GE U111 HTT&TL, Lab 4

Underwater Acoustic Distance Measurement,

Vector Analysis & Plotting in MATLAB

Contents

1 Overview: Programming & Experiment Goals

This session is used to practice some new (MATLAB) programming concepts and continue reinforcing previously used tools, including

- analysis of vector arrays
- additional plots of scalar functions
- advanced 2D looping
- logical constructs
- user defined functions

Basic (Engineering / Science) concepts, including

- Sampling & sample rates of sinusoidal signals
- Thresholds & noisy signal detection

The driving application is underwater acoustic distance measurement, using an ultrasound transducer attached to the programmable precision planar positioner from Lab 3. Specific tasks include

- A purely computerized estimation of the speed of sound in water
- Acoustic distance measurement

The Components & Equipment used in this experiment include:

- The planar positioner from Lab 3.
- A glass aquarium.
- An underwater[1] 1 MHz ultrasound transducer, serving as both an emitter and a receiver, attached to the positioner. The emitted sound wave travels to a remote object or the aquarium wall, and the return echo is then received by the transducer.
- A digital oscilloscope.

Experiments Outline & Goal. In this lab we revisit and expand the tasks performed in the first session: measuring the speed of sound and using an acoustic transducer to create a remote distance measurement device. This time we shall use the underwater transducer and positioner system, and measurement will be of the speed of sound in water. The innovation will be in that "human intervention", reading an oscilloscope, or manually tabulating and plotting measured signals, will be replaced by automatic data acquisition and processing, by the computer, using MATLAB. The digital oscilloscope is used here primarily as an intermediary between the transducer and the computer (instead of a more efficient but much more expensive sampling device) and secondarily for deductive purposes, to compare programmed results with visual observations.

2 Homework Assignments

- Pre-Lab Homework
  - Read this handout carefully
  - Refresh previous reading assignment.
  - Read Section 3.6 (built-in array functions), Chapter 5 (2D plotting), and Chapter 6 (function m-files).
  - The Pre-Lab tasks in this handout.
- Post-Lab report due one week after completion of lab.

[1] 1 MHz = 1 mega Herz = 10^6 cycles / per second.

3 Sampling of Sinusoidal Signals

While most observed processes evolve continuously, computers are inherently finite and discrete: Quantities are stored in the computer using a finite number of digits and evolve at discrete clock ticks. Sampling is therefore essential to signal storing and processing by a computer. The simplest and most common way is to sample time signals at a fixed frequency (of samples-per-second). There is an obvious tradeoff involving the sampling frequency: High sampling frequency provides better representation but requires more data storage and slower processing. In what follows we make some simple observations regarding this tradeoff in relation to the signal `y=sin(x)`:

Pre-Lab Task 1. In a MATLAB script file named Sample.m Define the following time grids and samples of `y=sin(x)`:

```
x0=0:.01:8*pi;    y0=sin(x0);    % high rate:  about 628 samples per period
x1=0:6.1:8*pi;    y1=sin(x1);    % approximately 1 sample per period
x2=0:3.1:8*pi;    y2=sin(x2);    % approximately 2 samples per period
x5=0:1.25:8*pi;   y5=sin(x5);    % approximately 5 samples per period
x10=0:0.6:8*pi;   y10=sin(x10);  % approximately 10 samples per period
x20=0:0.31:8*pi;  y20=sin(x20);  % approximately 20 samples per period
```

Following these definitions, make the necessary `subplot` commands to include six plots of the pairs `(x0,y0)`, `(x1,y1)`, `(x2,y2)`, `(x5,y5)`, `(x10,y10)` and `(x20,y20)` in one figure window, organized in 2 rows and 3 columns (see Section 5.10 in the book). The plots should follow these instructions-

1. The plot of `x0,y0)` should be in solid line.
2. The remainder five plots should include *both* a solid line *and* dots at data points (see tables in Section 5.1 in the book)
3. Each plot should have a title: `plot of (x0,y0)`, `plot of (x1,y1)`, etc., and axis labels: `x` and `y`

Run your program and inspect the plots. Assuming that our goal is to estimate the signal amplitude to within 5% accuracy, suggest an appropriate sampling rate, based on your inspection. As usual, include a copy of the program both in your pre-lab report an on a memory stick that you will bring to the lab.

