Problem 4: Design of linear-phase FIR filters via linear programming

2005 ICCAD/SIGDA CADathlon

November 2, 2005

1 Introduction

This problem asks you to implement a program that designs linear-phase finite impulse response (FIR) filters.

A ***FIR filter*** is an array of L real numbers

$$a_0, a_1, \ldots, a_{L-1} \ ,$$

where L is the ***filter length***.

A filter is usually applied to a (discrete-time) input signal x_t, where t is an integer, to produce an output signal y_t as follows:

$$y_t = \sum_{0 \le k < L} a_k x_{t-k} \ .$$

If the input signal is the sine wave $x_t = e^{j\omega t}$, where $j = \sqrt{-1}$, then the output signal is

$$y_t = \sum_{0 \le k < L} a_k e^{j\omega(t-k)} = x_t \sum_{0 \le k < L} a_k e^{-j\omega k} = x_t H(\omega) \ .$$

Thus, the filter outputs a sine wave of the same frequency as the input, while multiplying the wave amplitude by $H(\omega)$. The function $H(\omega)$ is therefore called the ***frequency response*** of the filter.

The ***filter design problem*** is the problem of finding the coefficients a_k such that $|H(\omega)|$ has a desired shape. In general, for finite filter length L, we cannot obtain a desired $H(\omega)$ exactly, and we must resort to a suitable approximation.

In this problem, we focus on ***linear phase*** FIR filters, in which the coefficient a_k enjoy one of the following two symmetries:

Even symmetry: $a_k = a_{L-1-k}$. In this case, we have

$$H(\omega) = e^{-j(L-1)/2} A(\omega) \ ,$$

where $A(\omega)$ is a weighted sum of *cosine* functions.

Odd symmetry: $a_k = -a_{L-1-k}$. In this case, we have

$$H(\omega) = j\, e^{-j(L-1)/2} A(\omega) \ ,$$

where $A(\omega)$ is a weighted sum of *sine* functions.

In either case, we have $|H| = |A|$, which makes the filter design problem easier because approximating the real function A easier than approximating an arbitrary complex function H. With a slight abuse of language, we call $A(\omega)$ the ***frequency response*** as well, since the relation between A and H is fixed by the filter symmetry.

Steiglitz et al. [1] propose a filter design methodology where $A(\omega)$ is constrained to be in the range $[L(\omega), U(\omega)]$, for given upper and lower bounds L and U. Specifically, their method finds a filter whose frequency response $A(\omega)$ satisfies the constraints

$$L(\omega) + y \leq A(\omega) \leq U(\omega) - y$$

and $y \geq 0$ is as large as possible. A filter that maximizes y is said to be ***optimal*** (in the L_∞ norm sense).

In this CADathlon, we ask you to use Steiglitz's approach to solve a slightly modified problem. Instead of using upper and lower bounds, we aim at approximating a desired frequency response $F(\omega)$. Our goal is to find the optimal linear-phase filter whose frequency response $A(\omega)$ satisfies the constraints

$$|A(\omega) - F(\omega)| \leq w(\omega) \cdot (-y) \ , \tag{1}$$

where $(-y)$ is as small as possible, and $w \geq 0$ is a given real function (the ***weight***). We use $-y$ instead of y for consistency with the use of y in the paper, and our weight function avoids the funny notion of "hugged" constraints employed in the paper. Observe that if $w(\omega_0) = 0$, then $A(\omega_0) = F(\omega_0)$: the filter's response must exactly match the desired response at ω_0. Conversely, if we specify that $w(\omega_0) = \infty$, then $A(\omega_0)$ can assume an arbitrary value.

The constraints from Eq. 1 are equivalent to the contraints

$$F(\omega) + w(\omega)y \leq A(\omega) \leq F(\omega) - w(\omega)y \ ,$$

which can be solved by means of linear programming as in the paper.

In order to keep the input format simple, we restrict $F(\omega)$ to be piecewise linear, and $w(\omega)$ to be piecewise constant.

2 Problem statement

2.1 Brief description

Write a program that designs optimal linear-phase FIR filters given a filter length L, a filter symmetry, a desired frequency response F, and a desired weight function w.

2.2 Input specification

The first line of the input has the form

`<SYMMETRY> <FILTER-LENGTH> <SAMPLING-FREQUENCY>`

The `<SYMMETRY>` is one character, either '`E`' (for even symmetry) or '`O`' (for odd symmetry). The `<FILTER-LENGTH>` is an integer. The `<SAMPLING-FREQUENCY>` is in Hz and a real number.

