism used to construct the W coefficients from the V coefficients is almost certainly wrong. Nevertheless, it is listed here in case it turns out to be useful.

A1: The Wavelet

The code used to generate the "wavelet", and hence its coefficients, is listed below.

```
% Create Unknown scaling and wavelet coefficients for signal of length len.
function [V11, W11] = sym3(len)
    V11 = zeros(1, len);
    V11(1) = 0.1767767;
    V11(2) = 0.5303301;
    V11(3) = 0.5303301;
    V11(4) = 0.1767767;
    W11 = zeros(1, len);
    W11(1) = V11(4);
    W11(2) = -V11(3);
    W11(3) = V11(2);
    W11(4) = -V11(1);
end
```



(a) Scaling Function Coefficients (b) Wavelet Coefficients (c) Scaling Function (d) Wavelet

Figure 75: Sym3 Wavelet

A2: Standard Transforms

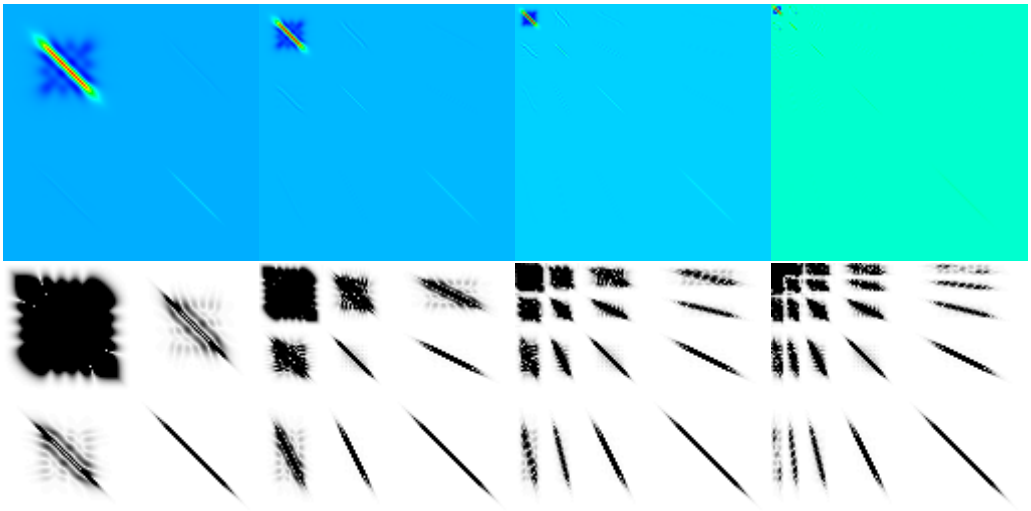


