Tworzenie i optymalizacja filtrów FIR

Raport 1 (25.03.2016)

Uniwersytet Jagielloński – Zakład Fotoniki

1 Wstęp

Celem niniejszego raportu jest podsumowanie i przedstawienie wyników uzyskanych przy analizie filtrów typu FIR z myślą o zastosowaniu w układach FPGA (field-programmable gate array), jak również podkreślenie istotnych problemów napotkanych po drodze.

Zapis współczynników filtru w układach FPGA zazwyczaj ogranicza się do kilkunastu bitów na liczbę celem uzyskania możliwie największej wydajności, dlatego istotnym problemem staje się znalezienie 'kompromisu' pomiędzy liczbą współczynników oraz liczbą bitów (zwanej dalej dyskretyzacją) na których zostają zapisane.

Raport składa się z dwóch części: generowanie współczynników filtru oraz ich analizy.

1.1 Właściwości transformaty fouriera (przypomnienie)

Def. 1.1

Transformację Fouriera funkcji $f(x)$ definiujemy jako $F(s) = \int_{-\infty}^{\infty} f(x)\, e^{-i\,2\pi x s}\, dx$. Odwrotną zaś

$$f(x) = \int_{-\infty}^{\infty} F(s)\, e^{i\,2\pi x s}\, ds$$

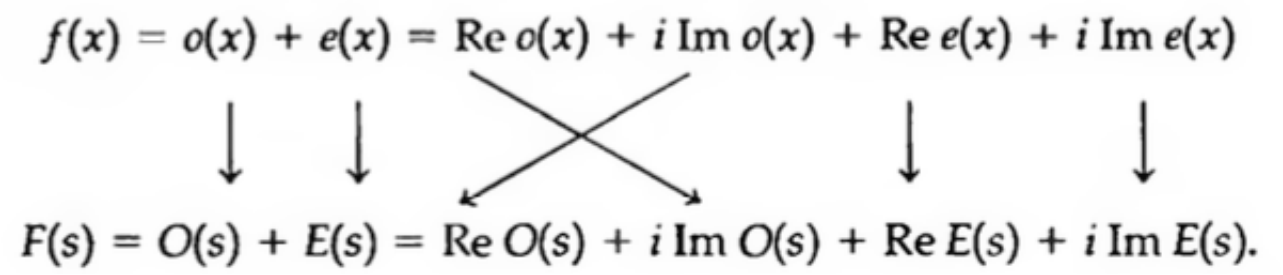
Każdą funkcję możemy rozłożyć na część parzystą i nieparzystą, tj. $f(x) = e(x) + o(x)$, gdzie:

$e(x) = \frac{1}{2}[f(x) + f(-x)]$, $o(x) = \frac{1}{2}[f(x) - f(-x)]$.

Wtedy transformata Fouriera $f(x)$ redukuje się do (całka iloczynu funkcji parzystej i nieparzystej jest nieparzysta):

$$\int_{-\infty}^{\infty} e(x)\cos(2\pi x s)\,dx - i\int_{-\infty}^{\infty} o(x)\sin(2\pi x s)\,dx$$

Widać stąd że transformata funkcji parzystej jest parzysta, a część rzeczywista pozostaje rzeczywista. Można też wyciągnąć więcej wniosków, które najlepiej podsumowywuje poniższy rysunek:



Rysunek 1.
Diagram prezentujący transformatę funkcji
(źródło 'The Fourier Transform and it's applications' – Bracewell)

Zatem, aby operować (po wykonaniu transformaty) na wielkościach rzeczywistych, musimy ograniczyć się do parzystych funkcji.
W przypadku dyskretnej transformaty fouriera oś symetrii również przechodzi przez zero co prowadzi to do nieparzystej liczby próbek.
Konkretne implementacje różnią się od siebie, jednak często (również tutaj) do zapisu sygnału stosuje się tablice numerowane od 0, dlatego zwykle część próbek (normalnie znajdująca się po stronie ujemnych współczynników) dodawana jest na koniec tablicy.

2 Znalezienie współczynników filtru

Współczynniki filtru FIR powinny zostać znalezione za pomocą kodu uruchamianego na CPU układu. Z tego powodu powstał program (afilters) w języku C++, mający na celu wygenerowanie współczynników filtru z zadanej, dyskretnej transmitancji.

Program używa istniejącej i otwartej biblioteki do transformaty fouriera zwanej KISS FTT (licencja BSD – http://kissfft.sourceforge.net).

Obecnie kod znajduje się w prywatnym repozytorium pod adresem:
https://github.com/masteranza/SignalProcessing
oraz posiada szereg funkcji diagnostycznych ułatwiających szybkie testowanie filtrów, takie jak generacja filtrów low–, mid–pass.