Each subsequent line of the input is called a ***specification***, and it has the form

`<LEFT-FREQ> <RIGHT-FREQ> <LEFT-VAL> <RIGHT-VAL> <WEIGHT>`

Such a line specifies that F(`<LEFT-FREQ>`) = `<LEFT-VAL>`, F(`<RIGHT-FREQ>`) = `<RIGHT-VAL>`, and that values of F in the range [`<LEFT-FREQ>`, `<RIGHT-FREQ>`], are obtained by linear interpolation. In the same range, the weight function w has the constant value `<WEIGHT>`.

All frequencies are relative to the sampling frequency. To convert a frequency f into an angular frequency ω, use the formula

$$\omega = 2\pi f / f_0 \ ,$$

where f_0 is the sampling frequency.

2.2.1 Example input

The following input specifies a filter of even symmetry, length 10, running at 60 Hz. Its frequency response is 1 in the range 0–10 Hz and 0 in the range 20–30 Hz. It is therefore a low-pass filter.

```
E 10 60
0 10 1 1 1
20 30 0 0 1
```

2.3 Output specification

The program must print the L filter coefficients a_0, a_1, …, a_{L-1} in this order, one per line. Use the `printf` format `%g` followed by a newline character.

2.3.1 Example output

In response to the example input in Section 2.2.1, your program should produce something like the following output:

```
0.019804
-0.040647
-0.0739521
0.134029
0.447932
0.447932
0.134029
-0.0739521
-0.040647
0.019804
```

3 Additional information

3.1 Programming infrastructure

We do provide a parser for the input format, and a linear programming solver. Your task is to write the function `solve` in `src/solution.c` that creates a linear program, invokes the linear programming solver, and prints the result.

3.1.1 Parser

The parser is in `src/main.c:parse()`. It reads the input format and sets the four variables `symmetry`, `sampling_frequency`, `filter_length`, and `specifications` that are declared at the top of the file. The variable `specifications` is a linked list of specifications, each corresponding to an input line. The `next` field points to the next element in the linked list, and a null pointer denotes the end of the list.

3.1.2 Linear programming solver

We provide the LPPRIM linear programming solver. The following example shows how to use it.

Assume that you want to solve the linear program:

$$\min c^T x \qquad \text{subject to } Ax = b \text{ and } x \geq 0,$$

where A is an `nrow` $\times$ `ncol` matrix, and the sizes of x, c, and b are implied.

You first create a "simplex object"

```
SIMPLEX *S = simplex_make(nrow, ncol);
```

Then you set the "objective function" $c^T x$ as follows:

```
simplex_set_objective(S, c);
```

where `c` is an array of `ncol` `double`s.

Then, for $j = 0, 1, 2, \ldots, \texttt{nrow} - 1$, you add equation $A_j x = b_j$, where A_j is the j-th row of A, as follows:

```
simplex_add_equation(S, row, rhs);
```

where `row` is an array of `ncol` `double`s containing the j-th row of A, and `rhs` is the `double` value of b_j (the right-hand side of the equation).

Finally, you call

```
simplex_solve(S);
```

If the return value of `simplex_solve(S)` is not `S_OPTIMAL`, then you have a bug. (A general linear program might return `S_INFEASIBLE` or `S_UNBOUNDED`, but all our test cases have a unique optimal solution.)

You can read the solution x of the problem in `S->primals`, an array of `ncol` `double`s. More importantly, `S->duals` (an array of `nrow` `double`s) contains the solution y of the dual problem

$$\max b^T y \qquad \text{subject to } A^T y \leq c.$$

3.2 Hints

- The four cases even/odd symmetry and even/odd filter length are all slightly different. You may want to focus on even symmetry and odd length first, as in [1], and then solve the other cases.

  Your program should work in all four cases, but partial credit will be given if you don't get all cases right.

- Make sure that your program works when `<LEFT-FREQ>` = `<RIGHT-FREQ>`.

- We provide sample inputs in the `tests/` directory, together with the expected output.

  Numerical differences between your output and the output that we provide are normal. Two filters with moderately different coefficients can still have a very similar frequency response. Your output will be judged in the frequency domain, not in the time domain.

- As a debugging aid, it may be helpful to plot the frequency response of your filter to see whether it makes sense. For your convenience, program `src/plot_response.c` reads the filter coefficients and produces a file of pairs $\omega, |A(\omega)|$, which you can plot using, e.g., `gnuplot`, or

  ```
  fir <../tests/01.in | plot_response | graph -TX
  ```

References

[1] K. Steiglitz, T. W. Parks, and J. F. Kaiser. METEOR: a constraint-based FIR filter design program. *IEEE Transactions on Signal Processing*, 40(8):1901–1909, August 1992.