Signals and Systems

Lecture4: Convolution, Frequency Responses, and Fourier Transform

Instructor: Prof. Xiaoming Chen
Zhejiang University

05/23/2024

Outline

Discrete-Time Convolution

The MATLAB function **conv** computes the convolution sum

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m] \tag{2.3}$$

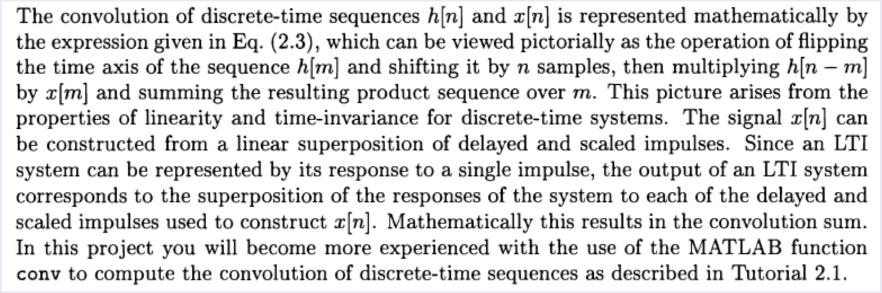
assuming that $x[n]$ and $h[n]$ are finite-length sequences. If $x[n]$ is nonzero only on the interval $n_x \le n \le n_x + N_x - 1$ and $h[n]$ is nonzero only on the interval $n_h \le n \le n_h + N_h - 1$, then $y[n]$ can be nonzero only on the interval

$$(n_x + n_h) \le n \le (n_x + n_h) + N_x + N_h - 2\,, \tag{2.4}$$

meaning that **conv** need only compute $y[n]$ for the $N_x + N_h - 1$ samples on this interval. If **x** is an N_x–dimensional vector containing $x[n]$ on the interval $n_x \le n \le n_x + N_x - 1$ and **h** is an N_h–dimensional vector containing $h[n]$ on the interval $n_h \le n \le n_h + N_h - 1$, then **y=conv(h,x)** returns in **y** the $N_x + N_h - 1$ samples of $y[n]$ on the interval in Eq. (2.4). However, **conv** does not return the indices of the samples of $y[n]$ stored in **y**, which makes sense because the intervals of **x** and **h** are not input to **conv**. Instead, you are responsible for keeping track of these indices, and will be shown how to do this in this tutorial.

Discrete-Time Convolution

Problem1



The convolution of discrete-time sequences $h[n]$ and $x[n]$ is represented mathematically by the expression given in Eq. (2.3), which can be viewed pictorially as the operation of flipping the time axis of the sequence $h[m]$ and shifting it by n samples, then multiplying $h[n-m]$ by $x[m]$ and summing the resulting product sequence over m. This picture arises from the properties of linearity and time-invariance for discrete-time systems. The signal $x[n]$ can be constructed from a linear superposition of delayed and scaled impulses. Since an LTI system can be represented by its response to a single impulse, the output of an LTI system corresponds to the superposition of the responses of the system to each of the delayed and scaled impulses used to construct $x[n]$. Mathematically this results in the convolution sum. In this project you will become more experienced with the use of the MATLAB function `conv` to compute the convolution of discrete-time sequences as described in Tutorial 2.1.

Discrete-Time Convolution

Problem1

In these problems, you will define some discrete-time signals and the impulse responses of some discrete-time LTI systems. Then the output of the LTI systems can be computed using `conv`.

(a). Since the MATLAB function `conv` does not keep track of the time indices of the sequences that are convolved, you will have to do some extra bookkeeping in order to determine the proper indices for the result of the `conv` function. For the sequences $h[n] = 2\delta[n+1] - 2\delta[n-1]$, and $x[n] = \delta[n] + \delta[n-2]$ construct vectors **h** and **x**. Define $y[n] = x[n] * h[n]$ and compute `y=conv(h,x)`. Determine the proper time indexing for **y** and store this set of time indices in the vector **ny**. Plot $y[n]$ as a function of n using `stem(ny,y)`.

(b). Consider two finite-length sequences $h[n]$ and $x[n]$ that are represented in MATLAB using the vectors **h** and **x**, with corresponding time indices given by `nh=[a:b]` and `nx=[c:d]`. The function call `y=conv(h,x)` will return in the vector **y** the proper sequence values of $y[n] = h[n] * x[n]$, however you must determine a corresponding set of time indices **ny**. To help you construct **ny**, consider the sequence $h[n] = \delta[n-a] + \delta[n-b]$ and $x[n] = \delta[n-c] + \delta[n-d]$. Determine analytically the convolution $y[n] = h[n] * x[n]$. From your answer, determine what **ny** should be in terms of a, b, c, and d. To check your result, verify that $y[n]$ is of length $M + N - 1$ when $a = 0$, $b = N - 1$, $c = 0$, and $d = M - 1$.

