Universidad de Buenos Aires
Facultad De Ingeniería
Año 2016 - 2do Cuatrimestre

Algoritmos y Programación II (95.12)

TRABAJO PRÁCTICO N°0
TEMA: Promedio móvil de módulos de complejos
FECHA: 28 de septiembre de 2016

INTEGRANTES:

Andreasen, Ricardo - #96322
<ra_95_1@hotmail.com>
Manso, Juan - #96133
<juanmanso@gmail.com>

Resumen

El siguiente trabajo práctico tiene como objetivo el diseño e implementación de un programa en C++, ejercitando los conceptos básicos del lenguaje y documentando dicha implementación.

1. Diseño e implementación

1.1. Diseño del algoritmo

En el presente informe se detalla la implementación de una herramienta para procesar señales moduladas en amplitud, de acuerdo al siguiente esquema:

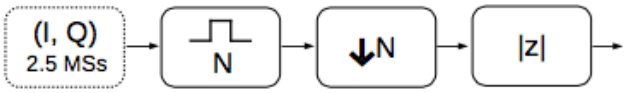

Figura 1: Flujo de procesamiento de señales.

Se provee como entrada una secuencia de números complejos $(I[n], Q[n])$ que representan las muestras en fase y cuadratura de la señal modulada. Esta secuencia se pasa por un *moving average*, un decimador y finalmente se le calcula el módulo punto a punto para obtener a la salida una secuencia de números reales $R[n']$.

En primer lugar se definió el algoritmo mediante el cual se realizaría el procesamiento de señales.

Se tuvo en cuenta el hecho de que, a partir del *downsampling*, se desecha una cantidad importante de muestras. Específicamente, si se fijó un parámetro de decimación N, de cada N muestras procesadas sólo una es finalmente utilizada. Por lo tanto sólo hace falta realizar los cálculos para las muestras que sabemos van a "sobrevivir" la decimación.

Para cada muestra n, el *moving average* realiza un promedio de las N últimas muestras recibidas, es decir sigue la ecuación:

$$y[n] = \frac{1}{N} \sum_{k=0}^{N-1} x[n-k].$$

De lo anterior se sigue que cada muestra a la salida $y[n_0]$ es sólo función de los N últimos valores a la entrada $x[n_0], x[n_0 - 1], \ldots, x[n_0 - N + 1]$. Los demás bloques de procesamiento sólo dependen del valor actual de su entrada, por lo que finalmente en el programa sólo hace falta recordar las últimas N muestras que se recibieron para realizar los cálculos.

Con estas observaciones en mente, se optó por el siguiente algoritmo de implementación:

mientras el *stream* de datos no termine **hacer**
  $promedio := 0$
  para las N siguientes muestras **hacer**
    $promedio \leftarrow promedio + x[n]$
  fin para
  $promedio \leftarrow promedio/N$ **devolver** $\sqrt{\mathbf{Re}\{promedio\}^2 + \mathbf{Im}\{promedio\}^2}$
fin mientras

1.2. Implementación

De acuerdo a lo requerido, la implementación de la herramienta se realizó en C++, para entradas y salidas de texto en los formatos especificados. Para ello se aprovecharon las clases `cmdline` y `complex` provistas en el curso, para el manejo de argumentos y números complejos respectivamente.

Con respecto al manejo de argumentos, para utilizar la clase `cmdline` hace falta definir la tabla de opciones que se esperan recibir, de tipo `option_t`. Se optó por definirla dentro del `main` del programa, con las siguientes opciones:

```
static option_t options[] = {
    {1, "i", "input", "-", opt_input, OPT_DEFAULT},
    {1, "o", "output", "-", opt_output, OPT_DEFAULT},
    {1, "N", "n_decimator","500",opt_n_decimator, OPT_DEFAULT},
    {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
    {0, },
```

Como es requerido, se definió a la cadena "-" (tercera columna de la tabla de opciones) como el valor por defecto para las opciones de entrada y de salida. De esta manera, ya sea que se haya omitido esa opción o se la especifique explícitamente para que asuma su valor por defecto, en ambos casos se obtendrá el mismo resultado.