2.1 Format plików wyjściowych

Współczynniki generowane przez program współczynniki są rzeczywiste i jest ich nieparzysta ilość. Zdyskretyzowane współczynniki również są rzeczywiste (nie całkowite), dokładny opis procesu dyskretyzacji znajduje się w dalszej części.

Na obecną chwilę ilość danych w plikach wyjściowych jest również nadmiarowa.
Każdemu wierszowi odpowiada jeden współczynnik, jednak kolumn (dla potrzeb analizy i korekcji błędów jest wiele), poniżej przedstawiono pełne zestawienie poszczególnych (numerowanych) kolumn. Najistotniejszymi kolumnami są jednak kolumna: 1 (transmitancja), 9 (współczynniki) i 11 (zdyskretyzowane współczynniki)

1 – zadana na wejściu transmitancja używana przy obliczaniu współczynników, z możliwymi dodatkami, np. wygładzaniem stromych zboczy, domyślnie wyłączone
2 – część rzeczywista transmitancji obliczonej pomocniczo z obliczonych współczynników
3 – część urojona transmitancji obliczonej pomocniczo z obliczonych współczynników
4 – część rzeczywista transmitancji obliczonej pomocniczo z obliczonych, dyskretyzowanych współczynników
5 – część urojona transmitancji obliczonej pomocniczo z obliczonych, dyskretyzowanych współczynników
6 – wartość bezwględna współczynników ($\sqrt{\mathrm{Re}\{\mathrm{wsp}\}^2 + \mathrm{Im}\{\mathrm{wsp}\}^2}$)
7 – wartość bezwględna dyskretyzowanych współczynników
8 – numer współczynnika począwszy od 0.
9 – część rzeczywista współczynników
10 – część urojona współczynników (0)
11 – część rzeczywista zdyskretyzowanych współczynników
12 – część urojona zdyskretyzowanych współczynników (0)
13 – dodatkowa kolumna na zapis niezmodyfikowanej transmitancji w przypadku dodatków (domyślnie 0)

2.2 Przykład

Kompilacja kodu źródłowego:

```
gcc - g - lm afilters.c kiss_fft.c kiss_fftr.c - o afilters;
```

Uruchomienie i wygenerowanie filtru low–pass o szerokości 10% (przepuszczającego 10% spektrum częstości) :

```
./afilters 0.1
```

Uruchomienie i wygenerowanie filtru mid-pass o szerokości 10% i przesunięciem 30% (0% ≡ low–pass):

```
./afilters 0.1 0.3
```

Efektem działania programu jest wygenerowanie szeregu plików (używanych później w dalszej analizie) o różnych długościach filtrów i różnej liczbie bitów użytych do dykretyzacji.

W przypadku filtrów low pass generowane są pliki postaci:
filtr ⟨długość filtru⟩ – ⟨liczba bitów⟩.dat,
gdzie ⟨długość filtru⟩ przybiera wartości: 31, 63, 127, 255, 511, 1023,
a ⟨liczba bitów⟩ wartości: 0, 2, 4, …, 64.

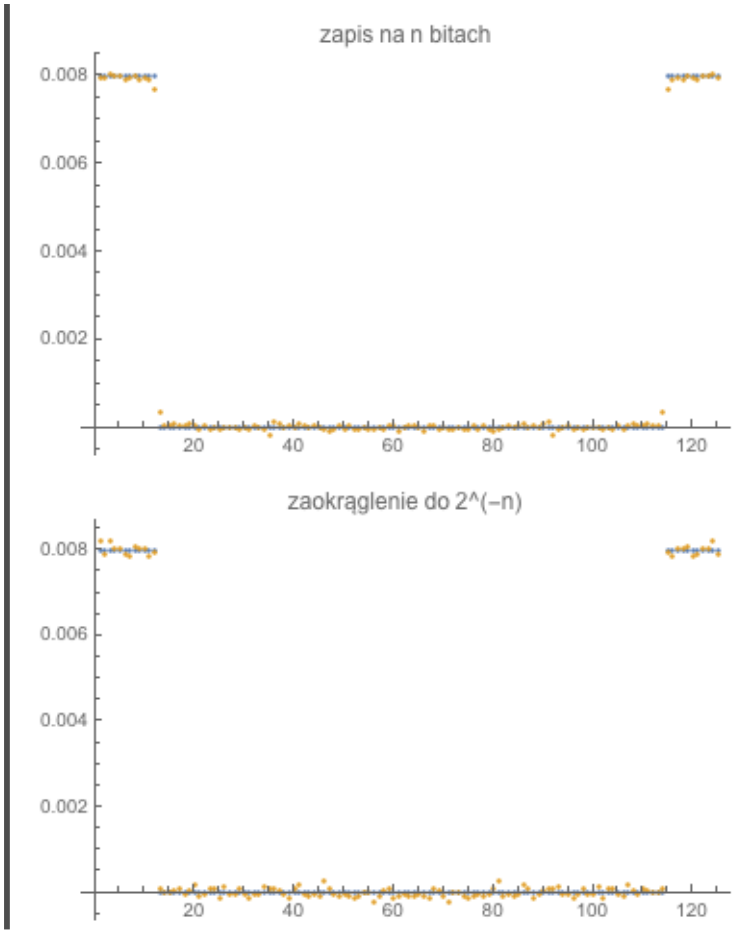
Dla filtrów mid–pass sytuacja wygląda analogicznie, zmieniają się tylko nazwy plików, dla uniknięcia kolizji: filtr ⟨długość filtru⟩ m – ⟨liczba bitów⟩.dat