Figure 76: Sym3 Standard Transform of χ_0

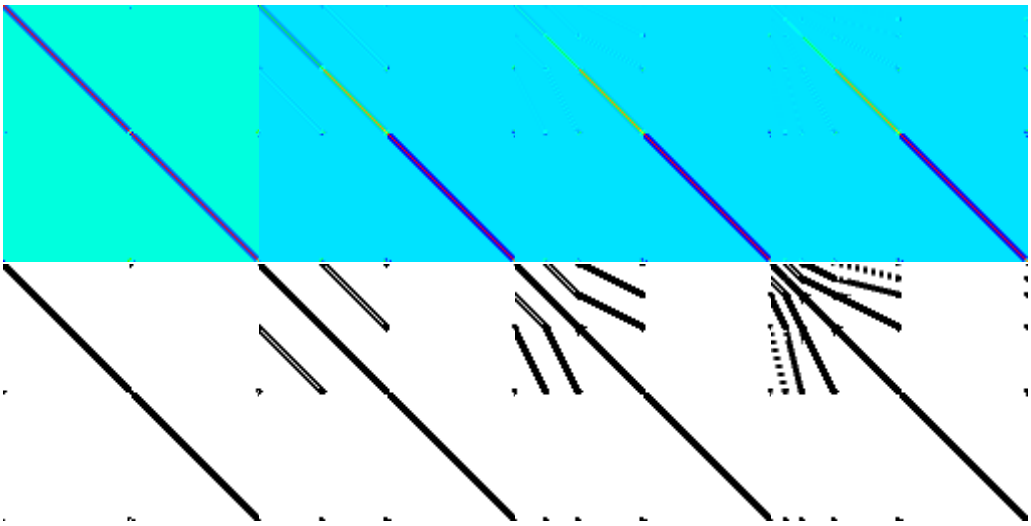


Figure 77: Sym3 Standard Transform of $\frac{d^2}{dz^2}$

A3: Packet Transforms

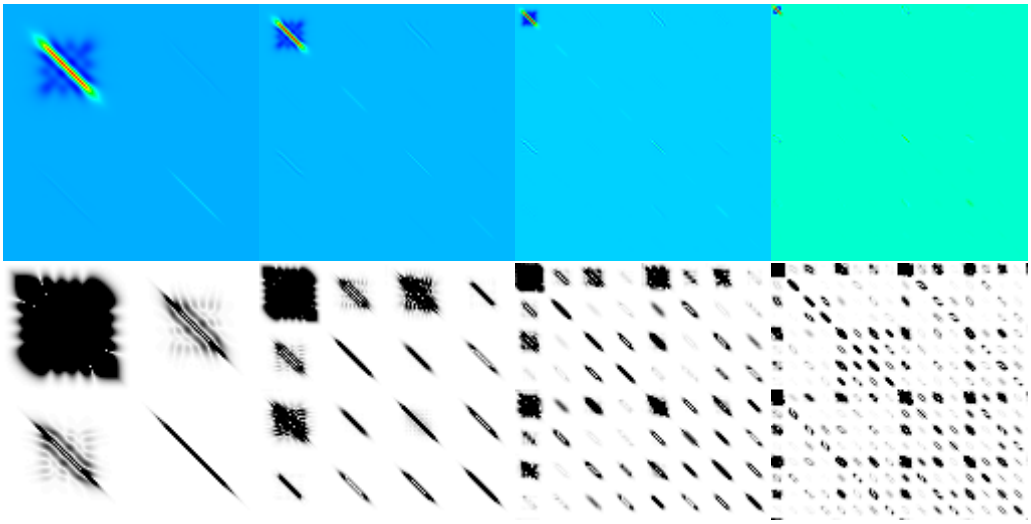


Figure 78: Sym3 Packet Transform of χ_0

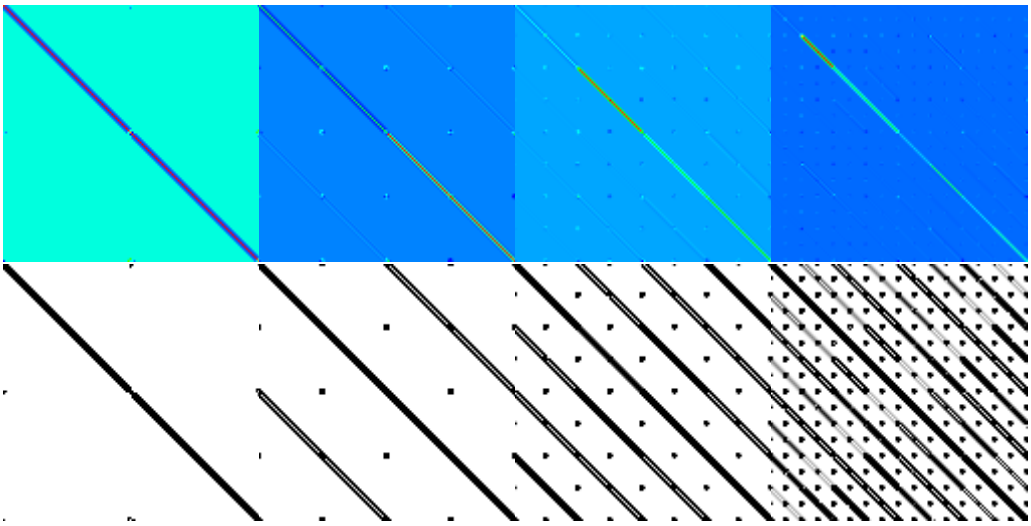


Figure 79: Sym3 Packet Transform of $\frac{d^2}{dz^2}$

A4: Non-Standard Transforms

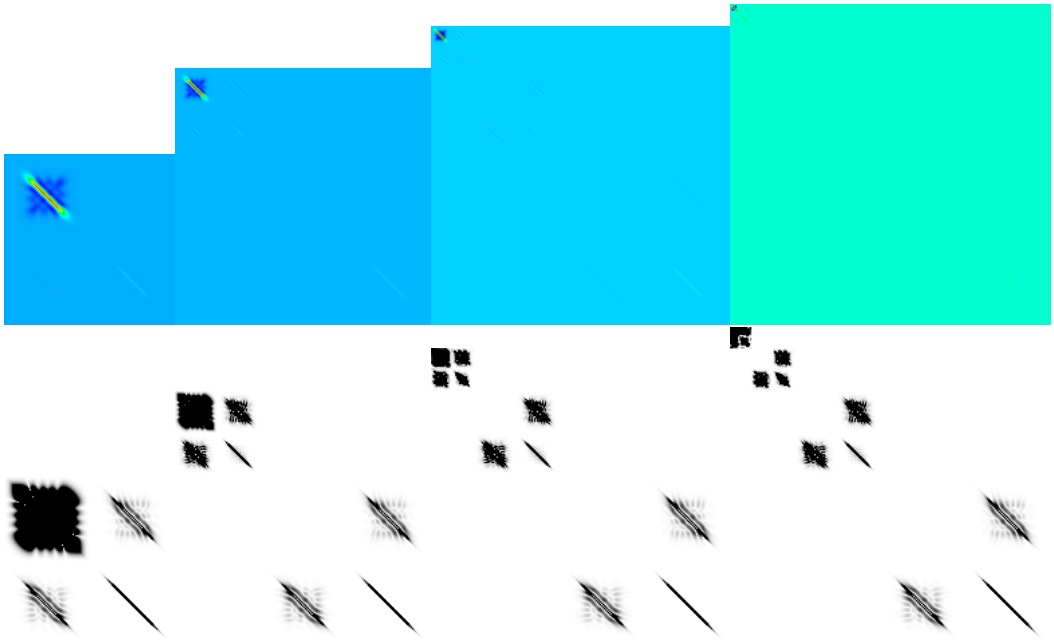


Figure 80: Sym3 Non-Standard Transform of χ_0

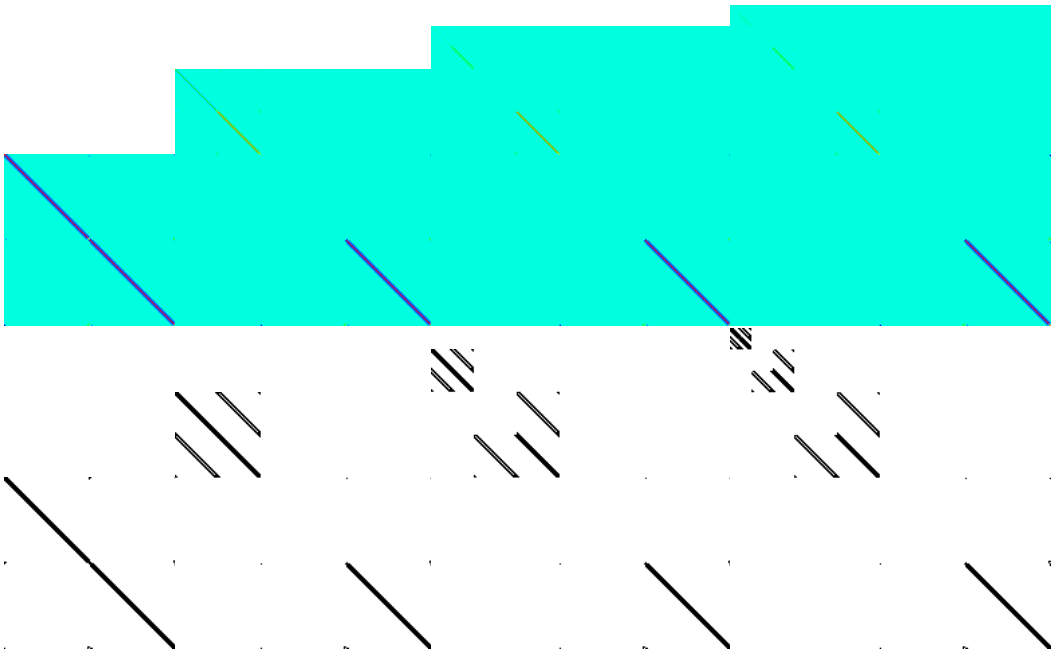


Figure 81: Sym3 Non-Standard Transform of $\frac{d^2}{dz^2}$

The transforms behave similarly to what we have already seen, only better.

A5: Densities of Transforms

Level	Standard Density	Packet Density	NS Density	NS Density Ratio
1	3127.56	3127.56	3127.56	0.1908910
2	1915.97	1892.22	1925.25	0.0522257
3	1621.40	1596.65	1755.29	0.0349826
4	1622.21	1594.88	1764.52	0.0306340
5	1663.43	1980.08	1766.43	0.0287206
6	1641.36	2863.75	1766.43	0.0278160

Table 1: Sym3 Density for Different Resolutions of χ_0

Level	Standard Density	Packet Density	NS Density	NS Density Ratio
1	648	648	648	0.0395508
2	932	1236	912	0.0247396
3	1330.15	2543.71	1044.15	0.0208098
4	1586.33	4570.54	1111.5	0.0192968
5	1613.71	7248.85	1120.29	0.0182149
6	1639.03	13145.20	1118.35	0.0176107

Table 2: Sym3 Density for Different Resolutions of $\frac{d^2}{dz^2}$

The density figures listed here are generally lower than any other transform shown in this report. If this "sym3" were in fact a proper wavelet basis it would be a very good choice for this problem. Unfortunately, this may not be the case.

Appendix B: Source Code Employed in this Project

Code for this project was written for the free open-source Matlab-like system called "Octave". Octave programs are very similar to Matlab programs, but there are small differences. Converting this code into Matlab code should for the most part be trivial. All wavelet coefficients used can be found in Daubechies [1].

B1: Measuring Density (`density.m`)

```
% density(matrix, threshold)
% This gives a crude approximation of how easy a matrix is to solve.
% The higher the number, the more non-zero elements are in the matrix.
% To get the ratio of how many elements are greater than the threshold,
% divide this by the matrix area.
function res = density(matrix, threshold)
    res = 0;
    [r, c] = size(matrix);
    for i = 1:r
        for j = 1:c
            x = abs(matrix(i, j)) / threshold;
            if x < 1
                res += x;
            else
                res += 1;
            end
        end
    end
end
```

B2: Reading χ_0 Data (`readch0.m`)

```
% readch0(file) reads the given file the contains the usual expected Chi0
% format to produce a square matrix of the Chi0 signal.
function ret = readch0(file)
    [fid, msg] = fopen(file, 'rt', 'native');
    % Ignore IDs and some stuff
    fgetl(fid);
    fgetl(fid);
    fgetl(fid);
    fgetl(fid);
    % Read start, step, num
    [start, step, num] = fscanf(fid, '%d %d %d', 'C');
    fgetl(fid);
    range = start:step:(num * step + start - 1);
    ret = zeros(num);
    % NOTE: Chi0 is symmetric in i, j.
    for i = range
        p = fscanf(fid, '%d', 'C');
        fgetl(fid);
        for j = range
            ret(i, j) = fscanf(fid, '%d', 'C') * 10**p;
        end
    end
    fclose(fid);
```

```
end
```

B3: Generating $\frac{d^2}{dz^2}$ Data (`makediffop.m`)

Note: The following function produces a $\frac{d}{dz}$ operator matrix. Multiply this matrix with itself to find a $\frac{d^2}{dz^2}$ matrix.

```
% makediffop(size)
% Makes a differentiation matrix operation
% returns square matrix of size, uses central difference method.
% To get d2/dx2, square this matrix.
function ret = makediffop(size)
    % diag with an offset extends the matrix to use all the elements,
    % thus give it one less element to make sure we get the right size
    ret = (diag(-ones(1, size - 1), -1) + diag(ones(1, size - 1), 1)) ./ 2;
end
```

B4: Creating a Wavelet Transform Matrix (`matlet.m`)

```
% matlet(basis, len, res) creates a transformation matrix that will
% perform a wavelet transform in the given basis of resolution res on a signal
% of lenth len.
% basis must be a string giving the name of the basis to be used.
function ret = matlet(basis, len, res)
    halflen = len / 2;
    ret = zeros(len);
    [V11, W11] = feval(basis, len);
    ret(1, :) = V11;
    ret(halflen + 1, :) = W11;
    for i = 2:halflen
        ret(i, :) = shift(ret(i - 1, :), 2);
        ret(i + halflen, :) = shift(ret(i + halflen - 1, :), 2);
    end

    % Yay recursion!
    if res > 1
        next = matlet(basis, halflen, res - 1);
        ret = expand(next, len) * ret;
    end
end
```

B5: Creating a Wavelet Packet Transform Matrix (`matlet_packet.m`)

```
% matlet_packet(basis, len, res) creates a transformation matrix that will
% perform a wavelet packet transform in the given basis of resolution res on a
% signal of lenth len.
function ret = matlet_packet(basis, len, res)
    % this is really easy, just apply matlet to itself res times
    ret = matlet(basis, len, 1)^res;
end
```

B6: Expanding a Matrix Operator (`expand.m`)

```
% expand(matrix, size) resizes a square matrix by copying it to
```

```
% the top corner of an identity matrix of given size.
function ret = expand(m, s)
    ret = zeros(s);
    [r, c] = size(m);
    for i = 1:s
        for j = 1:s
            if (i <= r && j <= c)
                ret(i, j) = m(i, j);
            elseif (i == j)
                ret(i, j) = 1;
            else
                ret(i, j) = 0;
            end
        end
    end
end
```

B7: Performing a Wavelet Transform on a Matrix (`transformch0.m`)

```
% transformch0(data, basis, res) performs a wavelet transform of given
% basis and resolution res on the given data, a square matrix (most
% often the Chi0 signal).
function ret = transformch0(data, basis, res)
    [rows, cols] = size(data);
    ret = zeros(rows, cols);
    m = matlet(basis, cols, res);
    for i = 1:rows
        ret(i, :) = transpose(m * transpose(data(i, :)));
    end
    for i = 1:cols
        ret(:, i) = m * ret(:, i);
    end
end
```

B8: Performing a Wavelet Packet Transform on a Matrix (`packettransformch0.m`)

```
% packettransformch0(data, basis, res) performs the wavelet packet
% transform of given basis and resolution res on the given data, a square
% matrix (most often the Chi0 signal).
function ret = packettransformch0(data, basis, res)
    [rows, cols] = size(data);
    ret = zeros(rows, cols);
    m = matlet_packet(basis, cols, res);
    for i = 1:rows
        ret(i, :) = transpose(m * transpose(data(i, :)));
    end
    for i = 1:cols
        ret(:, i) = m * ret(:, i);
    end
end
```

B9: Performing a Wavelet Non-Standard Transform on a Matrix (`nstransformch0.m`)

```
% nstransformch0(data, basis, res) performs the non-standard wavelet
% transform of given basis and resolution res on the given data, a square
% matrix (most often the Chi0 signal).
function ret = nstransformch0(data, basis, res)
    if (res == 0)
        % there is no 0th level! no transform.
        ret = data;
        return
    end

    if (res == 1)
        % dont need to expand the matrix for last level
        ret = transformch0(data, basis, 1);
        return
    end

    % insert the zeros
    % endcols = cols * (1 - 0.5^(res)) / 0.5
    % from sum of GP
    [rows, cols] = size(data);
    outrows = 2 * rows * (1 - 0.5^(res));
    outcols = 2 * cols * (1 - 0.5^(res));
    ret = zeros(outrows, outcols);
    datat = transformch0(data, basis, 1);
    for i = 1:rows
        ret((outrows - rows + i), (outcols - cols / 2 + 1):outcols) \
            = datat(i, (cols / 2 + 1):cols);
    end
    for i = 1:(cols/2)
        ret((outrows - rows / 2 + 1):outrows, (outcols - cols + i)) \
            = datat((rows / 2 + 1):rows, i);
    end
    ret(1:(outrows - rows), 1:(outcols - cols)) \
        = nstransformch0(datat(1:(rows / 2), 1:(cols / 2)), basis, res - 1);
end
```

B10: Recording Transformation Data (`savetransform.m`)

```
% savetransform(name, data)
% makes files name.[data|png]
function savetransform(name, data)
    % the in-built save function does not like dynamic generated
    % filenames, so we have to use our own little format:
    % First is number of columns rows newline
    % then a row newline, another row newline, etc
    f = fopen(strcat(name, '.data'), 'w');
    [h, w] = size(data);
    fprintf(f, '%d %d\n', w, h);
    for i = 1:h
        for j = 1:w
            fprintf(f, '%g ', data(i, j));
        end
```

```
        fprintf(f, '\n');
    end
    fclose(f);
    % end hack

    threshold = 1e-4;

    % save the extremes for easy lookup
    f = fopen(strcat(name, '.extrema'), 'w');
    little = min(transpose(min(data)));
    big = max(transpose(max(data)));
    dens = density(data, threshold);
    densrat = dens / rows(data) / columns(data);
    fprintf(f, 'min: %g\nmax: %g\ndensity: %g\ndensity ratio: %g',
        little, big, dens, densrat);
    fclose(f);

    % make some pictures
    % scale the result to something reasonable
    im = imagesc(data);
    savepng(strcat(name, '.png'), im);
    % also save a significance map
    % change the colormap to nice grays
    cm = colormap();
    colormap(1 - gray());
    im = imagesc(abs(data), [0, threshold]);
    savepng(strcat(name, '_sigmap.png'), im);
    colormap(cm);
end

function savepng(name, im)
    saveimage('temp.ppm', im, 'ppm');
    % This is probably NOT portable. Uses an ImageMagick program to covert
    % images into the more disk-space-friendly PNG format.
    system(strcat('convert temp.ppm ', name, '; rm temp.ppm'));
end
```

