San Francisco State University

Engineering 451

Laboratory #1 - Introduction to Matlab

Purpose

This is a brief tutorial on Matlab to help you get started. 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, FORTAN, 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.

Background

Read Appendix C in the notes. It gives a brief tutorial on Matlab, and introduces Matlab's object-oriented programming capabilities, which are at the heart of this assignment.

Your assignment

In this course, almost all your work will be to create Matlab functions to perform various tasks. This week's assignment is to help you become familiar with some of these Matlab operations.

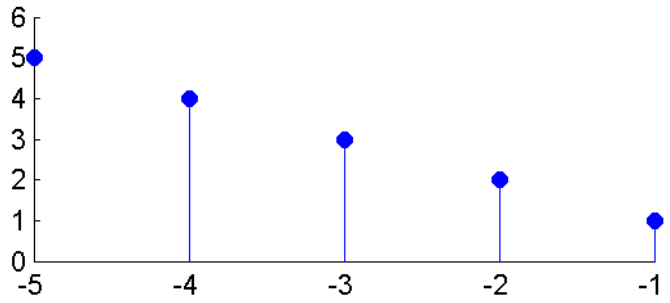
Once you have created the *sequence.m* file, and the sequence constructor as described above, you will write a series of methods to perform the basic DSP operations we have been learning: flipping, shifting and adding sequences. Here are the functions that you will write:

```
% 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
structures, or one of them may be a number.

% function stem(x)
% STEM Display a Matlab sequence, x, using a stem plot.
```

I've included the stubs for these functions in the *sequence.m* file that you downloaded above. When you are have properly written your functions 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 sweet, right? Note:

1) All *overloaded* methods (`plus`, `minus`, `times` and `stem`) that operate on sequences must live in the methods block of the *sequence.m* file.
2) 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.
3) 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]
```

4) Your `flip`, `shift`, `plus`, `minus` and `times` functions must return a sequence. For example, here's a bogus `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 function in order to create a new numeric variables `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.

5) You must properly handle the arguments to the `plus`, `minus` and `times` functions. Each of these functions has two arguments. Both arguments could be sequences, or one of the arguments could be a float 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
```

6) There should be no leading or trailing zeros in your data. 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.

7) 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
```

Some 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 them, use them. This is a course in signal processing with Matlab, not a course 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 sequence. Also, `length(x)` gives the length of the array.
- You can concatenate two sequences, `x` and `y`, by using `[x y]`.
- To pad the beginning or end of a sequence 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 sequence with two zeros and the end with three zeros. Note that `zeros(1, 3)` is not the same as `zeros(3, 1)`.

  ```
  » [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 sequences of equal length, investigate 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 functions 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
  ```

Submitting your assignment

- Your functions should have the same form as the ones described in this writeup. 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.

- When you are satisfied with your code, type `publish lab1`. I want only .pdf files. Please don't submit .html or .docx files
- You will submit this directory via iLearn in any form that's readable by a Windows machine (e.g. you can zip and rar if you wish). I'll give you the details in class

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

- You should be prepared to demonstrate that your code works, but **ONLY IF ASKED** by the instructor (c'est moi).
- 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.**