Discrete-Time Convolution

Problem1

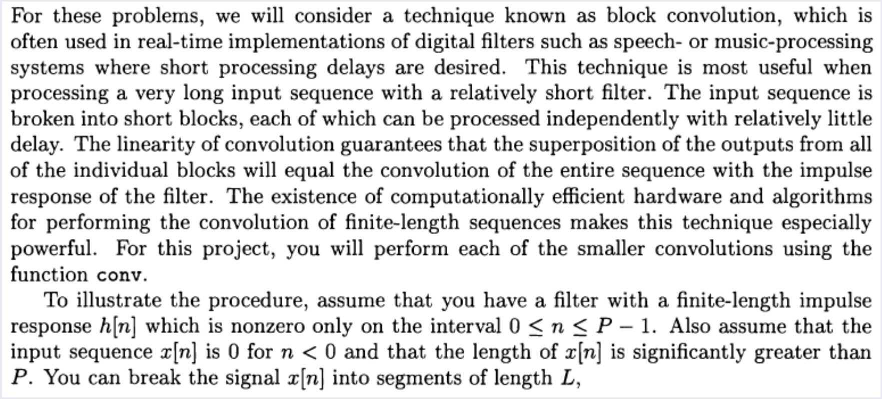
(c). Consider an input $x[n]$ and unit impulse response $h[n]$ given by

$$x[n] = \left(\frac{1}{2}\right)^{n-2} u[n-2],$$
$$h[n] = u[n+2].$$

If you wish to compute $y[n] = h[n] * x[n]$ using `conv`, you must deal appropriately with the infinite length of both $x[n]$ and $h[n]$. Store in the vector **x** the values of $x[n]$ for $0 \leq n \leq 24$ and store in the vector **h** the values of $h[n]$ for $0 \leq n \leq 14$. Now store in the vector **y** the result of the function call `conv(h,x)`. Since you have truncated both $h[n]$ and $x[n]$, argue that only a portion of the output of `conv` will be valid. Specify which values in the output are valid and which are not. Determine the values of the parameters a, b, c, and d such that `nx=[a:b]` and `nh=[c:d]`, and use your answer from Part (b) to construct the proper time indices for **y**. Plot $y[n]$ using **stem**, and indicate which values of $y[n]$ are correct, and which values are invalid. Be sure to properly label the time axis for $y[n]$.

Discrete-Time Convolution

Problem1

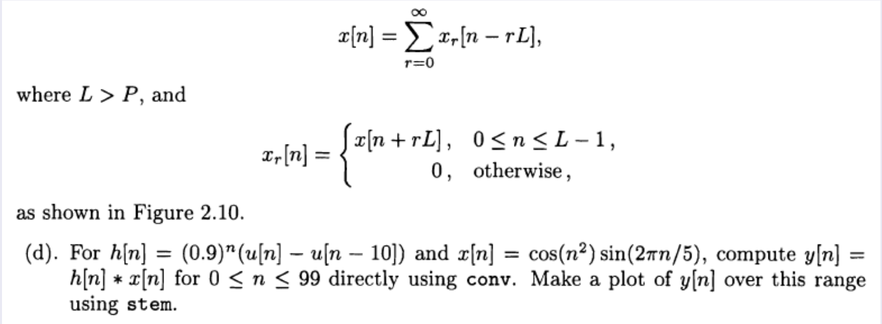


For these problems, we will consider a technique known as block convolution, which is often used in real-time implementations of digital filters such as speech- or music-processing systems where short processing delays are desired. This technique is most useful when processing a very long input sequence with a relatively short filter. The input sequence is broken into short blocks, each of which can be processed independently with relatively little delay. The linearity of convolution guarantees that the superposition of the outputs from all of the individual blocks will equal the convolution of the entire sequence with the impulse response of the filter. The existence of computationally efficient hardware and algorithms for performing the convolution of finite-length sequences makes this technique especially powerful. For this project, you will perform each of the smaller convolutions using the function `conv`.

To illustrate the procedure, assume that you have a filter with a finite-length impulse response $h[n]$ which is nonzero only on the interval $0 \leq n \leq P-1$. Also assume that the input sequence $x[n]$ is 0 for $n < 0$ and that the length of $x[n]$ is significantly greater than P. You can break the signal $x[n]$ into segments of length L,

Discrete-Time Convolution

Problem1



$$x[n] = \sum_{r=0}^{\infty} x_r[n - rL],$$

where $L > P$, and

$$x_r[n] = \begin{cases} x[n + rL]\,, & 0 \leq n \leq L - 1\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$$

as shown in Figure 2.10.

(d). For $h[n] = (0.9)^n(u[n] - u[n - 10])$ and $x[n] = \cos(n^2)\sin(2\pi n/5)$, compute $y[n] = h[n] * x[n]$ for $0 \leq n \leq 99$ directly using `conv`. Make a plot of $y[n]$ over this range using `stem`.

Discrete-Time Convolution

Problem1

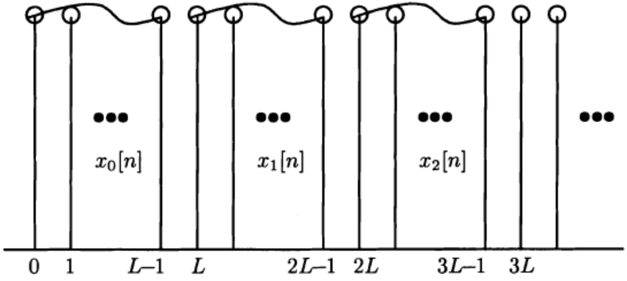


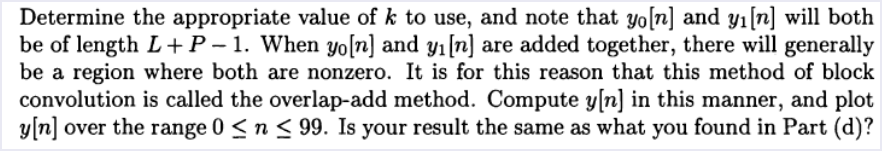
Figure 2.10. Block decomposition of $x[n]$.