4 Threshold for Signal Detection

Key to the estimation of the speed of sound in air, in Lab 1, was the ability to detect the arrival of the sound wave at the receiver and thereby, the travel time from the emitter to the receiver. In Lab 1 we did that by visual inspection of the signal in the oscilloscope window. Here we plan to replace visual inspection by computer analysis of a sampled signal, and need a programmed detection mechanism.

To illustrate the main idea we use a schematic depiction of the transducer signal, in the left plot, in Figure 1. The signal begins with an emitted burst, represented here by a single cycle of a 5Hz sine wave[2] of amplitude = 3, along the interval $0 \leq time \leq 0.2$. This is followed by an interim portion of background noise along $0.2 \leq time \leq 3$, during which the signal travels to the aquarium wall, or a remote object and then echoes back, towards the transducer. A higher amplitude of a 5 Hz sine wave indicates the arrival of the returned echo. Detection of the returned echo will be therefore be in terms of this rise of the signal amplitude.

Ideally, the mean of a sine wave is zero and its amplitude is therefore measured by its peak absolute value. However, sensor signals are often *biased*. This means that, on average, the sensor adds (or subtracts) a small quantity to the actual signal. Thus, the absolute value of the transducer signal (middle plot) might not be the exact amplitude. The bias is determined as the *mean* value of the sensor signal, and is removed when the mean value is subtracted from from the sensor signal. The right plot in Figure 1 provides a correct depiction of the amplitudes of the centered signal.

The detection of the arrival of the returned echo will be by finding the point where the signal amplitude exceeds a set *threshold*. If the threshold is set too high, it will result with delayed detection (or no detection). This is represented by the cyan threshold in the right plot, in Figure 1. Conversely, if the threshold is too tight, detection will be susceptible to false positives, due to occasional high spikes in the background noise, as represented by the red threshold in Figure 1. A successful selection, such as the blue threshold in the figure, is a balance of these two considerations.

[2] 20 cycles of a 1 MHz sound wave will be used in the actual experiment.

Finally, we want to prevent false detection of the high amplitude emitted signal as an arrived echo. A time threshold must therefore be set, to guarantee that an amplitude rise is detected only after the emitted signal is sent. This is represented by the vertical green line.

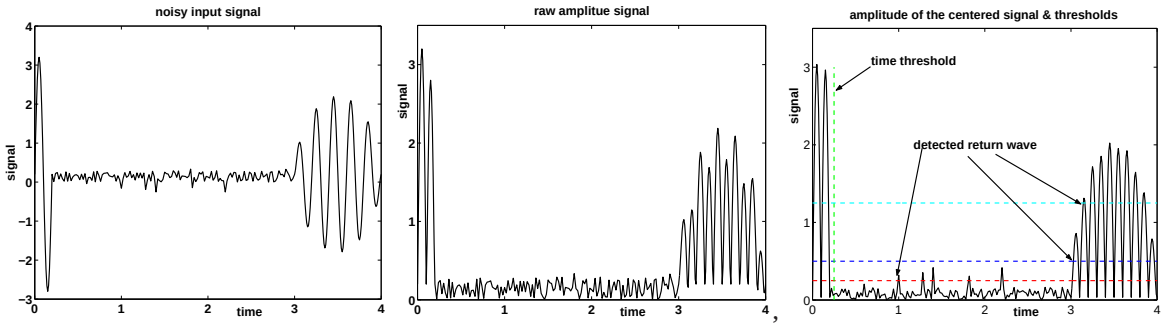

Figure 1: A Schematic depiction of the transducer signal. Left: The input signal, include the emitted burst, represented here by a single cycle of a 5Hz sine wave of amplitude = 3 along the interval $0 \leq time \leq 0.2$, an interim portion of background noise along $0.2 \leq time \leq 3$, and the higher amplitude return echo of the 10 Hz sine wave. Middle: The sensor signal might be *biased*, meaning that it will not be centered at zero. Thus, its absolute value will not be the correct amplitude. Right: subtracting the *mean* before taking the absolute value centers the signal, and now its absolute value is the correct amplitude. Arrival of the returned echo is detected by the amplitude exceeding an amplitude threshold. The noise includes infrequent high spikes. A too low threshold (red) will therefore result with premature prediction of the return wave arrival. A useful threshold (blue) will be high enough to clear such spikes, but not too high to result with delayed prediction (cyan). A minimum time threshold (green) is necessary to clear the emitted signal.

