Memorial University of Newfoundland
ECE 8600 – Design of Digital Signal Processing Systems – 2023
Design Assignment #2

Due Date and Time Wednesday, March 29, by 10pm. *Please submit your completed assignment either in hard-copy, or electronically to a D2L Assignment Folder (i.e. Dropbox) for this assignment. Your completed assignment should consist of a document (in hardcopy or a PDF for submission to D2L Dropbox) that contains all required components including analysis, results. figures, comments, etc. as well as the code you wrote and/or used. Also submit any Matlab code (m-files for script or function files) separately to the D2L Dropbox.*

Overall Instructions

- After reviewing the Overall Instructions, read through the statement of the Problem of this Assignment as detailed below.
- Think about and work through the assignment's design problem. This is one of only two design assignments for ECE 8600, so you will want to start early. Don't leave it to the night before the due date. If you start too late, you will find that your thinking time, your work time, or your write-up time may be compromised. In any case, start early to *think* about the problem(s) and how you might approach the design.
- You will find that your design problem will need some computer-based aids (e.g Matlab) to augment or produce your calculations, plots, etc. When your workings go beyond pen-and-paper analysis and illustrations, or hand calculations or rather straightforward calculator calculations, then you must indicate what aids you used, and provide the source code for the procedures employed. (In other words, when you use Matlab, write M-files and provide the source in your write-up).
- You may use reference material, as appropriate, provided you cite them in your write-up. Similarly, you may search out and use public domain data sets, as relevant, provided these are acknowledged and the source URLs and date of download cited.
- Write up your solution as completely as possible. Note that marks will be given for your approach and clarity as well as for the correctness of your solution. Your solution should read smoothly from beginning to end, like a professional report for the stated problem. However, there is no need to type-set it; if convenient, hand-written parts of your report, as well as hand-annotated plots, data tables, and/or source code, as suitable, are perfectly adequate, provided the flow is clear and the problem solution well-served.

- Note that **your solutions, figures, code, conclusions, and write-up must be your own work, unless there is specific code and/or data and/or idea(s) that you have re-used or modified from permitted sources, and that you cite clearly in your assignment report.**
- Submit a complete copy of your assignment write-up by the due date and time. You can submit one paper copy of your completed assignment. Or you can submit your completed assignment in one PDF that is submitted to the D2L Assignment Folder (i.e. Dropbox) for this assignment. Please include your Matlab code as text or as print-to-PDF directly in your assignment for ease of marking, as well as all output plots and values - but also upload your Matlab code as m-files to the Dropbox, so that they can be run as needed to understand what you did.

  Ensure that your design assignment solution is submitted exactly as you wish it to be marked.
- Your assigment will be marked using a grading rubric that reflects the relative weightings and work load of each problem and problem part.

Next

- The design assignment problem
- Technical notes of general use when working with DSP design problems.

The Design Assignment Problem *This assignment addresses the problem of designing and testing schemes (via a filter bank or via an FFT) to segment the DT frequency range into low and high frequencies.*

Preamble In many DSP applications there is the need to divide an input signal into adjacent but non-overlapping frequency bands. For example, it may be required to divide a DT signal $x(n)$ into its low and high frequencies, or perhaps to segment into three bands of the low-frequencies, mid-frequencies, and high frequencies, or any other desired segmentation of the frequency range of $x(n)$. The purpose of the frequency segmantation may be to process or code the signal differently according to its frequency bands. Subsequently, the bands may be restored and recombined to reconstruct the original signal.

Of course, it is difficult to do so exactly with real-world, non-ideal filters. This assignment problem investigates this important application of segmentation into two equal-width bands, the low frequencies $|\omega| \leq \pi/2$, and the high frequencies $\pi/2 < |\omega| \leq \pi$.

The design assignment has three parts, with each part investigating a different segmentation (and reconstruction) scheme. Schemes 1 and 2 are based on a two-band filter bank, and Scheme 3 on using the FFT. A few more notes on each before we start the problem proper.

Preamble for Schemes 1 and 2: Segmentation and reconstruction via a filter bank Consider a filter bank consisting of causal filters that can divide an input signal $x(n)$ into its low frequencies (i.e. $|\omega| \leq \pi/2$), and its high frequencies (i.e. $\pi/2 < |\omega| < \pi$). A filter bank is a set of filters implemented in parallel. For example, see the filters in the analysis part of the filter bank in DTSP3 Figure 4.44 (Figure included below.)

