

Filtrarea și spectrul unui semnal

În această lucrare de urmărește extinderea aplicației de achiziție monocanal prin aplicarea filtrelor și analiza spectrală.

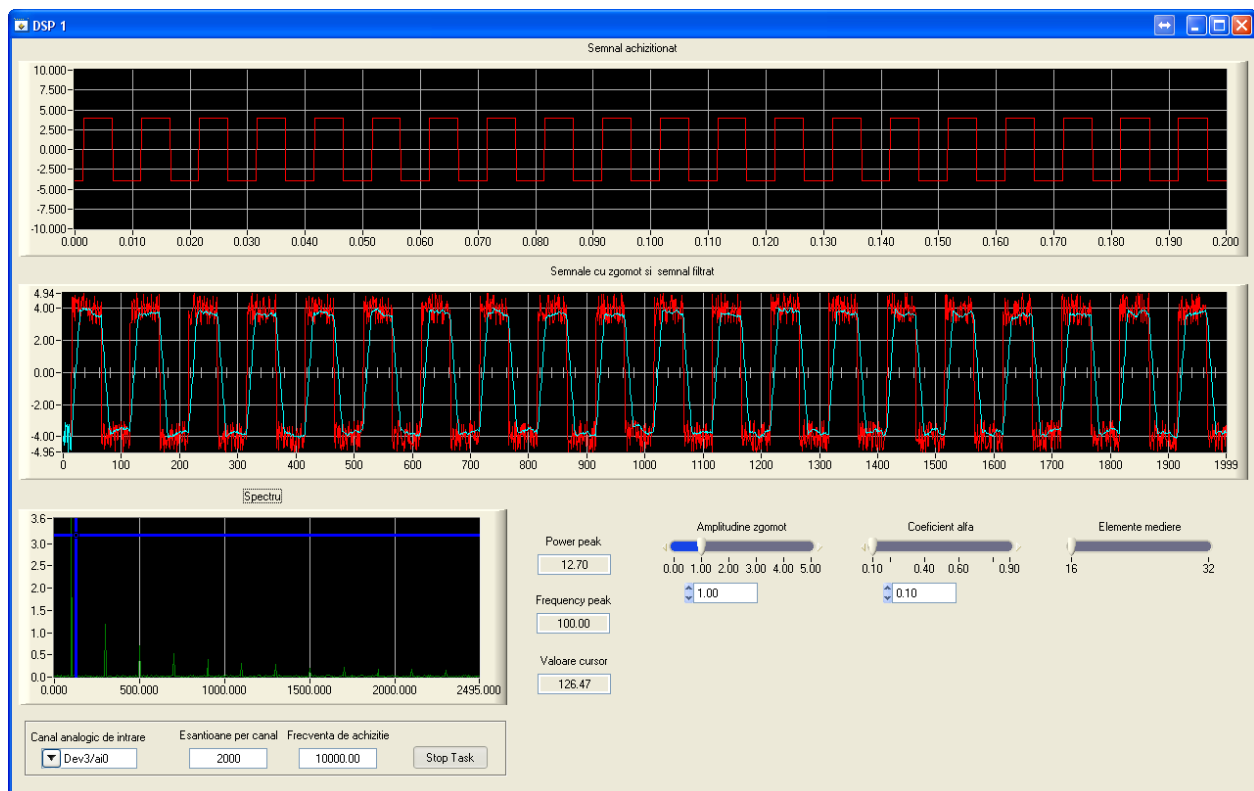
1. Filtrarea în domeniul timp

La semnalul achiziționat, se adună zgomot alb utilizând funcțiile CVI (semnificația argumentelor este evidentă):

- generare zgomot alb:
`WhiteNoise(ssize_t nElements, double amplitude, int seed, double noise[]);`
- adunare a doi vectori:
`Add1D(double arrayX[], double arrayY[], ssize_t numberOfElements, double outputArray[]);`

Se implementează funcțiile pentru filtrare prin mediere (pe 16 sau 32 de elemente) și filtrare cu un element de ordin I ($filt[i] = (1 - \alpha) * filt[i-1] + \alpha * signal[i]$). Pe interfața aplicației se creează un control pentru selecția tipului de filtru. Pentru filtrul de ordin I, valoarea parametrului alpha se va fixa prin intermediul unui control numeric în intervalul (0÷1).