Pre-Lab Task 2. Here you will write a MATLAB function m-file named `travel_time.m` that detects the travel time of the signal, from the beginning of the emitted burst, till the detection of the arrival of the returned echo. General guidelines for the definition of user defined functions are part of the reading assignment for this lab. The specific setting for this functions are as follows:
Input variables:

- Two, same-length vectors of signal data, including
  - `signal` is a list of sampled transducer signal.
  - `time` is the corresponding list of sampling times.

  Thus `signal(k)` is the value of `signal` at the time `time(k)`. The length of these two vectors is not known a priori, but the function can determine it using the MATLAB function `K=length(signal)`.
- Two thresholds
  - An amplitude threshold, named `amp_th`: a return echo is detected when the signal amplitude exceeds `amp_th`.
  - A time threshold, named `time_th`: a return echo is sought only after the time `time_th`

Output variable: The estimated travel time of the sound wave, given the input data and selected thresholds. The output variable will be named `ttime`.

Accordingly, the first line of your M-file `travel_time.m` will be:

```
function  ttime=travel_time(signal,time,amp_th,time_th)
```

The following are hints and guidelines for the actual programming:

- To remove sensor bias you have to subtract the mean from `signal`:

  ```
  c_signal=signal-mean(signal);
  ```

- The logic function `condition=(time>time_th & abs(c_signal)>amp_th);` defines a vector named `condition`, whose entries are zeros and ones, and whose size is the same as the size of `time` and `signal`:

$$\texttt{condition(k)} = \begin{cases} 1 & \text{if} \quad \begin{array}{c} \texttt{time(k)} > \texttt{time\_th} \\ \texttt{abs(c\_signal(k))} > \texttt{amp\_th} \end{array} \\ 0 & \text{else} \end{cases}$$

  In Figure 1, this relates to the requirement that the amplitude of `signal` is above the blue line, and points that are to the right of the green line.

- The command `ind=find(condition)` defines the vector of indices `ind` = {`k: condition(k)=1`}. This is the set of indices where both threshold conditions are satisfied. Thus, the sub-vector `time(ind)` is the vector of times where both conditions are satisfied.

- The index `k0=min(ind)` is such that `time(k0)` is the first time when both threshold conditions are satisfied. That is, **`ttime=time(k0)` is the estimated travel time of the returned echo.**

- Finally, despite all good intentions, it might happen in an experiment (and did more than once in our HTT&TL lab) that the selected value of the threshold `amp_th` is too large, and that the condition `abs(c_signal)>amp_th` is never satisfied. If we follow the logic of the definitions above, then

  – All entries of `condition` will be zeros.

  – The vector `ind` will be empty

  – The index `k0` will take the `empty variable` value []. (Try `min([])` in a MATLAB command window.)

  – The returned value will be empty and a program that uses your function might get stuck, later on.

  To avoid that you need to guarantee that your function will always return a numerical value for `ttime`. At the same time you need to be able to distinguish a returned value that results from no detection. One way to do so is to select a returned value that is impossible for a legitimate value of `ttime`, such as zero[3]. The function therefore needs to do the following

  – The logical condition `isempty(ind)` returns the value of 1 when `ind` is empty.

  – If that is the case, the program should set `ttime=0;`

  – If that is not the case, the program can continue, as indicated earlier.

You have to write the function, and try it on some test input that you will create yourself. Include the function and a print of the command window from a successful run, in your pre-lab report, as well as a copy of the function on the diskette you bring to class.

[3] ttime=0 is physically impossible since it always takes some positive duration of time for the sound wave to travel to the remote object and back to the transducer.

5 Experiment: Measuring the Speed of Sound in Water

5.1 Operating & Retrieving Transducer Data

As in the case of the positioner and the stepper motor, an initiation command is needed, to activate the system: The following command must therefore be issued *once* at the beginning of the lab session, in the MATLAB command window-

```
[dio,scope]=setup_tank;
```

This command

(a) Initiates the power supply setting.

(b) Initiates the signal generator: repeated 20 microseconds ($2 \cdot 10^{-5}$ sec) bursts of a 1MHz (10^6 cycles per second) sound wave are transmitted from the ultrasound transducer. (Thus, each burst lasts 20 acoustic cycles.)

(c) Initiates the digital oscilloscope setting.

(d) Creates the instrument control / data acquisition objects `dio` and `scope` that are needed in order to control the positioner and download data from the oscilloscope.