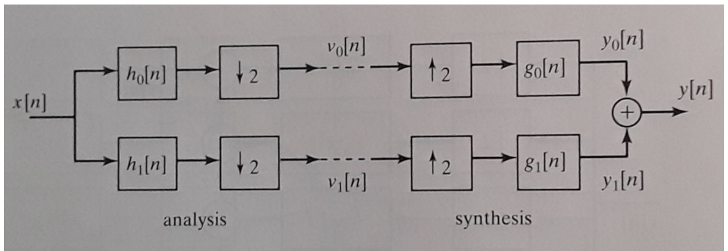

Figure 4.44 Two-channel analysis and synthesis filter bank.

The filter bank has two paths in parallel, the low-frequency path (top path in Figure) and the high-frequency path (bottom path in Figure). Each path has an analysis (or segmentation) part with a filter and a down-sampler block denoted by $\downarrow 2$, followed by an synthesis (or reconstruction) part with an upsampler block denoted by $\uparrow 2$ and a different filter. In between the analysis part and the synthesis part would be a coding scheme for transmission or storage of the signal, or some other processing of the signal bands.

In Schemes 1 and 2, we will be ignoring the downsampling and upsampling blocks (denoted by $\downarrow 2$ and $\uparrow 2$, respectively) as well as the coding or processing of the band signals between analysis and synthesis. Since $\downarrow 2$ and $\uparrow 2$ are exact inverses of each other[1], we can treat the two paths as being as follows, starting from the input signal $x(n)$:

$$\text{Top path:}\quad x(n) \rightarrow \boxed{h_0(n)} \rightarrow v_0(n) \rightarrow \boxed{g_0(n)} \rightarrow y_0(n)$$

$$\text{Bottom path:}\quad x(n) \rightarrow \boxed{h_1(n)} \rightarrow v_1(n) \rightarrow \boxed{g_1(n)} \rightarrow y_1(n)$$

with the overall output signal being:

$$y(n) = y_0(n) + y_1(n)$$

Scheme 1 will be based on using simple FIR filters for the analysis filters $h_0(n)$ and $h_1(n)$, and for synthesis filters $g_0(n)$ and $g_1(n)$.

Scheme 2 will be based on using IIR filters for the analysis filters $h_0(n)$ and $h_1(n)$, and for synthesis filters $g_0(n)$ and $g_1(n)$.

In all Schemes you will consider how effective the Scheme is by measuring how well the analysis filters $h_0(n)$ and $h_1(n)$ can segment an input signal $x(n)$ into its low and high frequencies.

While not required in this assignment, you could also ask the follow-on question about how effectively the analysis filters $g_0(n)$ and $g_1(n)$ can recover the original signal $x(n)$; that is how closely the output of the filter bank $y(n)$ reconstructs the input signal $x(n)$.

Preamble for Scheme 3: Segmentation and reconstruction via the FFT The preamble notes are included with the problem statement for Scheme 3 below.

Scheme 1. Consider the filter bank in the Figure above (i.e. DTSP3 Figure 4.44) based on low-order causal and FIR filters to divide an input signal $x(n)$ into its low frequencies (i.e. $|\omega| < \pi/2$), and its high frequencies (i.e. $\pi/2 < |\omega| < \pi$). This scheme uses the Haar first-order filters that are defined in the z-domain as:

$$H_0(z) = \frac{1}{2}\left(1 + z^{-1}\right), \quad H_1(z) = \frac{1}{2}\left(1 - z^{-1}\right)$$

(a) Determine and plot the magnitude and phase responses of the filters $H_0(\omega)$ and $H_1(\omega)$, and thus show that these filters are, respectively, predominantly low-pass and high-pass filters (albeit of low order).

(b) Next investigate the following two filters for a synthesis filter bank (as in DTSP3 Fig 4.44 but minus the downsample and upsample blocks denoted by $\downarrow 2$ and $\uparrow 2$, respectively):

$$G_0(z) = \frac{1}{2}(1 + z^{-1}), \quad G_1(z) = \frac{1}{2}(-1 + z^{-1})$$

[1]Regarding the downsampling and upsampling blocks denoted by $\downarrow 2$ and $\uparrow 2$, respectively, in DTSP3 Fig 4.44, you can understand that if the high-frequencies are removed from an input signal by the operation of a low-pass filter $H_0(\omega)$, that the low-band signal can then be downsampled and transmitted at a lower-rate, and later upsampled and interpolated (by the $G(\omega)$ filters) back to the original rate. We are not considering the downsampling and upsampling operations in this assigment, but you can consider them to have happened in between the $H(\omega)$ and the $G(\omega)$ filters, effectively cancelling each other out.