În aplicația dezvoltată pentru achiziția monocanal, introduceți un control tip Graph (dacă acesta nu există) pentru reprezentarea semnalului cu zgomot și a celui filtrat. Semnalul achiziționat se reprezintă pe un control tip graph. Aplicația ar trebui să arate similar cu cea prezentată mai jos:



Interfața aplicației

Există diferențe între cele două tipuri de filtre?

Ce diferențe există între semnalul achiziționat (fără zgomot) și semnalul filtrat?

Alte tipuri de filtre vor fi studiate în laboratoarele viitoare.

2. Spectrul de putere

Pentru a obține spectrul de putere a semnalului achiziționat se vor utiliza mai multe funcții CVI, care vor fi apelate în funcția *EveryNCallback* aferentă task-ului de achiziție. Ordinea apelurilor va fi următoarea:

```
ScaledWindowEx (double xArray[], ssize_t numberOfElements, int
windowType, double windowParameter, WindowConst *windowConstants);
```

unde:

- *xArray[]* – bufferul cu eșantioane;
- *numberOfElements* – dimensiunea bufferului;
- *windowType* – tipul de fereastră, care pentru moment va fi *RECTANGLE_*;
- *windowParameter* – nu interesează pentru moment;
- *windowConstants* – parametru returnat. Atenție: Acesta va trebui declarat (*WindowConst winConst*).

Această funcție este necesară deoarece, unui semnal eșantionat a cărui perioadă nu se cunoaște (dacă semnalul este periodic) i se aplică în general o fereastră cu scopul de a “aplatiza” forma semnalului la capetele intervalului de eșantioane analizat. În acest fel, fiecare buffer de eșantioane va fi asimilat cu o perioadă a semnalului (detalii la curs și în laboratoarele viitoare).

```
AutoPowerSpectrum(double inputArray[], ssize_t numberOfElements, double
dt, double autoSpectrum[], double *df);
```

Această funcție calculează partea pozitivă a spectrului scalat de putere pentru un semnal eșantionat, după formula:

$$(FFT(X) \cdot FFT^*(X))/n^2, \quad (1)$$

unde *n* reprezintă numărul de puncte din bufferul cu eșantioane iar *FFT** reprezintă transformata Fourier complex conjugată. Semnificația parametrilor rezultă din help-ul contextual. Atenție: pasul în domeniul timp *dt* trebuie să fie 1.0/dimensiune buffer.

Funcția returnează:

- *autoSpectrum[]* – spectrul de putere cu un număr de valori egal cu jumătate din dimensiunea bufferului de intrare (*inputArray*);
- *df* – pasul în domeniul frecvenței.

```
PowerFrequencyEstimate (double autoSpectrum[], ssize_t numberOfElements,
double searchFrequency, WindowConst windowConstants, double df, ssize_t
frequencySpan, double *frequencyPeak,);
```

Parametrii sunt furnizați de funcțiile precedente sau se utilizează valorile implicite. Funcția returnează:

- *frequencyPeak* – frecvența estimată pentru spectrul de putere (maxim) din vectorul *autoSpectrum*.
- *powerPeak* – valoarea maxima din spectru de putere (din *autoSpectrum*).

Căutați în Help, formulele implementate de această funcție. **Valorile *frequencyPeak* și *powerPeak* se vor afișa pe interfața grafică.**

```
SpectrumUnitConversion ((double spectrum[], ssize_t numberOfElements, int type, int scalingMode, int displayUnits, double df, WindowConst windowConstants, double convertedSpectrum[], char *unitString);
```

Funcția convertește spectrul de intrare (care poate fi puterea, amplitudinea sau amplificarea) în formate alternative (linear, logarithmic, dB) ce permit o reprezentare grafică mai convenabilă.

