

Trabajo Práctico 1

Procesamiento de señales AM.

Juan Manso, 96133
Ricardo Andreasen, 96322

Introducción

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 $R[n']$.

Algoritmo

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 $x[n_0], x[n_0 - 1], \dots, x[n_0 - N + 1]$. 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  $\sqrt{\text{Re}\{\textit{promedio}\}^2 + \text{Im}\{\textit{promedio}\}^2}$ 
fin mientras
```

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:

```
1 static option_t options[] = {
2     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
3     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
4     {1, "N", "n_decimator", "500", opt_n_decimator, OPT_DEFAULT},
5     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
6     {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:

```
1 static void
2 opt_n_decimator(string const &arg)
3 {
4     istringstream iss(arg);
5
6     // Intentamos extraer el N de la l nea de comandos.
7     // Para detectar argumentos que nicamente consistan de
8     // n meros enteros, vamos a verificar que EOF llegue justo
9     // despu s de la lectura exitosa del escalar.
10
11     if(!(iss >> n_decimator) || !iss.eof()) {
12         cerr << "non-integer factor: "
13             << arg
14             << "."
15             << endl;
16         exit(1);
17     }
18
19     if (iss.bad()) {
20         cerr << "cannot read integer factor."
21             << endl;
22         exit(1);
23     }
24 }
```

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`:

```

1 static size_t n_decimator; // Decimator Factor (factor positivo de decimaci n)
2 static istream *iss = 0; // Input Stream (clase para manejo de los flujos de
   entrada)
3 static ostream *oss = 0; // Output Stream (clase para manejo de los flujos de
   salida)
4 static fstream ifs; // Input File Stream (derivada de la clase ifstream que
   deriva de istream para el manejo de archivos)
5 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:

```

1 int
2 main(int argc , char * const argv [])
3 {
4     cmdline cmdl(options); // Objeto con parametro tipo option_t (struct) declarado
   globalmente.
5     cmdl.parse(argc , argv); // Metodo de parseo de la clase cmdline
6     am_proc(iss , oss , n_decimator); // Procesamiento AM
7 }

```

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 esencia el pseudo-código de la sección anterior:

```

1 void am_proc(istream *is , ostream *os , const size_t& n_decimator){
2
3     bool eof_flag=false;
4     size_t i;
5     complejo c , aux; // aux_c tendr la suma y aux_x ser el que recibe el complejo
   del stream
6
7
8     // Si entra un archivo vacio (primero lee EOF), corta el for y luego el while ,
   devolviendo un vacio
9
10    while(!eof_flag){
11
12        // Se suman los primeros 'n_decimator' n meros hasta que corte
13        for(i=1; i<=n_decimator && ((*is)>>aux); i++)
14            c += aux;
15
16        // Compruebo si se lleg a EOF
17        if(is->eof())
18            eof_flag=true;
19
20        if(is->bad()){
21            // El for termin 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 promedio m vil
28     c=c / n_decimador;
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 }

```

Compilación

Corridas de prueba

Conclusiones