ELEC 324
Lab 2: Quantization and Digital Signals*

Winter 2022

1 Amplitude Quantization

Real-world analog signals can take on an infinite number of values, e.g., the temperature indicated by an alcohol-based thermometer could be any real number between say -40 and +100. Since it operates on a digital computer, Matlab cannot possibly represent such a number with full precision. Some precision must be lost, and this loss of precision can be viewed as an outcome of a *quantization* process. Low-power and low-cost devices typically do not even use floating point arithmetic at all, and instead represent numerical values as fixed-point numbers (integers). Performing fixed-point arithmetic causes additional losses in precision. This section will allow you to visualize quantization and its effects.

A discrete time (DT) signal is an idealized object because its sample amplitudes can take any values on the real line and hence cannot be represented on a digital computer. A *digital* signal is a discrete signal whose amplitude has been quantized. *Quantization* refers to the process of using a finite number of bits to represent the amplitude of an analog sample. That is, the values of the samples of a DT signal can be taken from a continuum of values, while that of a digital signal can only be taken from a finite number of values. The number of values a digital signal can take is determined by the number of bits used to represent the signal. For instance, an amplitude value represented with 2 bits can take one of 4 possible values, which could be -1.812, 0.2, 1, 3.3598001.

Quantization error refers to the numerical amplitude error between the digital representation and the analog representation. More generally, quantization error refers to the numerical amplitude error resulting from using a low resolution signal to approximate a high resolution signal.

1.1 Preparation

Quantization was covered very briefly in Lecture 1. In addition, Lecture 17 will provide additional material relevant to quantization.

Questions marked "Pre-lab" need to be answered *before* the lab. Bring your pre-lab answers to the lab with you.

You will need to bring a pair of headphones to the lab for listening. Ear buds will serve the purpose.

2 Discrete-time vs. Digital

Since Matlab operates on digital computers, it can only work with digital signals. However, by (i) using sufficiently high sampling resolution, and (ii) by carefully scaling an analog signal to the dynamic range of an A/D converter, digital signals can approximate DT signals fairly closely. As an introduction, run the Matlab code below:

```
% digital.m
a = 10;
x = 0:2*pi/1000:2*pi;
y = a*sin(x);
y_int = int8(y);

subplot(3,1,1);
plot(x,y);
grid;
axis([0 2*pi -a*1.2 a*1.2]);
title('Graph 1');

subplot(3,1,2);
plot(x,y_int);
grid;
axis([0 2*pi -a*1.2 a*1.2]);
title('Graph 2');

subplot(3,1,3);
plot(x,abs(double(y_int)-y));
grid;
axis([0 2*pi 0 1.2/2]);
title('Graph 3');
```

Questions:

2.0.1 Pre-lab: Numeric Resolution

Matlab conventionally stores real values in floating point format with high numerical resolution. Use Matlab's `eps` function, comment on the resolution of a *double*-type variable on the interval $[0, 1]$. Can we have a reasonable approximation of the amplitude of a real-valued signal on this interval? [Note: Investigate how the *relative* numerical

error, or the error as a percentage of the numerical value, changes with the value being represented in floating point format. Don't forget to use `help eps`.]

Answers:

2.0.2 Pre-lab: Quantization

How is quantization achieved in the above code? Which variable controls the amplitude resolution of the digital signal `y_int`?

Answers:

2.0.3 Digitizing

Give your interpretation for each of the three graphs produced by the above code.

Answers:

2.0.4 Error

Change a to 5 and re-run the code; comment on the differences observed for all three graphs. Repeat for $a = 100$.

Answers:

2.1 Quantizing an Audio File

For music on CDs, each sample has an amplitude resolution of 16 bits. In this section you will get to *hear* what happens when this resolution is reduced. Run the code below:

```
% truncate_wav.m
truncate = 6;
[f, fs] = audioread('flute.wav');
info = audioinfo('flute.wav');
nbits = info.BitsPerSample;
f_int = int16(f*2^(nbits-1-truncate));
f_back_to_float = (double(f_int))/2^(nbits-1-truncate);
f_diff = f - f_back_to_float;
audiowrite('truncate_6.wav', f_back_to_float, fs);
audiowrite('diff_6.wav', f_diff, fs);
```

Questions:

2.1.1 Pre-lab: Truncate

What does the `truncate` variable represent? Logically, what are its upper and lower bounds? Justify. Hint: `wavread` stores the audio samples as *signed* floating point num-

bers between -1 and $+1$.

Answers:

2.1.2 Subjective Comparison

Change *truncate* to 9, 10, 11 and 12 and change the output file names so that you won't overwrite your previous results, and re-run the code. Listen to the output files for both values of `truncate` and comment on the differences. One way to do this is to run the Matlab function `sound`. Can you relate this to what you saw in Section 2?

Answers:

2.1.3 Quantization Error

For a `truncate` value of 9, what is the maximum possible value of `f_diff`? Calculate this analytically, then use the `max` function in Matlab to validate your result. Comment on the possible cause(s) of any observed discrepancy between your calculation and the value found using `max`.

Answers:

2.1.4 Signal-to-Distortion Ratio

We define the signal-to-distortion ratio (SDR) as the ratio of average signal power to average error power. SDR in dB can be calculated as

$$\gamma = 10 \log_{10} \left(\frac{\frac{1}{N} \sum_{n=0}^{N-1} s^2(n)}{\frac{1}{N} \sum_{n=0}^{N-1} e^2(n)} \right)$$

where $\{s(n), n = 0, \ldots, N-1\}$ are signal samples, and $\{e(n) = s(n) - \hat{s}(n), n = 0, \ldots, N-1\}$ are the quantization error samples, with $\hat{s}(n)$ denoting the quantized $s(n)$.