B11: Generating Density-Graph Data (`buildresgraph.m`)

```
% buildresgraph(prefix, bases, levels, type)
% eg buildresgraph('transforms/CHOREc', ['haar'; 'daub4'; 'daub6'; \
%   'daub10'; 'daub20'; 'coif6'; 'coif12'; 'sym4'; 'sym7'], \
%   [1, 2, 3, 4, 5, 6, 7], 'ns')
% Reads previously calculated extrema/density data and builds data for two
% line graphs showing density and density ratios of wavelet transforms as the
% resolutions changes over the given bases. Also includes the stats of the
% original. If a file for a resolution doesn't exist, 'X' will be put in the
% place of the data output for that res.
function buildresgraph(prefix, bases, levels, type)
    dendata = zeros(rows(bases), levels + 1);
    ratdata = zeros(rows(bases), levels + 1);

    [dendata(:, 1), ratdata(:, 1)] = readdensity(strcat(prefix, '.extrema'));
```

```
    for j = 1:rows(bases)
        % try to get rid of trailing whitespace
        % thats added to make the matrix rectangular
        [base, n] = sscanf(bases(j, :), '%s');

        for i = 1:length(levels)
            % load the stuff
            s = sprintf('%d', i);
            name = strcat(prefix, '_', base, '_', s);
            if type != ''
                name = strcat(name, type);
            end
            name = strcat(name, '.extrema');
            [dens, rat] = readdensity(name);
            dendata(j, i + 1) = dens;
            ratdata(j, i + 1) = rat;
        end
    end

    dname = strcat(prefix, '_density');
    rname = strcat(prefix, '_dratio');
    if type != ''
        dname = strcat(dname, '_', type);
        rname = strcat(rname, '_', type);
    end

    writegraphdata(strcat(dname, '.data'), bases, dendata);
    writegraphdata(strcat(rname, '.data'), bases, ratdata);
end

function [d, r] = readdensity(name)
    f = fopen(name, 'rt');
    if f == -1
        d = -1;
        r = -1;
        return;
    end
    fgetl(f);
    fgetl(f);
    [data, krap] = fscanf(f, 'density: %g\ndensity ratio: %g');
    fclose(f);
    d = data(1);
    r = data(2);
end

function writegraphdata(name, bases, data)
    f = fopen(name, 'w');
    [nbases, nlevels] = size(data);

    fprintf(f, '# l ');
    for i = 1:nbases
        fprintf(f, '%s ', bases(i, :));
    end
    fprintf(f, '\n');
```

```
    for i = 1:nlevels
        fprintf(f, '%d ', i - 1);
        for j = 1:nbases
            x = data(j, i);
            if x != -1
                fprintf(f, '%g ', data(j, i));
            else
                fprintf(f, 'X ');
            end
        end
        fprintf(f, '\n');
    end
    fclose(f);
end
```

B12: Performing a Set of Transforms (`dotransforms.m`)

```
% dotransforms(prefix, method, data, [base; base; ...], [level, level, ...])
% makes files: prefix_base_levelM.[data|png]
% M may be nothing, 'packet', or 'ns', corresponding to method
% methods are: 'normal', 'packet', 'nonstandard', or 'all'
function dotransforms(prefix, method, data, bases, levels)
    if (~strcmp(method, 'normal') && ~strcmp(method, 'packet') \
        && ~strcmp(method, 'nonstandard') && ~strcmp(method, 'all'))
        printf('Please specify a method; normal, packet, or nonstandard.\n');
        return
    end

    % The rest
    datat = data;
    name = '';
    for j = 1:rows(bases)
        % try to get rid of trailing whitespace
        % thats added to make the matrix rectangular
        [base, n] = sscanf(bases(j,:), '%s');
        printf('Doing base %s level ', base);
        fflush(stdout);
        for i = 1:length(levels)
            printf('%d ', levels(i));
            fflush(stdout);
            if (strcmp(method, 'normal') || strcmp(method, 'all'))
                datat = transformch0(data, base, levels(i));
                name = strcat(prefix, '_', base, '_', int2str(levels(i)));
                savetransform(name, datat);
            end
            if (strcmp(method, 'packet') || strcmp(method, 'all'))
                datat = packettransformch0(data, base, levels(i));
                name = strcat(prefix, '_', base, '_', \
                    int2str(levels(i)), 'packet');
                savetransform(name, datat);
            end
            if (strcmp(method, 'nonstandard') || strcmp(method, 'all'))
                datat = nstransformch0(data, base, levels(i));
```

```
                name = strcat(prefix, '_', base, '_', int2str(levels(i)), 'ns');
                savetransform(name, datat);
            end
        end
        printf('\n');
    end
end
```

B13: The Haar Wavelet (haar.m)

```
% Create Haar scaling and wavelet coefficients for signal of length len.
function [V11, W11] = haar(len)
    V11 = zeros(1, len);
    V11(1) = 1 / sqrt(2);
    V11(2) = 1 / sqrt(2);
    W11 = zeros(1, len);
    W11(1) = V11(2);
    W11(2) = -V11(1);
end
```

B14: The Daub4 Wavelet (daub4.m)

```
% Create Daub4 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = daub4(len)
    rt2t4 = sqrt(2) * 4;
    rt3 = sqrt(3);
    V11 = zeros(1, len);
    V11(1) = (1 + rt3) / rt2t4;
    V11(2) = (3 + rt3) / rt2t4;
    V11(3) = (3 - rt3) / rt2t4;
    V11(4) = (1 - rt3) / rt2t4;
    W11 = zeros(1, len);
    W11(1) = V11(4);
    W11(2) = -V11(3);
    W11(3) = V11(2);
    W11(4) = -V11(1);
end
```

B15: The Daub6 Wavelet (daub6.m)

```
% Create Daub6 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = daub6(len)
    V11 = zeros(1, len);
    V11(1) = 0.332670552950083;
    V11(2) = 0.806891509311092;
    V11(3) = 0.459877502118491;
    V11(4) = -0.135011020010255;
    V11(5) = -0.0854412738820267;
    V11(6) = 0.0352262918857095;
    W11 = zeros(1, len);
    W11(1) = V11(6);
    W11(2) = -V11(5);
    W11(3) = V11(4);
```

```
    W11(4) = -V11(3);
    W11(5) = V11(2);
    W11(6) = -V11(1);
end
```

B16: The Daub10 Wavelet (daub10.m)

```
% Create Daub10 scaling and wavelet coefficients for signal of length len.
% This is just one kind of Daub10, pilfered from Goedecker
function [V11, W11] = daub10(len)
    % todo
    V11 = zeros(1, len);
    V11(1) = 0.1601023979741929;
    V11(2) = 0.6038292697971897;
    V11(3) = 0.7243085284377729;
    V11(4) = 0.1384281459013207;
    V11(5) = -0.2422948870663820;
    V11(6) = -0.0322448695846384;
    V11(7) = 0.0775714938400457;
    V11(8) = -0.0062414902127983;
    V11(9) = -0.0125807519990820;
    V11(10) = 0.0033357252854738;
    W11 = zeros(1, len);
    W11(1) = V11(10);
    W11(2) = -V11(9);
    W11(3) = V11(8);
    W11(4) = -V11(7);
    W11(5) = V11(6);
    W11(6) = -V11(5);
    W11(7) = V11(4);
    W11(8) = -V11(3);
    W11(9) = V11(2);
    W11(10) = -V11(1);
end
```

B17: The Daub20 Wavelet (daub20.m)

```
% Create Daub20 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = daub20(len)
    V11 = zeros(1, len);
    V11(1) = 2.667005790055555358661744877130e-02;
    V11(2) = 1.881768000776914890208929736790e-01;
    V11(3) = 5.272011889317255864817448279590e-01;
    V11(4) = 6.884590394536035657418717825490e-01;
    V11(5) = 2.811723436605774607487269984450e-01;
    V11(6) = -2.498464243273153794161018979200e-01;
    V11(7) = -1.959462743773770435042992543100e-01;
    V11(8) = 1.273693403357932600826772332010e-01;
    V11(9) = 9.305736460357235116035228983540e-02;
    V11(10) = -7.139414716639708714533609307000e-02;
    V11(11) = -2.945753682187581285828323760000e-02;
    V11(12) = 3.321267405934100173976365318200e-02;
    V11(13) = 3.606553566956169655423291417100e-03;
    V11(14) = -1.073317548333057504431811410000e-02;
```

```
    V11(15) = 1.39535174705290116578931844 79e-03;
    V11(16) = 1.99240529518505611715874224 26e-03;
    V11(17) = -6.85856694959711626561370981 9e-04;
    V11(18) = -1.16466855129285450951480971 0e-04;
    V11(19) = 9.35886703200695913340501303 42e-05;
    V11(20) = -1.32642028945212448124366753 1e-05;
    W11 = zeros(1, len);
    W11(1) = V11(20);
    W11(2) = -V11(19);
    W11(3) = V11(18);
    W11(4) = -V11(17);
    W11(5) = V11(16);
    W11(6) = -V11(15);
    W11(7) = V11(14);
    W11(8) = -V11(13);
    W11(9) = V11(12);
    W11(10) = -V11(11);
    W11(11) = V11(10);
    W11(12) = -V11(9);
    W11(13) = V11(8);
    W11(14) = -V11(7);
    W11(15) = V11(6);
    W11(16) = -V11(5);
    W11(17) = V11(4);
    W11(18) = -V11(3);
    W11(19) = V11(2);
    W11(20) = -V11(1);
end
```