Precizări:

- *scalingMode* - *SCALING_MODE_LINEAR*
- *displayUnits* - *DISPLAY_UNIT_VRMS*
- *convertedSpectrum[]* – vectorul utilizat pentru reprezentarea spectrului
- *unitString* – unit, declarat în funcția *EveryNCallback* (*char unit[32]="V"*, dacă mărimea de intrare este în volți);

Cerințe:

1. Reprezentați grafic spectrul de putere (*convertedSpectrum*) cu funcția *PlotWaveform* (control tip Graph) pe un număr de puncte egal cu jumătatea dimensiunii *convertedSpectrum* și analizați rezultatul obținut.
2. Fixați frecvența de eșantionare la 2kHz și modificați frecvența semnalului de intrare (achiziționat) în intervalul 500 Hz ÷ 3 kHz. Care vor fi spectrele pentru semnale de intrare de: 0.5, 1, 1.2, 1.5, 2.5 kHz? Cum explicați rezultatele obținute?
3. Reprezentați spectrul semnalului de intrare utilizând funcția FFT a cărei implementare ilustrează modul teoretic în care am prezentat la curs Transformata Fourier Discretă (pentru detalii – fisierul “Transformata Fourier-Rezumat si proprietati.pdf” de pe Moodle):

```
AnalysisLibErrType FFT (double arrayXReal[], double arrayXImaginary[], ssize_t numberOfElements);
```

Această funcție calculează TFD ”*in place*”, adică rezultatul va fi rescris peste vectorii de intrare.

- *numberOfElements* - numărul de elemente a vectorilor de intrare ;
- *arrayXReal* – la apel reprezintă vectorul cu partea reală a semnalului de intrare, după execuție conține partea reală a transformatei Fourier ;
- *arrayXImaginary* - la apel reprezintă vectorul cu partea imaginară a semnalului de intrare, după execuție conține partea imaginară a transformatei Fourier ; dacă semnalul este real (ca în cazul nostru) la alep acest vector trebuie să fie zero.

Reprezentați cei doi vectori obținuți. Ce observați?

Utilizând cei doi vectori returnați de FFT, calculați și reprezentați spectrul de putere după formula (1). Rezultatul este similar celui obținut anterior?

Sugestie de implementare pentru cerințele de la punctul 3:

```
//Toti vectorii de mai jos au dimensiunea lui rowSignal
for(int i=0; i<nSamples; i++)
{   WfR[i]=rowSignal[i]; //partea reală
```

```
        WfI[i]=0;          //partea imaginară - o în cazul dat
    }
    FFT(WfR,WfI,nSamples);

    for (i=0;i<nSamples-1;i++)
    { //spectrul după formulă
        p[i]=(WfR[i]*WfR[i]+WfI[i]*WfI[i]);
        ps[i]=p[i]/pow((double)nSamples,2.0);
    }

//Spectrum (wfm3, 1024); //Funcție CVI- se poate încerca ca alternativă

    delta_t=1/rate;    //pas în timp
    delta_f=1/(nSamples*delta_t); //pas în frecvență
    //construiesc axa pentru frecvență
    frequency_array[0]=0.0;
    frequency_array[nSamples-1]=1/(2*delta_t);
    for (i=0;i<nSamples/2;i++)
    {
        frequency_array[i]=i*delta_f;
        frequency_array[nSamples-1-i]=-1*delta_f;
    }

//Reprezentare componentei reale și a celei imaginare
//int ploth2, ploth4

if (ploth2>0)
{
    DeleteGraphPlot (mainPanel, MAIN_PANEL_IDC_OUTGRAPH_2, ploth2, 1);
    ploth2=0;
    DeleteGraphPlot (mainPanel, MAIN_PANEL_IDC_OUTGRAPH_2, ploth4, 1);
    ploth4=0;
}

ploth2=PlotXY (mainPanel, MAIN_PANEL_IDC_OUTGRAPH_2, frequency_array, WfI,
nSamples/2, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
VAL_SOLID,1, VAL_RED);

ploth4=PlotXY (mainPanel, MAIN_PANEL_IDC_OUTGRAPH_2, frequency_array, WfR,
nSamples/2, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID,1,
VAL_GREEN);

//sau doar pentru power spectrum (ps)

ploth2=PlotXY (mainPanel, MAIN_PANEL_IDC_OUTGRAPH_2, frequency_array, ps,
nSamples/2, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
VAL_SOLID,1, VAL_GREEN);
```