The function `pulseecho` is used to download from the scope and to down sample a reading of the transducer signal. The syntax of calling this function is

```
out=pulseecho(n,scope)
```

This command puts into effect the following steps

(a) Concurrently with the beginning each transmission of a 20 microseconds / 1MHz burst from the ultrasound transducer, sampling of the transducer signal begins, lasting a set time duration. The effective sampling rate is `n` MHz, meaning a rate of $n \cdot 10^6$ samples per second. Allowed values of `n` are 1, 2, 5, 10, 20, 50, 100 and 200. (Thus, the actual command may be, e.g. `out=pulseecho(5,scope)`.)

Notice: Since the emitted signal frequency is 1 MHz, the value of `n` determines how many samples-per-cycle we use. This connects to the discussion in Section 3, above.

(b) The returned value `out` is a two-columns array (the name `out` is arbitrary, and you may use any name you choose): The 1^{st} column consists of the list of sampling times, and the 2^{nd} column consists of the corresponding samples of the transducer signal. You may name the two column vectors time and signal. They are obtained using the array commands -

```
>> time=out(:,1);
>> signal=out(:,2);
```

Lab Experiment 1 Obtaining a transducer signal. In this lab the transducer is connected to the positioner, facing the front (long) wall of the aquarium, so that horizontal motion controls the distance between the face of the transducer and the facing wall.

1. Bring the positioner to the lowest position.
2. Move the positioner horizontally, so that the distance between the transducer and the aquarium wall it faces will be approximately 11 cm.

3. Issue the command `out=pulseecho(n,scope)`
4. Obtain the vectors `time1` and `signal1`, as above
5. Plot `signal1` as a function of `time1`. Provide axis names and title, as well as the names of your team members, and print a copy for each team member.
6. Now move the positioner 10 cm horizontally towards the front wall of the aquarium, repeat the experiment, obtain and plot vectors `time2` and `signal2`.

The purpose of this experiment is twofold: It verifies your ability to retrieve transducer data, and it provides a test samples, from which you will derive threshold values.

5.2 Obtaining Amplitude and Time Thresholds

Lab Experiment 2 Obtaining & testing detection thresholds.

1. Use the vectors `time1`, `signal1`, `time2` and `signal2`, from Lab Experiment 1 to obtain valid values for the amplitude threshold `amp_th` and the time threshold `time_th`, as follows -
   (a) Define the signals' amplitudes as the absolute values of the centered signals (i.e., after removal of the mean): `a_signal1=abs(signal1-mean(signal1));` and `a_signal2=abs(signal2-mean(signal2));`
   (b) In two new figure windows[4], plot `a_signal1` as a function of `time1` and `a_signal2` as a function of `time2`.
   (c) Since you will need to add lines to these plots, bring up each of the two figures and issue (for each) the command `hold on`.
   (d) Inspect your plots closely, using the plot zoom tool (the "magnifying glass"), and determine a suggested value of `amp_th`.
   (e) Plot horizontal dashed red lines across your figures, at the selected level of `amp_th`. This can be done using the command: `plot(time1,amp_th*ones(size(time1)),'r--')` and `plot(time2,amp_th*ones(size(time2)),'r--')`
   (f) Inspect your plots again and check
      i. Does your threshold leave enough clearance headroom above the noise signal to avoid false detection in case of noise spikes? Use the entire stretches of noise in your signal. It is important to be safe in this respect!
      ii. Is your threshold far too conservative (high) to cause delayed detection or no-detection?

   If you need to change your selection, use the figure pointer to highlight the red line, delete it, and plot again, with a corrected selection of `amp_th`.
   (g) Now select a candidate for `time_th`: It should be large enough to clear both the emitted burst and some initial, high amplitude internal ringing that will be evident in the plots. At the same time, `time_th` should not be too large to reach the returned echo in the plot of `a_signal2`.
   (h) Plot a dashed vertical green lines at the points `time=time_th`, in both figures. This can be done as follows:
      i. Define `Line=0:.1:max(a_signal1',a_signal2');`. (The primes (') following `signal1` and `signal2` convert the column vectors into row vectors so they can be conjoined into one row vector.) By extending to line to the `max` of `signal1` and `signal2`, the line height will be the same as the peak amplitude of the transducer signal.
      ii. `plot(time_th*ones(size(Line)),Line,'g--')`, in each of the two figures.
   (i) Your selection of `time_th` is critical too. If you are not satisfied with your selection, delete the two green lines, amend your selection and plot again.
   (j) Once you are satisfied, add axes names and figure titles, and print a copy for each team member.

[4] To open a new figure window, issue the command `figure`.

2. Now bring the positioner to an intermediate position and perform the following

   (a) Obtain a new sample of `time`, `signal` and `a_signal=abs(signal-mean(signal))` using `pulseecho`, as above.

