1

Signals and Systems Laboratory

1.1 Purpose

In this course, almost all our laboratories will be based on creating Matlab functions to perform various tasks. The purpose of this assignment is to help you become familiar with some of these Matlab operations.

1.2 Background

To prepare for this exercise, read Appendix C in the book. It gives a brief tutorial on Matlab.

Matlab has a number of core deficiencies, but the most severe for our purposes is that addressing of arrays in Matlab follows the ones-based indexing convention of an ancient programming language, FORTRAN, meaning that the first element of an array `s` is `s(1)`, not `s(0)` as it would be in C, C++, Java or other modern languages with zero-based indexing. In your assignment, we'll make a simple workaround that solves this problem.

The main feature we will use is the **structure**, which is a single variable with a number of **fields,** each of which is an independent piece of data. In our case, we want our variable to encapsulates the two pieces of information that completely describe a sequence: the sequence **data** and the sequence **offset**. For example, given the sequence $x[n]=2\delta[n+1]+\delta[n]-\delta[n-2]$, the data is the array [2 1 0 -1], and the offset is the time index of the leftmost data point, -1. In Matlab, as in other languages, you use a ‘dot' construction to create fields of a structure:

```
x.data = [2  1  0  -1];
x.offset = -1;
```

1.3 Your assignment

In this assignment, you will write a series of Matlab functions to perform the basic DSP operations on structures we have been learning: flipping, shifting and adding sequences. Here are the functions that you will write:

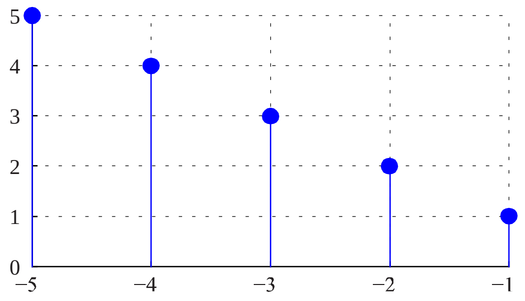
```
function y = flipit(x)
% FLIPIT Flip a Matlab sequence structure x so y = x[-n]

function y = shiftit(x, n0)
% SHIFTIT Shift a Matlab sequence structure x by integer amount n0 so that
%      y[n] = x[n - n0]

function z = addit(x, y)
% ADDIT  Add x and y. Either x and y will both be sequence structures or one of
them may be a number.

function z = subit(x, y)
% SUBIT Subtract x and y. Either x and y will both be sequence structures or one
of them may be a number.

function z = multit(x, y)
```

```
% MULTIT Multiply x and y (i.e. .*) Either x and y will both be sequence struc-
tures or one of them may be a number.

function z = trimit(x)
% TRIMIT Remove leading and trailing zeros from sequence x and adjust offset ap-
propriately.

function stemit(x)
% STEMIT Display a Matlab sequence x using a stem plot.
```

When you are have properly written your functions you will be able to create a sequence like this:

```
» x.data = [1 2 3 4 5];
» x.offset = -1;
```

Then, `stemit(flipit(shiftit(x, 2)))` will produce something like the following:



Here is a sample (though *bogus*) implementation of the `add` function. It takes two sequence structures `x` and `y` as input and produces sequence structure `z` as output.

```
function z = addit(x, y)
   z.data = x.data + y.data;
   z.offset = x.offset + y.offset;
end
```

Here we first access the `data` and `offset` elements of the sequence structures `x` and `y`. Then we create and return the new sequence structure `z`.

- You must properly handle the arguments to the `add`, `sub` and `mult` functions. Each of these functions has two arguments. Both arguments could be sequences, or one of the arguments could be a numerical constant. Hence, all of the following should work:

```
» x.data = [1 2 3 4];
» x.offset = -1;
» y.data = [1 1 1];
» y.offset = -2;
» addit(x, y)
ans =
      data: [1 2 3 3 4]
    offset: -2
```

```
» addit(x, 1)
ans =
      data: [2 3 4 5]
    offset: -1

» addit(1, x)
ans =
      data: [2 3 4 5]
    offset: -1
```

- There should be no leading or trailing zeros in the output of the `addit`, `subit` and `multit` functions. For this purpose, you can embed into the `addit`, `subit` and `multit` the function `trimit` that you have written. For example:

```
» x.data = [1 2 3 4 5];
» x.offset = 0;
» y.data = [1 2 3];
» y.offset = 0;
» subit(x, y)
ans =
      data: [4 5]
    offset: 3
```

- Your functions should produce no extraneous output, so if there is a semicolon at the end of the line, the function does its work silently:

```
» y = addit(x, 1); % should produce no output
```

1.4 Hints and points to consider

- You shouldn't have to use any Matlab's iterating operators such as `for` or `while` to perform this assignment. Matlab's array operators are sufficient. However, if you need `for` or `while`, use them. This is an assignment in signal processing with Matlab, not an assignment in clever Matlab programming.
- You can use the Matlab `find` command in your `trimit` function to determine where the zeros are in your data, and strip them off.
- Consider using reversed indices to index into an array, if necessary. For example, `x.data(end:-1:1)`. Note that `end` is a Matlab keyword that, when used as an array argument, denotes the last value in the array. Also, `length(x.data)` gives the length of the data in the structure.
- You can concatenate two arrays of data, `x` and `y`, by using `[x y]`as long as the number of rows is the same.
- To pad the beginning or end of an array with zeros, concatenate that array with a sequence of zeros made with the `zeros` command, for example `[zeros(1, 2) x.data zeros(1, 3)]` pads the beginning of the array with two zeros and the end with three zeros (assuming `x.data` has one row).

```
» [zeros(1, 2) x.data zeros(1, 3)]
ans = 0 0 1 2 3 4 5 0 0 0
```

Of particular note is the fun Matlab quirk that `zeros(1, n)` will add no zeros if `n` is less than or equal to zero. If you are clever, you can use this feature to your advantage so that you don't have to use if statements to distinguish between various cases. However, don't sweat it. Again, this isn't a course in Matlab cleverness. As long as you get code that works, it's good enough.

- To do a term-by-term multiply of two arrays of equal length, use the `.*` operator (notice the `'.'`)
- You can figure out whether a variable is a structure or just a plain variable using the `isstruct` command:

```
» x.data = [1 2 3];
» x.offset = 1;
» isstruct(x)
      ans =  1

» y = 1;
» isstruct(y)
      ans =  0
```

You can use this to determine the arguments of your functions are sequences or numbers. That's useful in implementing things like `add(1, x)`.

1.5 Submitting your assignment

- Your functions should have the same form as the ones described in this write-up. Make sure your code is commented.
- You should find the following files in the zip file for this lab: lab1_2024.m and test_lab1_2024.p
- To test your code, just type `lab1_2021` on the command line. The program will report whether your code is returning the correct answers or not. If it is, you are done. If not, you can go back and correct things. No charge!
- When you are satisfied with your code, type `publish('lab1_2024', 'pdf')`. For some reason, Matlab will create a directory called 'html' and will put the .pdf file in there. I will only accept .pdf files. Please don't submit .html or .docx files.

1.6 General comments

The following general comments apply to this and all subsequent lab assignments.

- You will submit your .pdf file via Canvas in a form that's readable by a Windows machine. If multiple files are required in future assignments, you can zip them if you wish.
- You should be prepared to demonstrate that your code works if asked by your instructor.
- You are encouraged to help each other solve problems, but YOU ARE EACH RESPONSIBLE FOR DESIGNING AND CREATING YOUR OWN WORK. If you merely copy your neighbor's lab work, you will both get exactly the same grade: ZERO.