(e). For $L = 50$, break the sequence $x[n]$ into two sequences, each of length 50. Compute $y_0[n] = h[n] * x_0[n]$, and $y_1[n] = h[n] * x_1[n]$, where $x_0[n]$ contains the first 50 samples of $x[n]$ and $x_1[n]$ contains the second 50 samples of $x[n]$. The form of the output $y[n]$ is given by

$$y[n] = x[n] * h[n] = y_0[n] + y_1[n-k].$$

Discrete-Time Convolution

Problem1



Determine the appropriate value of k to use, and note that $y_0[n]$ and $y_1[n]$ will both be of length $L+P-1$. When $y_0[n]$ and $y_1[n]$ are added together, there will generally be a region where both are nonzero. It is for this reason that this method of block convolution is called the overlap-add method. Compute $y[n]$ in this manner, and plot $y[n]$ over the range $0 \leq n \leq 99$. Is your result the same as what you found in Part (d)?

Outline

Image Processing with One-Dimensional Filters

A two-dimensional (2D) discrete-time signal $x[m,n]$ has two index variables rather than one. An image is a 2D discrete-time signal which is nonzero only over a finite region, e.g., the N-by-N square $0 \leq m,n \leq N-1$. For such images, m corresponds to the row index while n corresponds to the column index. Two 32-by-32 pixel images are illustrated in Figure 6.3. (A pixel is the name for a sample of an image, and is short for *picture element*.)

Many of the time-domain and frequency-domain tools used for designing and analyzing one-dimensional (1D) signals apply equally well to 2D signals.[1] In particular, a widely-used method for filtering 2D signals is to process the signal with a cascade of two 1D filters, one filter for each direction. The cascade of these two 1D filters, called a separable 2D filter, is given by the following two steps: (i) for each n compute the output $z[m,n]$ of the LTI

Image Processing with One-Dimensional Filters

system which satisfies

$$\sum_{k=-K}^{K} a_k z[m-k,n] = \sum_{j=-K}^{K} b_j x[m-j,n]\,; \tag{6.5}$$

(ii) for each m compute the output $y[m,n]$ of the LTI system which satisfies

$$\sum_{k=-K}^{K} c_k y[m,n-k] = \sum_{j=-K}^{K} d_j z[m,n-j]\,. \tag{6.6}$$

The output $y[m,n]$ is the filtered version of $x[m,n]$.

Note that both the column-wise filter defined by the coefficients a_k and b_j and the row-wise filter defined by the coefficients c_k and d_j are noncausal. In the context of image processing, causality is not a constraint, since one is often given the entire image before it is to be processed.

Image Processing with One-Dimensional Filters

In this exercise, you will lowpass filter an image using a cascade of two 1D filters. For simplicity, the two cascaded 1D filters are identical, i.e., $c_k = a_k$ and $d_j = b_j$. You will see that the time-domain and frequency-domain tools you have learned for 1D signals can provide considerable insight into 2D filtering.

For this exercise, you will need to load the data file `plus.mat`, which is provided in the Computer Explorations Toolbox. If this file is already in your MATLABPATH, you can load the data by typing `load plus`. If loaded correctly, your workspace will then contain a matrix `x`, which contains the image to be processed in this exercise, and a noise-corrupted version of `x` in the matrix `xn`, which will be filtered in the Advanced Problems. To display the image, type

```
>> colormap(gray);
>> image(64*x);
```

Your image should be identical to the one in Figure 6.3(a). The command `colormap(gray)` is used to select the mapping from the values of `x` to the colors displayed by `image`. Also, because the colormap has 64 color levels, the input to `image` is scale by 64.

Image Processing with One-Dimensional Filters

Tutorial: freqz

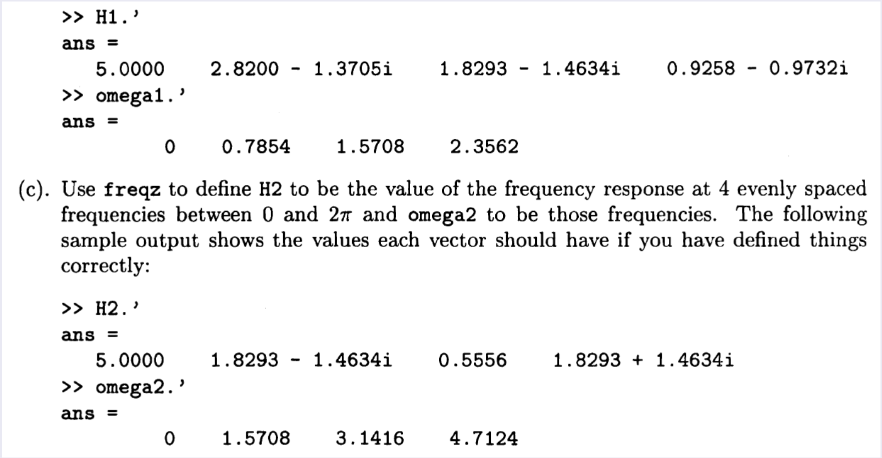
The signals $e^{j\omega n}$ are eigenfunctions of LTI systems. For each value of ω the frequency response $H(e^{j\omega})$ is the eigenvalue of the LTI system for the eigenfunction $e^{j\omega n}$; when the input sequence is $x[n] = e^{j\omega_0 n}$, the output sequence is $y[n] = H(e^{j\omega_0})e^{j\omega_0 n}$. For a causal LTI system described by a difference equation, the command `[H omega]=freqz(b,a,N)` computes the frequency response $H(e^{j\omega})$ at N evenly spaced frequencies between 0 and π, i.e., $\omega_k = (\pi/N)k$ for $0 \le k \le N-1$. The coefficient vectors `a` and `b` specify the difference equation using the same format as Eq. (2.8) in the `filter` tutorial. For the command above, `freqz` returns $H(e^{j\omega_k})$ in `H` and the frequencies ω_k in `omega`. Including the `'whole'` option as `[H omega]=freqz(b,a,N,'whole')` computes the samples of the frequency response $H(e^{j\omega})$ at N evenly spaced frequencies from 0 to 2π, $\omega_k = (2\pi/N)k$ for $0 \le k \le N-1$.