Again, determine and plot the magnitude and phase responses of these filters $G_0(\omega)$ and $G_1(\omega)$.

(c) Then, show that the synthesis filters $G(\omega)$ manage to "undo" the effects of the analysis $H(\omega)$ filters. That is, investigate and show the result of these filters in cascade with the $H(\omega)$ filters within the parallel filter bank, by considering the magnitude and phase of the overall frequency response:

$$H_{FB}(\omega) = H_0(\omega)G_0(\omega) + H_1(\omega)G_1(\omega)$$

- **Note:** You should find that the Haar filters form what is known as a perfect reconstruction filter bank, with at most a delay of a signal through the entire bank.

(d) Next, devise a method to test how well the Haar filters $H_0(\omega)$ and $H_1(\omega)$ perform in separating an input signal $x(n)$ into its low and high frequencies.

How to test the Scheme:

- The best way to do this is to set up test signals that have sinusoidal components in both the low and the high frequency ranges - say several frequencies in each of the low and high frequency range - then process them through the analysis part of the filter bank (See Fig 4.44), possibly one frequency at a time. That is, first an $x(n)$ with a single low-frequency sinusoid, then another, and so on, and then $x(n)$ with a single high-frequency sinusoid, etc.
  Select the frequencies you test across the bands, including near the edges of the bands as well as in the middle of the bands.
- Use SNR to measure how how well the filters of the **analysis** part of the filter bank do in separating the low and high components. That is, for a test sinusoid in the low frequency range, how much does it deviate from a gain of 1 through the low-pass filter, and how much does it deviate from a gain of 0 through the high-pass filter, and the reverse for a test signal in the high-frequency range.
- In this manner, use several such test signals through the analysis filter bank to test the filters for their ability to segment signals $x(n)$ into low and high frequency bands.

Notes for Scheme1:

- *In general*, SNR measures the ratio of the energy in an ideal or desired signal $x(n)$ (i.e. the “signal” of the SNR) to the energy in the error difference (or “noise”) between the signal you actually obtained, i.e. $\hat{x}(n)$, and the ideal $x(n)$. Thus, we can use the following form of the SNR:

$$SNR = 10\log_{10}\frac{\sum_{n=1}^{N}|x(n)|^2}{\sum_{n=1}^{N}|x(n)-\hat{x}(n)|^2}$$

where N is the length of the signals. Clearly, it is desired that an SNR should be as high as possible.

Thus, *for this problem*, if a test input signal to the filter bank is of the form $x(n) = x_L(n) + x_H(n)$, where $x_L(n)$ contains only low-frequencies and $x_H(n)$ contains only high frequenices, then the SNR for the low-frequency filter,

$$x(n) \rightarrow \boxed{h_L(n)} \rightarrow y_L(n) \text{ or in the Figure } v_0(n)$$

would be

$$SNR_L = 10\log_{10}\frac{\sum_{n=1}^{N}|x_L(n)|^2}{\sum_{n=1}^{N}|x_L(n)-v_0(n)|^2} \quad \text{or} \quad SNR_L = 10\log_{10}\frac{\sum_{n=n_T}^{N}|x_L(n-D)|^2}{\sum_{n=n_T}^{N}|x_L(n-D)-v_0(n)|^2}$$

where D accounts for delay-alignment and n_T accounts for the filter transient (see next notes).

- **Remember to delay-align** the input and output signals in the SNR calculations so that your comparisons are most valid. Why? Because a causal filter always causes a delay D in the output signal $y(n)$ relative to the input signal $x(n)$, you need to advance the ideal signal before comparing with the filtered signals. (Read the Technical note for more details).
- Also, for computing the SNR, **remember to pick your segments to compare** past the start-up transient of the filters, i.e. $n \geq n_T$. You need to have enough remaining points for comparison after the delay-alignment. For this reason you need to consider how long your test segment signals need to be.
- The reason for start with the Haar filters (not very good at segmentation, although good at overall reconstruction) is that they are very simple filters and you know what the analysis Haar filters should be doing, or can work out by hand what they should be doing. This can be used to check that your testing procedure is operating correctly.