B18: The Coif6 Wavelet (`coif6.m`)

```
% Create Coif6 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = coif6(len)
    if len < 6
        disp('Can\'t make coiflet that short!');
        % Force an error.
        x(-1) = 0;
    end

    rt2t16 = sqrt(2) * 16;
    rt7 = sqrt(7);
    V11 = zeros(1, len);
    V11(len - 1) = (1 - rt7) / rt2t16;
    V11(len) = (5 + rt7) / rt2t16;
    V11(1) = (14 + 2 * rt7) / rt2t16;
    V11(2) = (14 - 2 * rt7) / rt2t16;
    V11(3) = (1 - rt7) / rt2t16;
    V11(4) = (-3 + rt7) / rt2t16;
    W11 = zeros(1, len);
    W11(len - 1) = V11(4);
    W11(len) = -V11(3);
    W11(1) = V11(2);
    W11(2) = -V11(1);
    W11(3) = V11(len);
```

```
    W11(4) = -V11(len - 1);
end
```

B19: The Coif12 Wavelet (`coif12.m`)

```
% Create Coif12 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = coif12(len)
    if len < 12
        disp('Can\'t make coiflet that short!');
        % Force an error.
        x(-1) = 0;
    end

    V11 = zeros(1, len);
    V11(len - 3) = 0.011587596739;
    V11(len - 2) = -0.029320137980;
    V11(len - 1) = -0.047639590310;
    V11(len) = 0.273021046535;
    V11(1) = 0.574682393857;
    V11(2) = 0.294867193696;
    V11(3) = -0.054085607092;
    V11(4) = -0.042026480461;
    V11(5) = 0.016744410163;
    V11(6) = 0.003967883613;
    V11(7) = -0.001289203356;
    V11(8) = -0.000509505399;
    V11 = V11 .* sqrt(2);
    W11 = zeros(1, len);
    W11(len - 3) = V11(8);
    W11(len - 2) = -V11(7);
    W11(len - 1) = V11(6);
    W11(len) = -V11(5);
    W11(1) = V11(4);
    W11(2) = -V11(3);
    W11(3) = V11(2);
    W11(4) = -V11(1);
    W11(5) = V11(len);
    W11(6) = -V11(len - 1);
    W11(7) = V11(len - 2);
    W11(8) = -V11(len - 3);
end
```

B20: The Sym4 Wavelet (`sym4.m`)

```
% Create Sym4 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = sym4(len)
    V11 = zeros(1, len);
    V11(1) = -0.107148901418;
    V11(2) = -0.041910965125;
    V11(3) = 0.703739068656;
    V11(4) = 1.136658243408;
    V11(5) = 0.421234534204;
    V11(6) = -0.140317624179;
    V11(7) = -0.017824701442;
```

```
    V11(8) = 0.045570345896;
    V11 = V11 ./ sqrt(2);
    W11 = zeros(1, len);
    W11(1) = V11(8);
    W11(2) = -V11(7);
    W11(3) = V11(6);
    W11(4) = -V11(5);
    W11(5) = V11(4);
    W11(6) = -V11(3);
    W11(7) = V11(2);
    W11(8) = -V11(1);
end
```

B21: The Sym7 Wavelet (`sym7.m`)

```
% Create Sym7 scaling and wavelet coefficients for signal of length len.
function [V11, W11] = sym7(len)
    V11 = zeros(1, len);
    V11(1) = 0.003792658534;
    V11(2) = -0.001481225915;
    V11(3) = -0.017870431651;
    V11(4) = 0.043155452582;
    V11(5) = 0.096014767936;
    V11(6) = -0.070078291222;
    V11(7) = 0.024665659489;
    V11(8) = 0.758162601964;
    V11(9) = 1.085782709814;
    V11(10) = 0.408183939725;
    V11(11) = -0.198056706807;
    V11(12) = -0.152463871896;
    V11(13) = 0.005671342686;
    V11(14) = 0.014521394762;
    V11 = V11 ./ sqrt(2);
    W11 = zeros(1, len);
    W11(1) = V11(14);
    W11(2) = -V11(13);
    W11(3) = V11(12);
    W11(4) = -V11(11);
    W11(5) = V11(10);
    W11(6) = -V11(9);
    W11(7) = V11(8);
    W11(8) = -V11(7);
    W11(9) = V11(6);
    W11(10) = -V11(5);
    W11(11) = V11(4);
    W11(12) = -V11(3);
    W11(13) = V11(2);
    W11(14) = -V11(1);
end
```