2.3 Proces dyskretyzacji

Przy użyciu m bitów największą liczbą całkowitą możliwą do zapisania jest liczba $2^{m-1}-1$ (pierwszy bit użyty jest do zapisu znaku). Z tego powodu w procesie dyskretyzacji najpierw znajdywana jest największa (max) spośród liczb zapisanych w tablicy liczb x (i–ta liczba znajduje się pod $x[i]$). Następnie wszystkie liczby zostają podzielone/unormowane przez tą liczbę, a następnie przemnożone przez maksymalną liczbę $(2^{m-1}-1)$ i zaokrąglone do najbliższej liczby całkowitej również ustalając jej znak. Aby z tak zdyskretyzowanej liczby uzyskać liczbę, odpowiadającą oryginalnej liczbie mnożymy przez czynnik odwrotny, a więc $\max/\left(2^{m-1}-1\right)$.
Procedurę tę ilustruje pseudokod:

```
xd[i]  = ((x[i] > 0) - (x[i] < 0))*round ((fabs (x[i])/max)*(pow (2, m - 1) - 1))
* max / (pow (2, m - 1) - 1);
```

> Podczas prac nad analizą filtrów okazało się również, że ten proces dyskretyzacji, jest różny od znacząco różny od zaokrąglenia liczb do najbliższej wielokrotności 2^{-n}.



3 Analiza filtrów w programie Mathematica

W tej części przedstawiony zostaje końcowy zestaw skryptów użytych do analizy filtrów wraz z wynikami.

Aby skrypty mogły zostać ponownie przeliczone, ten notes aplikacji Mathematica powinien znajdować się w folderze zawierającym aplikację afilters.

3.1 Inicjalizacja

```
(*stałe*)
nn = {32, 64, 128, 256, 512, 1024};
bits = Range[2, 64, 2];
(*Utworzenie macierzy nazw wygenerowanych przez afilters plików*)
names = Outer["filtr" <> ToString[#1 - 1] <> "-" <> ToString[#2] <> ".dat" &, nn, bits];
(*wykresy*)
leeg[xl_List, d_, points_List, numb_List, nam_List, args___] := LogPlot[xl, d,
   Epilog → {PointSize[Medium], Point[points],
     Inset[
      Panel[
       Grid[
        MapIndexed[{Graphics[{ColorData[1, First@#2], Thick, Line[{{0, 0}, {1, 0}}]}],
            AspectRatio → .1, ImageSize → 20], First@nam[[#2]], First@numb[[#2]]} &, xl]]],
      Offset[{-2, 2}, Scaled[{1, 1}]], {Right, Top}]}, args];
TemperatureGrid[data_, bits_, args___] :=
  Grid[
   Prepend[
    Prepend[Map[Item[#, Background → ColorData[{"TemperatureMap", {Min[data], Max[data]}}][#]] &,
        data, {2}]ᵀ, nn]ᵀ, Prepend[bits, ""]], args];
(*ustawienie WorkingDirectory*)
AppendTo[$Path, NotebookDirectory[]];
```

3.2 Proste testy sprawdzające

Interesującym testem który można przeprowadzić jest wyrysowanie wygenerowanych współczynników oraz porównanie transformaty tych współczynników wykonanej wewnątrz programu afilters oraz w Mathematica.
W tym celu na początek generujemy współczynniki filtrów szerokości 90% za pomocą komendy:

```
./afilters 0.9
```

lub bezpośrednio z notesu Mathematica:

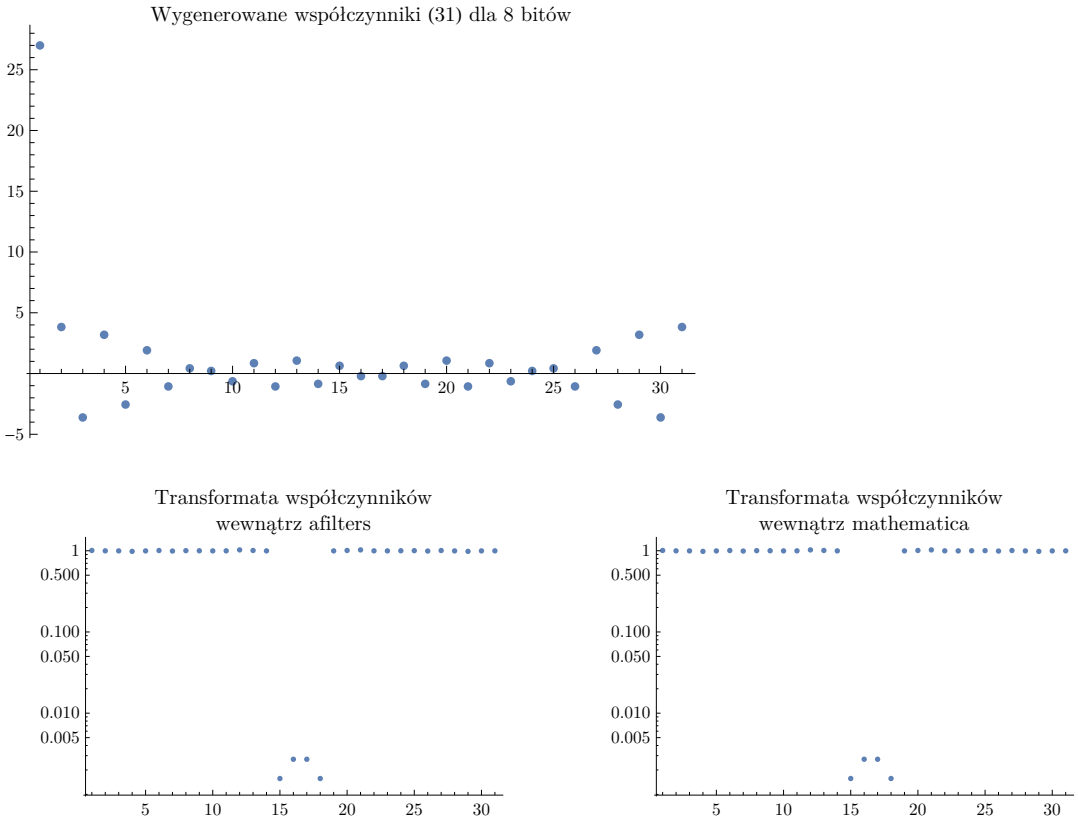
```
szerokosc = 0.9;
Run[
  OpenRead["!cd '" <> NotebookDirectory[] <>
    "'; gcc -g -lm afilters.c kiss_fft.c kiss_fftr.c -o afilters; ./afilters " <>
    ToString[szerokosc] <> ";"]];
```

Wczytujemy dane do notesu (należy zaczekać na wygenerowanie wszystkich plików):

```
(*wartość bezwzględna transmitancji z programu afilters*)
dataMag = Map[Import[#, "Data"][[All, 7]] &, names, {2}];
(*zdyskretyzowane współczynniki filtru*)
dataWsp = Map[Import[#, "Data"][[All, 11]] &, names, {2}];
```

Wybieramy dane dla przykładowej liczby współczynników: wsp = 1 ($2^{(4+\text{wsp})} = 32$) oraz liczby bitów: bit = 4 (bit $* 2 = 8$) i wyświetlamy je:

```
wsp = 1;
bit = 4;
ListPlot[dataWsp[[wsp, bit]], PlotRange → Full,
 PlotLabel → ("Wygenerowane współczynniki (" <> ToString[2^(4 + wsp) - 1] <> ") dla " <>
    ToString[bit * 2] <> " bitów")]
GraphicsRow[
 {ListLogPlot[dataMag[[wsp, bit]] / (2^(4 + wsp) - 1), PlotRange → Full,
   PlotLabel → "Transformata współczynników\nwewnątrz afilters"],
  ListLogPlot[Abs@Fourier[dataWsp[[wsp, bit]], FourierParameters → {-1, 1}],
   PlotRange → Full, PlotLabel → "Transformata współczynników\nwewnątrz mathematica"]},
 ImageSize → Large]
```



Wyniki są identyczne z dokładnością do stałej (liczby współczynników).

