1

Signals and Systems Laboratory

1.1 Purpose

In this book, 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, and introduces Matlab's object-oriented programming capabilities, which are at the heart of this assignment.
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, you'll use Matlab's object-oriented programming language to create a `sequence` class that solves this problem.

1.3 Your assignment

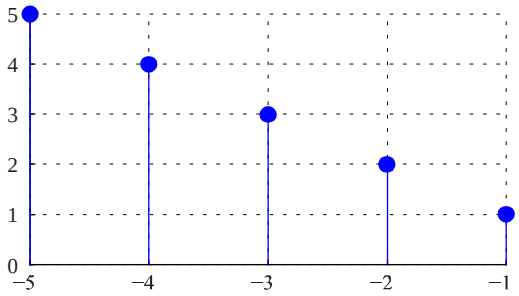
To begin, copy the sequence.m file, here. This contains the `sequence` class constructor and stubs for the methods that perform the basic DSP operations we have been learning: flipping, shifting and adding sequences. Here are the functions that you will write:

```
% function y = flip(x)
% FLIP Flip a Matlab sequence structure, x, so y = x[-n]

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

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

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

% function z = times(x, y)
% TIMES 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 stem(x)
% STEM Display a Matlab sequence, x, using a stem plot.
```

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

```
» x = sequence([1 2 3 4 5], -1);
```

Then, `stem(flip(shift(x, 2)))` will produce something like the following:



Pretty nice, right? Note that:

- All overloaded methods that operate on sequences (e.g., `plus`, `minus`, `times` and `stem`) must live in the methods block of the sequence.m file.
- Any method you put inside the methods block automatically has access to the properties (`data` and `offset`) of your `sequence` object. Hence, `flip` and `shift` must also be in the methods block if you want to access the properties easily.
- You can always access the properties `data` and `offset` of the object as if it were a simple structure:

```
» x = sequence([1 2 3 4], -1);
» x.data
ans =
    [1 2 3 4]
```

- Your `flip`, `shift`, `plus`, `minus` and `times` methods must return a `sequence` object. For example, here's a bogus implementation of the `plus` function:

```
function s = plus(x, y)
   data = x.data + y.data;
   offset = x.offset + y.offset;
   s = sequence(data, offset);
return
```

Here we first access the `data` and `offset` elements of the `sequence` object within the `plus` methods in order to create a new properties `data` and `offset`. Then we must call the `sequence` constructor with the new `data` and `offset` as arguments. The constructor returns a new `sequence` object `s`, which we then return from the `plus` function.

- You must properly handle the arguments to the `plus`, `minus` and `times` functions. Each of these functions has two arguments. Both arguments could be `sequence` objects, or one of the arguments could be a number. Hence, all of the following should work:

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

```
» x + 1
ans =
      data: [2 3 4 5]
    offset: -1

» 1 + x
ans =
      data: [2 3 4 5]
    offset: -1
```

- There should be no leading or trailing zeros in your answers. For example:

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

You can use the Matlab `find` command to determine where the zeros are in your data, and strip them off.

- 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 = 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.
- Consider using reversed indices to index into an array, if necessary. For example, `x(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)` gives the length of the array.
- You can concatenate two matrices (or 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 sequence 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 state-

ments 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 `'.'`)
- Remember that you can figure out the class of a variable using Matlab's `class` or `isa` functions:

```
» class(x)
ans =    sequence
» isa(x, 'sequence')
ans =    1
```

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

- When you modify parts of the `methods` block of your sequence.m file, it is possible that Matlab may complain. To fix this, you should clear all variables and classes as follows:

```
clear
clear classes
```

1.5 Submitting your assignment

- Your methods should have the same form as the ones described in this write-up. Make sure your code is commented.
- You should download the following files: sequence.m, lab1.m and test_lab1.p
- To test your code, just type `lab1` 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', '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.
- Make sure that your sequence file has all lab partners' names

1.6 General comments

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

- You will submit your .pdf file via iLearn 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.