(a). Define `a1` and `b1` to describe the causal LTI system specified by the difference equation $y[n] - 0.8y[n-1] = 2x[n] - x[n-2]$.

(b). Use `freqz` with the coefficients from Part (a) to define `H1` to be the value of the frequency response at 4 evenly spaced frequencies between 0 and π and `omega1` to be those frequencies. The following sample output shows the values each vector should have if you have defined things correctly:

Image Processing with One-Dimensional Filters

Tutorial: freqz



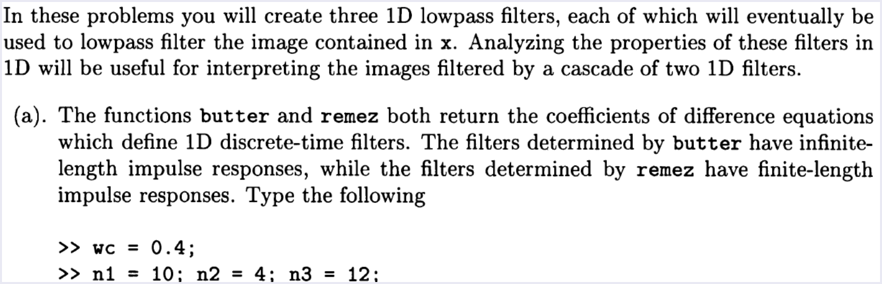
```
>> H1.'
ans =
   5.0000    2.8200 - 1.3705i    1.8293 - 1.4634i    0.9258 - 0.9732i
>> omega1.'
ans =
         0    0.7854    1.5708    2.3562
```

(c). Use **freqz** to define H2 to be the value of the frequency response at 4 evenly spaced frequencies between 0 and 2π and **omega2** to be those frequencies. The following sample output shows the values each vector should have if you have defined things correctly:

```
>> H2.'
ans =
   5.0000    1.8293 - 1.4634i    0.5556    1.8293 + 1.4634i
>> omega2.'
ans =
         0    1.5708    3.1416    4.7124
```

Image Processing with One-Dimensional Filters

Problem 2



In these problems you will create three 1D lowpass filters, each of which will eventually be used to lowpass filter the image contained in **x**. Analyzing the properties of these filters in 1D will be useful for interpreting the images filtered by a cascade of two 1D filters.

(a). The functions **butter** and **remez** both return the coefficients of difference equations which define 1D discrete-time filters. The filters determined by **butter** have infinite-length impulse responses, while the filters determined by **remez** have finite-length impulse responses. Type the following

```
>> wc = 0.4;
>> n1 = 10; n2 = 4; n3 = 12;
```

Image Processing with One-Dimensional Filters

Problem 2

```
>> [b1,a1] = butter(n1,wc);
>> a2 = 1; b2 = remez(n2,[0 wc-0.04 wc+0.04 1],[1 1 0 0]);
>> a3 = 1; b3 = remez(n3,[0 wc-0.04 wc+0.04 1],[1 1 0 0]);
```

where **wc** is the cutoff frequency of each filter (normalized by π) and **n1**, **n2**, and **n3** are the orders of the three filters.

(b). Use **freqz**, which is described in Tutorial 3.2, to plot the magnitude and phase of each of the three 1D filters determined in Part (a). As a check, verify that **wc** is the approximate cutoff frequency of each filter. Which filters have linear phase?

(c). Use **filter** to determine the step response of each filter for `n=[0:20]`. Plot each of these step responses. Which step response has the largest overshoot, i.e., the difference between the steady-state value and the maximum value of the step response? You will see that large overshoots or ringing in the step response can lead to undesirable artifacts in the filtered images.

Image Processing with One-Dimensional Filters

Problem 2

For image processing applications, noncausal filters can be used and are in fact desired. One reason for using noncausal filters is that the contents of an image should not be shifted in location after processing. In the following problems, you will first learn how to implement 1D noncausal filters using `filter`. Then you will write an M-file which will repeatedly call `filter` to process the columns and rows of the image stored in `x`.