That is, we define the error signal $e(n)$ to be the difference between the original and truncated signals. Write a Matlab program to calculate γ using the above formula for:

1. the truncated signal when `truncate` $= 6$;
2. the truncated signal when `truncate` $= 9$;
3. the truncated signal when `truncate` $= 12$. .

Can you reconcile these values with the listening quality you perceive?

Answers:

2.2 Quantizing an Image File

The manipulations we made on an audio file in Section 2.1 can also be applied to an image file. Instead of listening to the results you will now view them.

The `imread` command is used to read an image into Matlab:

```
x = imread('image.jpg');
```

The **x** matrix contains the image data, and its dimensions depend on the type of image. For a gray-scale image, it will be a two-dimensional matrix with (horizontal) x and (vertical) y dimensions equal to that of the image (i.e., a 100 x 100 pixel image will be stored in a 100 x 100 matrix). Each element of the matrix is of type `uint8` (an unsigned 8-bit integer). Each pixel can therefore represent one of 256 shades of grey, with 0 value representing the darkest shade and 255 the brightest white.

Mathematically, a digital image can be modelled as a two-dimensional discrete(-space) function. A gray scale image is a scalar-valued function.

A colour image is a matrix of vectors, where each vector has three components. Multi-spectral images such as those taken by satellites may have more than three components. In a colour image, each pixel located in the two-dimensional spatial domain has a vector amplitude value. Each amplitude vector stores three values, one for each of the three primary spectral colour components: red, green, and blue. `imread` stores a colour image in a three-dimensional matrix. A 100 x 100 pixel colour image is stored in a 100 x 100 x 3 matrix, i.e., the third dimension is used to store the red, green and blue values. In other words, we have three 100 x 100 matrices, one for each of the colour components. Each colour component value is again of type `uint8`, with 0 representing null intensity and 255 maximum intensity of each respective colour.

Now run the code below:

```
% truncate_grey_img.m
clf;
truncate = 5;
x = imread('aqua_grey.jpg');
subplot(2,2,1);
imshow(x);
subplot(2,2,2);
x_t = (x/2^truncate) * 2^truncate;
imshow(x_t);
subplot(2,2,3);
imshow(uint8(abs(double(x_t) - double(x))));
```

Questions:

2.2.1 Pre-lab: Truncate

What is the practical range for the `truncate` variable in the above code?

Answers:

2.2.2 Program Output

What does each of the three images generated by the above code represent?

Answers:

2.2.3 Subjective Comparison

Using the same technique as prescribed in Lab 1, "Varying the Time Resolution", modify the program so that it loops over the range of valid `truncate` values, pausing for half a second between each value. Comment on what you observe.
Answers:

2.2.4 Working with Colour

Modify the above code to operate on a colour file, "aqua.jpg". *Hint:* there is not much code change required.

Answer:

2.3 Colour Separation

If `x` is an array containing a colour image, the following commands can be used to extract its red, blue and green components:

```
x = imread('aqua.jpg');
red = x(:,:,1);
green = x(:,:,2);
blue = x(:,:,3);
```

What do your observe by executing `image(red)`? You might expect this to result in a purely red image, but it does not. Try `image(green)` and `image(blue)` also. The explanation for this behavior is that each element of a two-dimensional matrix (such as `red`, `green` or `blue` above) is only a scalar. We need three scalars to specify the intensity of the three colour components. One way to get around this is to map each scalar to a triplet of scalars. This mapping can be instrumented using a *colormap*, which is also called a colour quantization table or colour quantization codebook. The notion of colour quantization relates to the fact that indexing a colourmap with m rows allows only m out of 2^{24} possible colour values to represent or display the image. (You could imagine how a video card on a PC could utilize a colourmap.)

The colour map is *not* used when displaying a full colour image contained in a three-dimensional matrix, such as the one read from a file in Section 2.2. The three colour

components of each pixel are fully specified and so no colour map is needed to determine how it should be displayed.

The colourmap is an $m \times 3$ matrix. m can be any nonnegative integer. By default, the colourmap is a 64×3 matrix. Each scalar element in a two-dimensional matrix corresponds to a pixel to be displayed. The value of the scalar element is used to index the rows of the colourmap. For instance, suppose $m = 256$ and each scalar element in the matrix is an 8-bit number. The value of the scalar element is used as a row index value, i.e., a value i maps to the i-th row. $i = 0$ and $i = 1$ both map to the first row. The three values in the indexed row determines the colour component values of the pixel to be displayed. In this manner, the colourmap maps a one-dimensional value to a three-dimensional value, $\{red, green, blue\}$, so that each pixel represented with one scalar value can be displayed in colour.

For example, if the first row of the colour map is $[0, 1, 0]$, any pixel with the scalar value of "1" in the two-dimensional matrix (`red` or `green` or `blue`) will result in a green dot.

Questions:

2.3.1 Changing the Colour Map

Write code to properly display the two-dimensional matrix `red` as a true-colour component image. To do so you will need to modify the colour map. Since each element of the `red` image is a value from 0 to 255, the colour map should have 256 entries. *Hint:* In Matlab, dot-multiply a *gray* colour map of the proper size and with a matrix containing *ones* and *zeros*.

Answer:

How would you manipulate the colour map to create a yellow-coloured image?

Answer:

2.3.2 Manipulating the Colour Map

In much the same way that an image can be decomposed into red, green and blue components, the image can be recomposed from these components.

Using this and your knowledge from the previous sections, write a program which can "truncate" a variable number of bits on each of the three colour components and recompose the image from the components. Experiment to find the maximum total number of bits that can be eliminated without any noticeable degradation in image quality. Comment on how the size of an image file could be reduced using this approach.
Answer: