Assignment 3: Polynomials and Sequences

EC602 Design by Software

Fall 2017

Contents

1 Introduction

1.1 Assignment Goals

The assignment goals are to

- introduce polynomials
- introduce sequence data types
- introduce discrete-time signals

1.2 Due Date

This assignment is due 2017-09-26 at midnight.

1.3 Group Size

For this assignment, the maximum group size is 3.

1.4 Submission Link

You can submit here: week 3 submit link

2 Background: Polynomials

The standard way to write a polynomial in x containing only positive powers of x is

$$p(x) = \sum_{i=0}^{N} c_i x^i = c_N x^N + c_{N-1} x^{N-1} + \cdots + c_1 x^1 + c_0$$

If negative powers of x are involved, then you can write

$$p(x) = \sum_{i=-M}^{N} c_i x^i = c_N x^N + c_{N-1} x^{N-1} + \cdots + c_1 x^1 + c_0 + c_{-1} x^{-1} + \cdots + c_{-M} x^{-M}$$

The coefficents can be integer, real, or complex. The two most common cases we will see are integer and real. The polynomial variable x can also be integer, real, or complex (there are also more general types of polynomials on other algebraic systems).

2.1 Representing Numbers as Polynomials.

When we write an integer, say 451, or a real number, say 31.4159, we are actually using a shorthand for a polynomial in 10, as follows:

$$451 = 4x^2 + 5x^1 + 1x^0$$

where $x = 10$

and

$$31.4159 = 3x + 1 + 4x^{-1} + 1x^{-2} + 5x^{-3} + 9x^{-4}$$

where $x = 10$ again.

For convenience, to establish uniqueness, and by convention, when we write a number this way we also restrict the coefficients to be *integers* in the range

$$0 \leq c < x$$

Numbers can also be written in base 2 (binary), which is important because computer memory is binary.

For example, 16.125 can be represented as

$$16.125 = 1x^4 + 1x^{-3}$$

where $x = 2$

and so $16.125 = 1000.001_2$

3 Background: Sequence Types

3.1 Sequences in Python

A sequence is a generic term for a data type that has a "length" and so can be considered "one-dimensional."

In python, the primary sequence types are

- `str`: immutable sequence of characters. Each character is also a `str` of length 1.
- `list`: a mutable sequence with an `append` method.
- `tuple`: an immutable sequence

The key property of a sequence is that the data it holds can be looked up by *location* or *index*, so

```
>>> Apples = [6,8,9]  # create a 3-element list called Apples
>>> print(Apples[1])  # printout the middle value of Apples
8
```

The items in a sequence of length `N` are numbered 0, 1, 2, and so on up to `N-1`

Note that `list` and `tuple` can both hold any data type at each element, and these can be different.

For example,

```
Fridge = ['eggs',6,(1,2)]
```

is a list with a string, integer, and tuple as its elements.

3.2 Sequences in C++

In C++, the most commonly used sequence types are:

- C-style array
- vector
- string
- array

Here is an example of defining each of these.

```
#include <string>
#include <vector> // vectors are indexable and variable size.
#include <array>  // arrays are indexable and fixed size.

string s;
array<int,3> position;
vector<int> v;
int a[6];    // this is a built-in "C" array.
```

In C++, every element of a sequence must be the same type.

3.3 Representing polynomials in C++

We will represent a polynomial in x using a `vector`. The value of the polynomial x is not stored, but instead is implicit.

Suppose the size of the vector `v` is `N`, then the polynomial represented is

$$v[N-1]x^{N-1} + v[N-2]x^{N-2} + \cdots v[1]x^1 + v[0]$$

4 Part A: Polynomial Operations

Write a C++ library file `polyops.cpp` which contains two functions `add_poly` and `multiply_poly`

The signature for both functions must exactly match the following:

```
typedef vector<double> Poly;

// Add two polynomials, returning the result
Poly add_poly(const Poly &a,const Poly &b);

// Multiply two polynomials, returning the result.
Poly multiply_poly(const Poly &a,const Poly &b);
```

You will need to include `<vector>` in your program, but you must not include any other library.

Since your program will be included into a test program, you must not have a `main()` function in this file.

Your program will be tested using

```
#include "polyops.cpp"
```

You should write your own main program to test your functions.