The function `filter` assumes that the filter coefficients stored in the vectors `a` and `b` correspond to a causal filter. For example, if `x16` contains the column of the image for $n = 16$, i.e., $x[m, 16]$ for $0 \leq m \leq N-1$, then `y16=filter(b,a,x16)` stores in `y16` the output of the causal filter

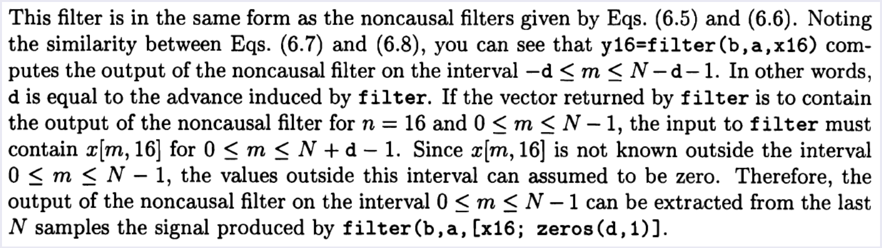
$$\sum_{k=0}^{\texttt{2*d}} \texttt{a}(k+1)\, y[m-k, 16] = \sum_{j=0}^{\texttt{2*d}} \texttt{b}(j+1)\, x[m-j, 16] \tag{6.7}$$

on the interval $0 \leq m \leq N-1$, where the vectors `a` and `b` have `2*d+1` elements. For this exercise, however, you will need to implement the following noncausal filter using `filter`:

$$\sum_{k=-\texttt{d}}^{\texttt{d}} \texttt{a}(k+1+\texttt{d})\, y[m-k, 16] = \sum_{j=-\texttt{d}}^{\texttt{d}} \texttt{b}(j+1+\texttt{d})\, x[m-j, 16]\,. \tag{6.8}$$

Image Processing with One-Dimensional Filters

Problem 2



This filter is in the same form as the noncausal filters given by Eqs. (6.5) and (6.6). Noting the similarity between Eqs. (6.7) and (6.8), you can see that `y16=filter(b,a,x16)` computes the output of the noncausal filter on the interval $-\mathtt{d} \leq m \leq N-\mathtt{d}-1$. In other words, `d` is equal to the advance induced by `filter`. If the vector returned by `filter` is to contain the output of the noncausal filter for $n = 16$ and $0 \leq m \leq N-1$, the input to `filter` must contain $x[m, 16]$ for $0 \leq m \leq N + \mathtt{d} - 1$. Since $x[m, 16]$ is not known outside the interval $0 \leq m \leq N - 1$, the values outside this interval can assumed to be zero. Therefore, the output of the noncausal filter on the interval $0 \leq m \leq N-1$ can be extracted from the last N samples the signal produced by `filter(b,a,[x16; zeros(d,1)]`.

Image Processing with One-Dimensional Filters

Problem 2

(d). Create the column vector **`x16=x(:,16)`**, which provides a 1D signal defined on the interval $0 \le m \le 31$. Use **`filter`** to compute the output of the noncausal filter given by Eq. (6.8) for **`a=a1`** and **`b=b1`**, assuming the input is given by **`x16`** for $0 \le m \le 31$ and is zero elsewhere. Store in **`y16`** the filter response on the interval $0 \le m \le 31$. Plot **`x16`** versus **`y16`**. If **`y16`** is computed correctly, the discontinuities in **`x16`** line up with the "smoothed" discontinuities in **`y16`**.

(e). For filters two and three, determine the response of these 1D filters to the column vector **`x16`**. For each filter, you will have to append zeros to **`x16`** before calling **`filter`**. Again plot the responses versus **`x16`** to ensure that the discontinuities are aligned and there is no delay or advance in your filter implementation.

(f). Write an M-file for the function **`filt2d`** which noncausally filters the 2D image stored in the matrix **`x`**. The first line of the your M-file **`filt2d.m`** should read

Image Processing with One-Dimensional Filters

Problem 2

```
function y = filt2d(b,a,d,x)
```

The vectors **a** and **b** contain the coefficients of the 1D filter and **d** contains the "delay" associated with a causal implementation of the filter. (Note that **d** is just **n/2**, where **n** is the order of the 1D filter). Your M-file should essentially consist of two steps: (i) filtering each column of an N-by-N matrix **x** and storing the results in an N-by-N matrix **z**, followed by (ii) filtering each row of **z**. Note that you will have to append an appropriate number of zeros to each column of **x** and each row of **z** before filtering. The matrix **y** returned by **2dfilt** should have the same dimensions as **x**.

(g). Use **image** to display the filtered image given by **filt2d** for each of the three filters. (Remember to scale the input to **image** by 64.) Make sure that the image has not been delayed, i.e., the filtered "plus" contained in **y** should have the same center location as the "plus" in **x**.

(h). Which filter leads to more distortion in the shape of the original image? Pay attention to any destruction of the symmetries present in the input image. This distortion is due in part to the nonlinearity of the phase in one of the filters.

Outline

Tutorial: Computing the Discrete-Time Fourier Series with fft

A signal $x[n]$ with fundamental period N, the DTFS synthesis and analysis equations are given by