   (b) Plot `a_signal` as a function of `time`

   (c) Make sure the current directory is A:\ and call your function from Pre-Lab Task 2:

   ```
   ttime=travel_time(signal,time,amp_th,time_th)
   ```

   (d) Use the oscilloscope cursers to measure the travel time, following the instructions in Lab 1

   (e) Compare the visual measurement with the oscilloscope and the value of `ttime`. A substantial gap means either that your thresholds where inappropriate or that your function `travel_time` contains a mistake.

   (f) Make the appropriate corrections till you get the right answer. Ask for help if needed.

3. Summarize your findings in your Lab Report.

5.3 Estimating the Speed of Sound in Water

The estimation of the speed of sound in water will follow the same logic as in Lab 1, but will be executed by a single program. As in Lab 1, the program will conduct 10 experiments where the travel time of the emitted sound wave from the transducer to the front wall and back. Clearly there holds

$$2 \times \texttt{distance to the wall} \;=\; \texttt{speed of sound} \times \texttt{travel time} \tag{1}$$

While we will not measure the exact initial distance from the transducer to the wall, we are in full control of *changes* made to this distance through positioner motion: We shall command 1cm steps towards the wall. Thus `distance to the wall` in the $k+1^{st}$ step is shorter by precisely 2cm, as compared with the $k^{th} step$, or -2k cm, as compared with the first step. We can thus re-write (1) as

$$-.02 \times (\texttt{k}-1) \;=\; \texttt{speed of sound} \times \texttt{travel time} \;+\; \texttt{shift} \tag{2}$$

where `shift = -2` $\times$ `distance to the wall in step #1`. The advantage of the formulation (2), as compared with (1), is that the left hand side is known, and the unknown value of `shift` in the right hand side has no bearing on the slope, which is the sought speed of sound. In particular the estimation of the speed of sound from a straight line approximation of the plot of the k^{th} travel time as a function of $-2(k-1)$ does not require knowledge of `shift`! We shall obtain an estimate of the slope using `polyfit`, as in Lab 1.

Pre-Lab Task 3. Here you will write a MATLAB function named `SOS_water.m`. The input to the function will be the two tolerance values `amp_th` and `time_th`. The output will be an estimate of the speed of sound in water. As a preliminary step, the positioner should be brought to an initial position as close as possible to the rear wall of the aquarium.

The function `SOS_water.m` should execute a 10 steps `for` loop , where at each cycle

1. The `pulseecho` command will be used to obtain a sampled transducer signal.
2. The function `travel_time` will be used to estimate the travel time of the emitted sound wave from the transducer to the front wall and back. In the k^{th} step. this value will be saved as the k^{th} entry of a vector named `ttime_v`.
3. If $k < 10$ the positioner will move 1 cm towards the front wall.

Following termination of the loop, the `polyfit` function will be called, to fit the data with a first order polynomial $p = [p(1),\, p(2)]$, approximating -

$$-2(k-1) \approx p(1)\texttt{ttime\_v}(k) \;+\; p(2), \qquad k = 1, 2, \ldots, 10$$

This is done with the command `p=polyfit(travel_time,(0:-2:-18),1)`. The returned value of the function will be the slope, $p(1)$.

Lab Experimet 3. Debug and execute the function `SOS_water.m`. Compare your estimate to known estimates, using www resources. Summarize your findings and include a print of the relevant part of the command window in your report.

5.4 Acoustic Distance Measurement

As in Lab 1, here again we use the ultrasound transducer for distance measurement. having estimated the speed of sound, we can use (1) to translate travel time measurement (using the function `travel_time`) of the sound wave into distance estimates.

Lab Experiment 4 Here you have to write, debug and execute a MATLAB function named `distmeasure.m`, as follows-

1. The function has no input.
2. The thresholds `time_th` and `amp_th` will be globalized (using the command

   ```
   global time_th amp_th
   ```

   in both the command window and the function second line)
3. The function will use the `pulseecho` command to obtain the sampled sound wave and time signals.
4. It will then call the `travel_time` function, to compute the returned echo travel time.
5. Finally, it will use (1) to compute the distance to the remote object[5]. This will be the value the function returns.

Move the transducer to three different positions, and compare acoustic distance measurements with manual measurements, with a ruler. Include the program and a printout of the relevant section of the command window in your Lab report.

[5]Note that in its current form, the left hand side of (1) is *twice* the distance to the object.