Scheme 2. Next, design an analysis filter-bank consisting of higher-order causal, IIR filters $H_0(z)$ and $H_1(z)$ intended to segment an input signal $x(n)$ into its low frequencies (i.e. $|\omega| \leq \pi/2$), and its high frequencies (i.e. $\pi/2 < |\omega| < \pi$)

(a) Design your IIR filters using the methods in Module 6, based on an analog Butterworth filter. Apply the design specification that the analysis filters $H_0(z)$ and $H_1(z)$ taken together should be implemented by difference equations with at most 40 coefficients (i.e. in their b and a vectors).

Using Matlab, plot your two designed filters $H_0(\omega)$ and $H_1(\omega)$ in magnitude and phase.

To do the filter designs:

- See the Note below on understanding the tolerance requirements.
- Read the Note below for how to design a High-pass filter from a Low-pass filter.

- Note that this design approach will not be able to design perfect reconstruction filters (as with the Haar filters), but higher-order filters can produce much better frequency segmentation between the low and high frequencies.

(b) Next, use the same approach you devised in Scheme 1 to to test how well your IIR filters perform in separating input signals into low and high frequencies.

That is, put various test signals through the filter bank, and calculate the SNR (signal-to-noise ratio) for each of the top and bottom analysis filters.

Remember to delay-align the input and output signals in the SNR calculations so that your comparisons are most valid.

(c) Discuss your results from Schemes 1 and 2. How well does your designed Scheme 2 perform? How well does the Haar-filter Scheme 1 perform? What have you learned about real-world filtering, and about the problem of frequency band separation?

Notes for Scheme 2:

- To design the high-pass filter $H_1(\omega)$, one approach is to design a low-pass filter $H_{LP}(\omega)$ with the same pass-band width, then shift $H_{LP}(\omega)$ in frequency by π. This approach works well in DT, since all DTFTs are periodic, with period 2π and so a shift by π in either direction exchanges low and high frequencies.

  However, while shifting a frequency response by π, i.e. $H_{HP}(\omega) = H_{LP}(\omega - \pi)$ is relatively straightforward for an FIR filter, it is less so for an IIR filter since to effect the transformation of $H_{LP}(z)$ to $H_{HP}(z)$ means rotating the z-plane, for example by a variant of the bilinear transformation (as in Problem 2 of the Module 6 Problem Set).

  For this reason, to design the highpass filter $H_1(\omega)$ for the analysis filter bank, you can use the Matlab function **butter** with the option 'high' set. In the command window, type >> **help butter** for more details.

- You will notice that no tolerance constraints are provided (i.e. $\delta_1, \delta_2, \omega_p, \omega_s$) as such to constrain the filter design. The total filter length will set one of these, as apportioned to the filters in the filter bank; the choice of design method is another. To some extent, if the error (i.e. sometimes called ripple) is kept small, the ripple error (i.e. δ_1, δ_2) need not be unduly significant, as each filter band (low or high) in the bank can be seen to suffer equally from the errors. A best case scenario is when the pass and stop errors can "cancel out" between adjacent filters in the bank. This thinking is the motivation behind filter bank design methods based on the commonly used quadrature mirror filters (QMF). The Haar filters satisfy these criteria, despite their obvious deficiences, individually, as band-defining filters. QMF filter banks are beyond the scope of this course, but you may want to keep this thinking in mind as you do this problem.

Scheme 3. Finally, since the DFT / FFT gives us an analysis of a finite signal segment $x(n)$, $0 \le n \le N-1$ into its frequency components, it may seem to be an obvious strategy to simply divide the DFT/FFT $X(k)$ according to whether k indexes a low or high frequency.

Worth noting are the following elementary facts about the DFT/FFT:

- The DFT (discrete Fourier transform) $X(k), 0 \le k \le N-1$ refers to the transform of a finite signal segment $x(n)$, $0 \le n \le N-1$, while the FFT (fast Fourier transform) refers to an efficient algorithm to compute the DFT. For the purpose of this problem, we can refer to the DFT and FFT interchangeably in terms of the output they produce, so we will refer to the FFT. You can compute the FFT for any signal length N, i.e. for $x(n)$, $0 \le n \le N-1$, but the FFT will be most efficient if N is a power of 2, i.e. $N = 2^\nu$.
- The Matlab function