$$x[n] = \sum_{k=0}^{N-1} a_k e^{jk(2\pi/N)n} \tag{3.1}$$

and

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk(2\pi/N)n} \tag{3.2}$$

Tutorial: Computing the Discrete-Time Fourier Series with fft

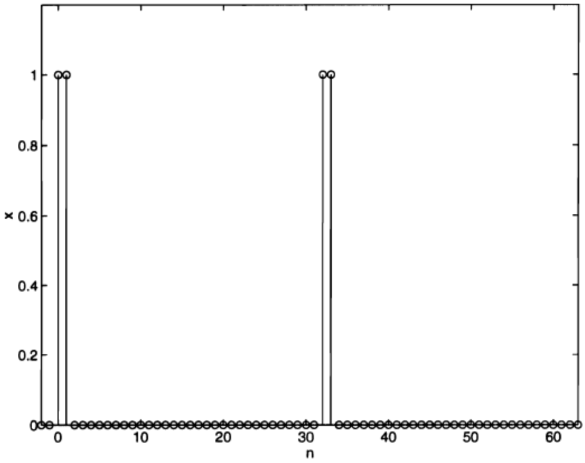
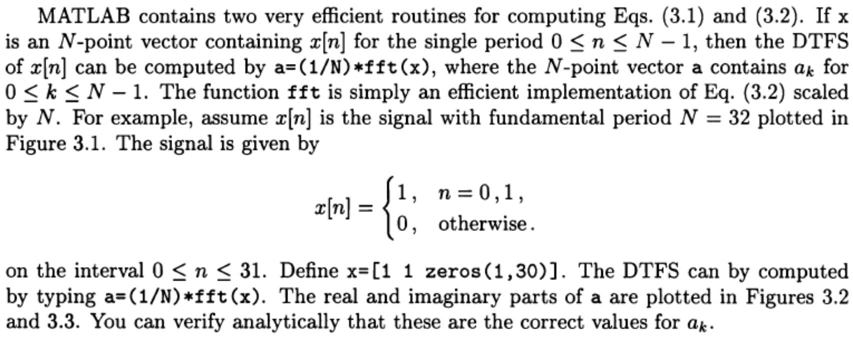


Figure 3.1. Periodic discrete-time signal $x[n]$ with fundamental period $N = 32$.

Tutorial: Computing the Discrete-Time Fourier Series with fft

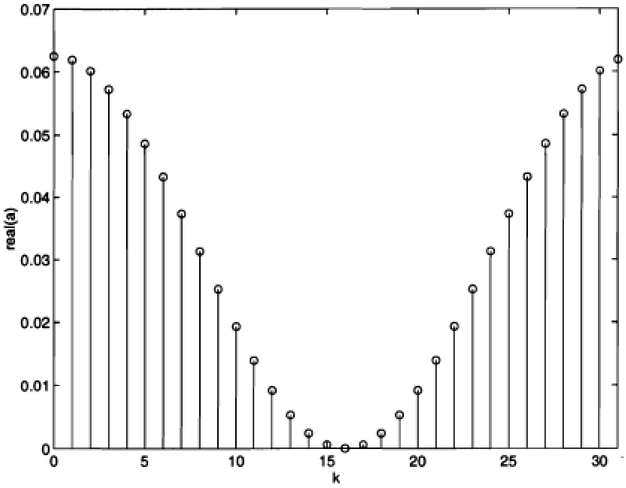


MATLAB contains two very efficient routines for computing Eqs. (3.1) and (3.2). If **x** is an N-point vector containing $x[n]$ for the single period $0 \leq n \leq N-1$, then the DTFS of $x[n]$ can be computed by `a=(1/N)*fft(x)`, where the N-point vector **a** contains a_k for $0 \leq k \leq N-1$. The function `fft` is simply an efficient implementation of Eq. (3.2) scaled by N. For example, assume $x[n]$ is the signal with fundamental period $N = 32$ plotted in Figure 3.1. The signal is given by

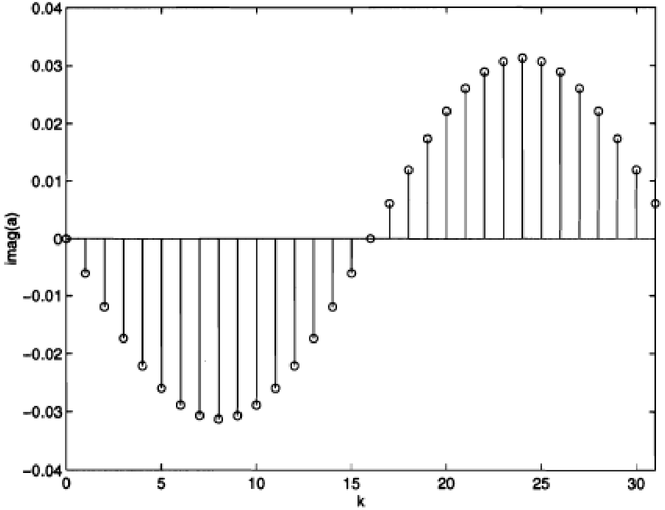
$$x[n] = \begin{cases} 1\,, & n = 0\,,1\,, \\ 0\,, & \text{otherwise}\,. \end{cases}$$

on the interval $0 \leq n \leq 31$. Define `x=[1 1 zeros(1,30)]`. The DTFS can by computed by typing `a=(1/N)*fft(x)`. The real and imaginary parts of **a** are plotted in Figures 3.2 and 3.3. You can verify analytically that these are the correct values for a_k.

Tutorial: Computing the Discrete-Time Fourier Series with fft



Tutorial: Computing the Discrete-Time Fourier Series with fft



Tutorial: Computing the Discrete-Time Fourier Series with fft

Given a vector a containing the DTFS coefficients $\mathrm{a_k}$ for $0 \leq k \leq N-1$, the function **ifft** can be used to construct a vector **x** containing $x[n]$ for $0 \leq \mathrm{n} \leq \mathrm{N}-1$ as follows: $\mathrm{x} = \mathrm{N} * \mathrm{ifft}(\mathrm{a})$.