Here is an example main program: polyops main example

The filename of the program submitted must be `polyops.cpp`

5 Part B: Numbers as polynomials

Write a C++ library file to perform arbitrary precision integer multiplication in C++.

Here is the function signature:

```
typedef string BigInt;
BigInt multiply_int(const BigInt &a,const BigInt &b);
```

The values will be stored as the base-10 representation of the number, i.e. as strings of ASCII characters `'0'` to `'9'`.

So, for example,

```
string b = "111111";
string c = "1111111";
string d = multiply(b,c);
cout << d << endl;
```

The output should be 123456654321

You may want re-use your work from part A to help with this program. You will need to include `<string>` in your program, and you are allowed to include `<vector>` but you must not include any other library.

Since your program will be included into a test program, you must not have a `main()` function in this file.

Your program will be tested using

```
#include "bigint.cpp"
```

You should write your own main program to test your functions.

Here is an example: bigint main example

The filename of the program submitted must be `bigint.cpp`

6 Part C: Application: Discrete-Time Signal Processing

6.1 Signals Review

6.1.1 Signals

A discrete-time signal $x[n]$ can be represented as a python sequence `x`, and for convenience we assume that $x[n] = 0$ for $n < 0$ and for $n \geq N$ so that we only need to store N possible non-zero values.

We use the notation $\delta[n]$ to mean the signal which is 1 at $n = 0$ and zero everywhere else. Hence the signal $x[n] = 3\delta[n-2] + \delta[n]$ can be represented as a list:

```
x= [1 0 3]
```

and so mathematically, $x[2] = 3$ and inside python, `x[2]` is also 3.

6.1.2 Systems

A discrete-time linear invariant system (DT LTI system) has an input $x[n]$ and an output $y[n]$ and is described by a convolution equation:

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} h[n-i]x[i]$$

and for our particular case, because $x[n]$ is time-limited, so can also write:

$$y[n] = h[n] * x[n] = \sum_{i=0}^{N-1} h[n-i]x[i]$$

The system is fully described by its impulse response $h[n]$, which is also a discrete-time signal.

A common representation is the *block diagram*, a crude one is given here:

```
           ---------
          |         |
x[n] -->  |  h[n]   | --> y[n]
          |         |
         -----------
```

6.1.3 Polynomials and Sequences

If one thinks of the sequence `x` as representing a polynomial:

$$X(p) = \sum_{n=0}^{N-1} x[n]p^n$$

then it turns out that convolution of the sequences is the same operation as multiplication of the polynomials, i.e.

$$Y(p) = H(p)X(p)$$

6.2 The assignment

Using the facilities of `numpy`, write a python program that reads the input signal as a space separated sequence of numbers on one line, and the system impulse response as a space separated sequence of numbers on the next line, and outputs (on one line) the output of the DT LTI system.

The input lines should be interpreted as

```
x[0] x[1] x[2] ... x[N-1]
h[0] h[1] h[2] ... h[M-1]
```

and the output should be produced in the same order:

```
y[0] y[1] y[2] ... y[S-1]
```

where S is related to N and M.

Your program should be called `system.py`

7 Checking your program

7.1 Prerequisites

This weeks checkers require a supporting script, which must be in your working directory along with the checker and your program.

It is ec602lib.py. Please download it.

The checker for part C (`system.py`) requires new software for your devbox, "pyminifier"

Please run

```
pip install pyminifier
```

from a terminal to install the script.

7.2 The checkers

Here are the checkers:

- polyops_checker.py
- bigint_checker.py
- system_checker.py

8 Clarifications

To avoid having to re-read this document, we will add clarifications and additional information here.

8.1 Checkers

See section above on checkers.

8.2 Part A: Polynomial Operations

- we cannot represent polynomials in negative powers of x, like $1 + x^{-2}$, using a simple vector representation.
- the polynomial 0 should be represented by a vector of length one, with a zero coefficient.
- The vector length N should always match the highest power with a non-zero coefficient, which should be x^{N-1}

8.3 Part B: Numbers as polynomials

- negative values need not be handled, and you will not be tested on negative values
- zero should be handled, and should be represented (when printed) as "0"

8.4 Part C: Systems

- the output should always be of size N+M-1, even if all or some of the values are zero

8.5 Group Size

For this assignment, the maximum group size is 3.

8.6 Submission Link

You can submit here: week 3 submit link