  `>>` **X = fft(x) ;**

  returns the (generally complex-valued) FFT $X(k), 0 \le k \le N-1$ of the input signal segment $x(n)$, $0 \le n \le N-1$, where k indexes into the DT baseband $[0, 2\pi]$ via a uniform sampling of the DT frequency band $[0, 2\pi)$, with a spacing of $\Delta\omega = 2\pi/N$, as (with subscripts added to X for clarity):

  $$X_{FFT}(k) = X_{DTFT}(\omega)\big|_{\omega=\frac{2\pi}{N}k} \qquad 0 \le k \le N-1$$

  where $X_{DTFT}(\omega)$ is the DTFT of the signal segment $x(n)$, $0 \le n \le N-1$.
- For example, if $N = 64$, then when $k = 1$, the FFT value $X_{FFT}(k = 1)$ is the DTFT value

  $$X_{DTFT}\left(\omega = \frac{2\pi}{64}\right) = X_{DTFT}(0.03125\pi)$$

  and when $k = 63$, the FFT value $X_{FFT}(k = 63)$ is the DTFT value

  $$\begin{aligned} X_{DTFT}\left(\omega = \frac{2\pi}{64}63\right) &= X_{DTFT}(01.96875\pi) \\ &= X_{DTFT}(-0.03125\pi) \qquad \text{(by periodicity of the DTFT)} \end{aligned}$$

  The same analysis can be applied to all other k's in the set $0 \le k \le N-1$, up to $k = N/2$ so that the FFT value $X_{FFT}(k = 32)$ is the DTFT value $X_{DTFT}(\omega = \pi)$.

  By the same analysis the DTFT values $X_{DTFT}(\omega = \pi/2)$ is found to be the FFT value for $k = N/4$, and the DTFT values $X_{DTFT}(\omega = -\pi/2) = X_{DTFT}(\omega = 3\pi/2)$ is found to be the FFT value for $k = 3N/4$.
- Thus, the set of FFT values $X_{FFT}(k)$ for $k = 0, 1, \ldots N/4$ and $k = 3N/4, \ldots N-1$ pull out the DTFT values of $X_{DTFT}(\omega)$ that are computed by the DFT for low-frequency ω values in the range $|\omega| \le \pi/2$ or $-\pi/2 \le \omega \le \pi$.

With these simple facts about the DFT/FFT (which we will explore more in Modules 7 and 8 as time permits), implementing the segmentation of a signal segment $x(n)$, $0 \le n \le N-1$, via the FFT is a simple matter of FFT indexing, followed by an inverse FFT to reconstruct the band in the time domain. For example, to implement the analysis part of the top or low-frequency path:

$$\text{Top path:} \quad x(n), 0 \le n \le N-1 \rightarrow \boxed{\text{FFT}_N} \rightarrow X(k), 0 \le k \le N-1$$

$$X(k), 0 \le k \le N-1 \rightarrow \boxed{\text{Zero } X(k) \text{ for } k\text{'s of high-frequencies}} \rightarrow V_0(k), 0 \le k \le N-1$$

$$V_0(k), 0 \le k \le N-1 \rightarrow \boxed{\text{inverse FFT}_N \text{ (ifft)}} \rightarrow v_0(n), 0 \le n \le N-1$$

And similarly for the Bottom path to obtain the high-frequency signal $v_1(n)$.

(a) Implement the above Scheme 3 and test it on the same or similar test signals you used in Schemes 1 and 2, to see how well this Scheme 3 performs in separating input signals into low and high frequencies.

- A Practical Note: The **ifft** function in Matlab may return a complex-valued array, due to accumulated round-off error remaining in the imaginary part. For this reason, you may need to extract the real-part of the array after the **ifft**.

(b) Comment on the FFT approach relative to the filter-bank approach in segmenting a signal into its low and high frequencies.

Technical Notes[2] The problems in the Design Assignments for ECE 8600 consider both the design and testing of filters and other DSP algorithms. Each Design Assignment problem will mention specific technical or design aspects pertinent to their problem statements, but since many problems in DSP have the same techical issues, they are emphasized in Technical Notes such as the one that follows.

In testing filters, it is often an important and insightful approach to process test signals through a system you have designed, analyzed or optimized, and then to quantify the error between the results and the theoretical ideal output. There are two things in particular that you need to know to do this correctly.

1. **Quantifying error:** You can quantify the error in processing through a filter via the difference between an actual output signal and its ideal or desired output signal, using the mean-squared error (MSE) and/or the signal-to-noise (SNR) ratio.

   The SNR is based on a ratio and, as such, can be considered to be a "normalized" error measure. For this reason, the SNR is often considered to be a better than MSE. However, the MSE is also useful, and used in the literature. For these reasons, I recommend that both error measures be computed and tracked or tabulated.