Tutorial: Computing the Discrete-Time Fourier Series with fft

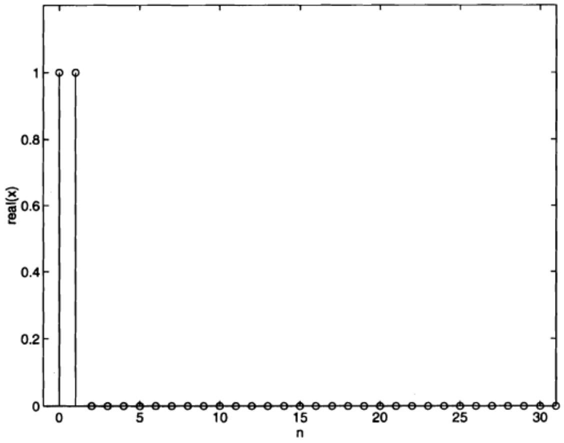


Figure 3.4. Real part of the synthesized discrete-time signal.

Tutorial: Computing the Discrete-Time Fourier Series with fft

Round off errors

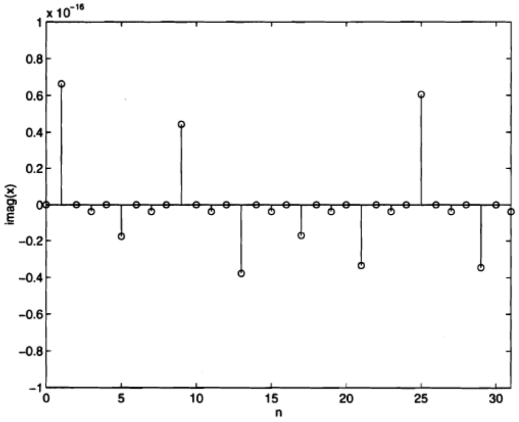
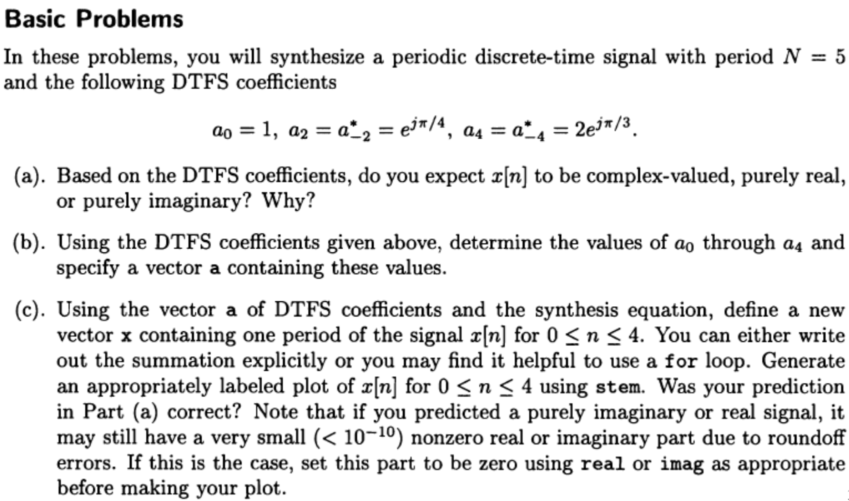


Figure 3.5. Imaginary part of the synthesized discrete-time signal. Note that the imaginary component is insignificant compared to the real component in Figure 3.4.

Synthesizing Signals with the Discrete-Time Fourier Series

Problem 3



Basic Problems

In these problems, you will synthesize a periodic discrete-time signal with period $N = 5$ and the following DTFS coefficients

$$a_0 = 1,\ a_2 = a^*_{-2} = e^{j\pi/4},\ a_4 = a^*_{-4} = 2e^{j\pi/3}.$$

(a). Based on the DTFS coefficients, do you expect $x[n]$ to be complex-valued, purely real, or purely imaginary? Why?

(b). Using the DTFS coefficients given above, determine the values of a_0 through a_4 and specify a vector **a** containing these values.

(c). Using the vector **a** of DTFS coefficients and the synthesis equation, define a new vector **x** containing one period of the signal $x[n]$ for $0 \leq n \leq 4$. You can either write out the summation explicitly or you may find it helpful to use a `for` loop. Generate an appropriately labeled plot of $x[n]$ for $0 \leq n \leq 4$ using `stem`. Was your prediction in Part (a) correct? Note that if you predicted a purely imaginary or real signal, it may still have a very small ($< 10^{-10}$) nonzero real or imaginary part due to roundoff errors. If this is the case, set this part to be zero using `real` or `imag` as appropriate before making your plot.

Synthesizing Signals with the Discrete-Time Fourier Series

Problem 3

Basic Problems

In these problems, you will synthesize a periodic discrete-time signal with period $N = 5$ and the following DTFS coefficients

$$a_0 = 1,\ a_2 = a^*_{-2} = e^{j\pi/4},\ a_4 = a^*_{-4} = 2e^{j\pi/3}.$$

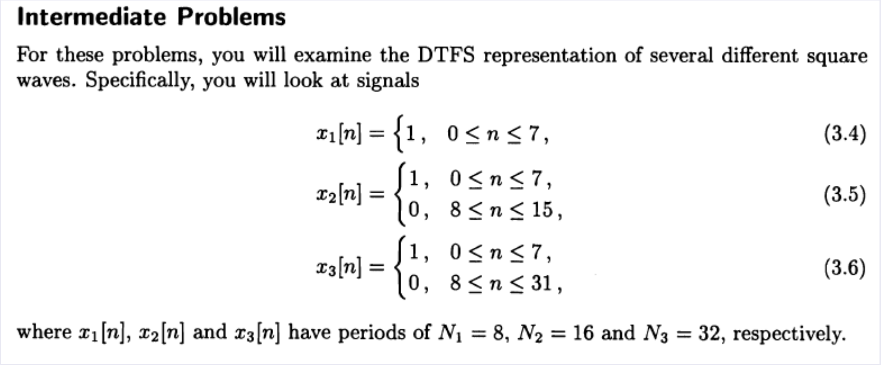
(a). Based on the DTFS coefficients, do you expect $x[n]$ to be complex-valued, purely real, or purely imaginary? Why?