Además la clase `cmdline` pide especificar las funciones que se utilizarán para parsear cada opción (5ta columna de la tabla) - se las definió en el mismo `main`. Para las funciones que parsean las opciones de entrada, salida y ayuda se aprovecharon las que ya estaban implementadas en el archivo `main.cc` provisto con la implementación de la clase `cmdline` del curso. Quedó por definir, entonces, el `parser` para el parámetro N de decimación:

```
static void
opt_n_decimator(string const &arg)
{
    istringstream iss(arg);

    // Intentamos extraer el N de la linea de comandos.
    // Para detectar argumentos que unicamente consistan de
    // numeros enteros, vamos a verificar que EOF llegue justo
    // despues de la lectura exitosa del escalar

    if(!(iss >> n_decimator) || !iss.eof()) {
        cerr << "non-integer factor: "
             << arg
             << "."
             << endl;
        exit(1);
    }

    if (iss.bad()) {
        cerr << "cannot read integer factor."
             << endl;
        exit(1);
    }
}
```

Como el método `parse` de `cmdline` ya le pasa al `parser` su valor por defecto si fue omitido, siempre esta función recibirá un argumento para N (no necesariamente válido). Se intenta leer el mismo como un entero, y de fallar se anuncia el error y se termina el programa.

Los argumentos se leen mediante `cmdline.parse(·)` a cada una de las variables estáticas definidas fuera del `main`:

```
static size_t n_decimator;  // Decimator Factor (factor positivo de decimacion)
static istream *iss = 0;    // Input Stream (clase para manejo de los flujos de entrada)
static ostream *oss = 0;    // Output Stream (clase para manejo de los flujos de salida)
static fstream ifs;         // Input File Stream (derivada de la clase ifstream que deriva de istream para el manejo de archivos)
static fstream ofs;     // Output File Stream (derivada de la clase ofstream que deriva de ostream para el manejo de archivos)
```

Estas son las variables que se le pasan a `am_proc` que es la encargada de implementar el algoritmo de la sección anterior, de manera que el `main` queda:

```
int
main(int argc, char * const argv[])
{
    cmdline cmdl(options);  // Objeto con parametro tipo option_t (struct) declarado globalmente.
    cmdl.parse(argc, argv); // Metodo de parseo de la clase cmdline
    am_proc(iss, oss, n_decimator); // Procesamiento AM
}
```

La función `am_proc` está declarada en "`am_proc.h`" y definida en "`am_proc.cc`". Fuera de chequear por errores de lectura, su implementación es en escencia el pseudo-código de la sección anterior:

```
void am_proc(istream *is, ostream *os, const size_t& n_decimator){

    bool eof_flag=false;
    size_t i;
    complejo c, aux; // c tendra la suma y aux sera el que recibe el complejo del stream


    // Si entra un archivo vacio (primero lee EOF), corta el for y luego el while, devolviendo un vacio

    while(!eof_flag){

        // Se suman los primeros 'n_decimator' numeros hasta que corte
        for(i=1; i<=n_decimator && ((*is)>>aux); i++)
            c += aux;

        // Compruebo si se llego a EOF
        if(is->eof())
            eof_flag=true;
```

```
19
20          if(is->bad()){
21          // El for termino por no poder guardar el caracter en x
22              cerr    << "Error: Cannot read complex on input stream"
23                  << endl;
24              exit(1);
25          }
26
27          // Realizo el proemdio movil
28          c=c / n_decimator;
29
30          // Imprimo el valor absoluto
31          *os << c.abs()<<endl;
32      }
33
34      if(os->bad()){
35          cerr    << "Error: Cannot write output file"
36              << endl;
37          exit(1);
38      }
39
40  }
```

2. Análisis

3. Corridas de prueba

4. Código fuente