[2]The following notes may, or may not, be necessary. They are included here as they are generally important when working with real filters, and we'll encounter the need for these points in our class work, as well as in the the Design Assignment problems.

Generic definitions of both the SNR and MSE follow.

MSE: Let the ideal or error-free or desired signal be $y_{ideal}(n)$ and the actual or processed or error-containing signal be $y_{actual}(n)$. Then the equation for MSE is:

$$MSE = \frac{1}{N}\sum_{n=1}^{N} |y_{ideal}(n) - y_{actual}(n)|^2 \tag{1}$$

where N is the number of samples in the signal segments.
SNR: A second commonly-used error measure is the signal-to-noise (SNR) ratio. SNR is based on the ratio A/B between A) the energy in the ideal or desired signal, here $y_{ideal}(n)$, i.e. the "signal" as so-called of the SNR, and B) the energy in the error difference (or "noise") between the signal you actually obtained, here $y_{actual}(n)$, and the ideal, here $y_{ideal}(n)$.

Thus, we can use the following form of the SNR:

$$SNR = 10\log_{10}\frac{\sum_{n=1}^{N}|y_{ideal}(n)|^2}{\sum_{n=1}^{N}|y_{ideal}(n) - y_{actual}(n)|^2} \text{ (in dB)} \tag{2}$$

The definition of SNR in Equation (7) is a decibel version, and this version is commonly used. However, you can also use the version of SNR that is the actual ratio A/B of Signal to Noise, i.e. Equation (7) without the $10\log_{10}$.

More information on the MSE and SNR can be found in the problem statements themselves, as well as in many books and web sources.

2. **Accounting for causal delay:** In order for error quantification to be meaningful, you will often need to account for the fact that causal filters cause delays relative to ideal zero-phase filters. Plus, sometimes it's necessary to account for delay differences between filters, e.g. since higher-order filters cause more delay than lower-order filters for the same filtering problem. For MSE and SNR error comparisons to be valid, you need to address these issues.

   To do this, you need to work out how much shift you need to apply in time between each actual output and the ideal zero-phase ideal output. And, to complicate matters, the shift due to phase delay need not be an integer, which means that you may need to interpolate off the n grid of one of the signals (i.e. actual or ideal) in order to align them for fair measures of comparison.

   It is worth noting that Matlab has a built-in function **finddelay** that you can use to estimate the delay between two signals. In most filtering cases (i.e. other than an integer delay between the arrays), the result from **finddelay** is an estimate based on the signals' autocorrelations. For these reasons, I recommend that you check the result(s) given by **finddelay** against what your filter's time delay is known to be**,** and as needed**,** code your own theoretical knowledge of the filter's delay into your calculation of error - this can be important and useful with some FIR filters. It is also easier to do theoretically with FIR filters.

   More information on dealing with causal delay will be discussed in class.

3. **Dividing up the DT frequency range, and / or shifting frequencies.** The design assignment problems may focus on the DT frequencies, either by dividing up the frequency range in defining a filter bank, or in using upsampling (or downsampling) to shift / compress/ stretch one frequency interval into another. These are both interesting topics, about which much could be said. This general note is kept brief here, as each problem has its own notes about what you need to know about dividing or shifting frequencies in order to do that particular problem.

4. **DFT via the FFT:** The DFT may feature in some aspects of this assignment. You can assume in all such cases that the FFT would be used in actual implementations, while the DFT informs your choices and designs. Recall that an FFT is most efficient when the length N of the signal segment $x(n)$, $0 \leq n \leq N-1$ is a power of 2, i.e. $N = 2^{\nu}$; however, the DFT can be computed for any $N > 0$, (although the computational complexity rises with N.)

   Specific details are provided with each individual problem, as needed.

5. **Performance Analysis:** One feature of interest with FFT based algorithms is their computational complexity. Often you can work out upper bounds or $O(\cdot)$ trends for such algorithms analytically. But sometimes you would simply like to compare algorithms on their run time or use of cpu. With Matlab, the latter is rather problematic as Matlab attempts to optimize computations as much as possible which can cloud comparisons. So while you can use the Matlab built-in function **cputime**, Matlab recommends tracking elapsed time using two functions **tic** and **toc**. See Matlab help for these functions, or search Matlab help for a technical note on "Analysing your Program's Performance" (also available on-line at www.mathworks.com/help/matlab/matlab_prog/analyzing-your-programs-performance.html)

6. Other hints and Matlab functions are provided with the individual problems, as needed.