(b). Using the DTFS coefficients given above, determine the values of a_0 through a_4 and specify a vector **a** containing these values.

(c). Using the vector **a** of DTFS coefficients and the synthesis equation, define a new vector **x** containing one period of the signal $x[n]$ for $0 \le n \le 4$. You can either write out the summation explicitly or you may find it helpful to use a `for` loop. Generate an appropriately labeled plot of $x[n]$ for $0 \le n \le 4$ using `stem`. Was your prediction in Part (a) correct? Note that if you predicted a purely imaginary or real signal, it may still have a very small ($< 10^{-10}$) nonzero real or imaginary part due to roundoff errors. If this is the case, set this part to be zero using `real` or `imag` as appropriate before making your plot.

Synthesizing Signals with the Discrete-Time Fourier Series

Problem 3



Intermediate Problems

For these problems, you will examine the DTFS representation of several different square waves. Specifically, you will look at signals

$$x_1[n] = \begin{cases} 1, & 0 \leq n \leq 7, \end{cases} \tag{3.4}$$

$$x_2[n] = \begin{cases} 1, & 0 \leq n \leq 7, \\ 0, & 8 \leq n \leq 15, \end{cases} \tag{3.5}$$

$$x_3[n] = \begin{cases} 1, & 0 \leq n \leq 7, \\ 0, & 8 \leq n \leq 31, \end{cases} \tag{3.6}$$

where $x_1[n]$, $x_2[n]$ and $x_3[n]$ have periods of $N_1 = 8$, $N_2 = 16$ and $N_3 = 32$, respectively.

Synthesizing Signals with the Discrete-Time Fourier Series

Problem 3

(d). Define three vectors **x1**, **x2**, and **x3** representing one period of each of the signals $x_1[n]$, $x_2[n]$ and $x_3[n]$. Using these vectors, make appropriately labeled plots of each of the signals over the range $0 \leq n \leq 63$. Note: You will have to repeat the vectors you have defined to cover this range of samples.

(e). Exercise 3.1 explains how to use `fft` to compute the DTFS of a periodic discrete-time signal from one period of the signal. Using the `fft` function, define the vectors **a1**, **a2**, and **a3** to be the DTFS coefficients of $x_1[n]$ through $x_3[n]$, respectively. Generate appropriately labeled plots of the magnitude of each of the DTFS coefficient sequences using `abs` and `stem`. From your time domain plots of Part (d) and Eq. (3.2), you should be able to predict the values of `a1(1)`, `a2(1)` and `a3(1)` — the DC components of the signals. Do your predicted values match those obtained with MATLAB?

(f). In this part, you will examine how $x_3[n]$ appears when it is synthesized a few coefficients at a time. Using the vector **a3** you found in the previous part, define vectors **x3_2**, **x3_8**, **x3_12** and **x3_all** corresponding to the following four signals

Synthesizing Signals with the Discrete-Time Fourier Series

Problem 3

$$x_{3\_2}[n] = \sum_{k=-2}^{2} a_k e^{jk(2\pi/32)n},$$

$$x_{3\_8}[n] = \sum_{k=-8}^{8} a_k e^{jk(2\pi/32)n},$$

$$x_{3\_12}[n] = \sum_{k=-12}^{12} a_k e^{jk(2\pi/32)n},$$

$$x_{3\_\text{all}}[n] = \sum_{k=-15}^{16} a_k e^{jk(2\pi/32)n},$$

Synthesizing Signals with the Discrete-Time Fourier Series

Problem 3

on the interval $0 \le n \le 31$. Note that since $x_3[n]$ is real, the DTFS coefficients for this signal will be conjugate symmetric, i.e., $a_k = a^*_{-k}$. Because a_k is conjugate symmetric and all of the sums except $x_{3\_\text{all}}[n]$ are symmetric about $k = 0$, the resulting time signals should be purely real. If you are unclear about why this is true, you may want to review the symmetry properties of the DTFS. Due to roundoff error in MATLAB, you may need to discard some very small imaginary parts of the signals you synthesize using `real`. The sums specified above are symmetric about $k = 0$ but the vector `a3` represents a_k for $k = 0, \ldots, 31$ as [`a3(1)`, . . . , `a3(32)`], so you will need to determine which elements of `a` correspond to the negative values of k when you implement the sums.

(g). Argue that $x_{3\_\text{all}}[n]$ must be a real signal.

(h). Generate a sequence of appropriately labeled plots using `stem` showing how the signals you created converge to $x_3[n]$ as more of the DTFS coefficients are included in the sum. Specifically, `x3_all` should be equal to the original vector `x3` within the bounds of MATLAB's roundoff error. Does the synthesis of this discrete-time square wave display the Gibb's phenomenon?

Outline

Assignments

Issued: May 23, 2024 Due: June 11, 2024

You should hand in the Matlab code (.m files), graphics, audio files and a brief description of your reasoning as well as comments if any. Please make sure that your Matlab code can be run on Matlab R2007b or higher version. You should pack all of your files into a .rar or .zip file, titled as xxxxxxx(your student ID) xxxx(your name) Lab pre, and then upload to blackboard before 11:59pm of the due day.