3.3 Użyta metoda szacowania błędu

Przy szacowaniu błędu dyskretyzacji współczynników wypróbowano wiele miar, będących wariacjami średniego błędu kwadratowego (porównując transformaty współczynników), jednak bardzo dogodną miarą okazało się obliczenie średniego błędu względnego na jeden współczynnik.

3.3.1 Średni błąd względny na jeden współczynnik

Oznaczmy spektrum filtru idealnego (składający się z N liczb) o szerokości w przez $T(w)$, a przez $T_M(w)$ filtr (również składający się z N liczb) uzyskany przy zapisie na M bitach. Wtedy nasza miara

wyraża się przez:

sredniblad

$$\sqrt{\frac{\sum_{i=1}^{M}(T_M(w)[i]-T(w)[i])^2}{\sum_{i=1}^{M}(T(w)[i])^2}} \tag{3.1}$$

lub w przypadku ciągłym:

$$\sqrt{\frac{\int_0^{\pi} dx\,(T_M(w)[x]-T(w)[x])^2}{\int_0^{\pi} dx\,(T(w)[x])^2}} \tag{3.2}$$

Oczywiście aby możliwe było obliczenie błędu w przypadku ciągłym, należy wykonać najpierw DTFT (Discrete Time Fourier Transform) na zestawie współczynników.

3.4 Generowanie filtru low–pass i szacowanie błędu w procesie dyskretyzacji na M bitach

Poniższy kod pozwala na zewnętrzne wygenerowanie współczynników filtru (poprzez program afilters), a następnie na obliczenie błędów opisanego wcześniej procesu dyskretyzacji, w zależności od szerokości i długości filtru lub ilości bitów.
Odpowiednie fragmenty mogą zostać w prosty sposób uaktywnione poprzez od–komentowanie.

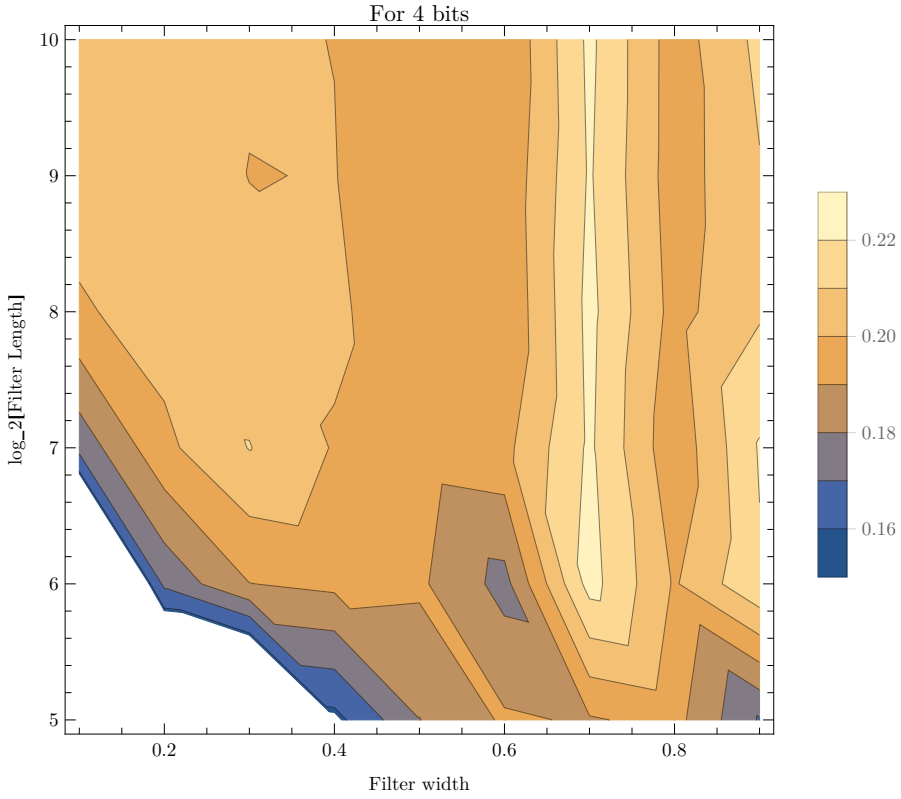
```
CalcErrors[part_: 0.4, bit_] :=
  Module[{transPlots, trans, transref, disc, dplots, start, names, data, rotData, data2,
    rotData2, dataPlot, dataPlot2, DTFT, rotDTFT, names2, dataDouble, data2Double,
    dataForInterLog, dataForInterDouble, interdDouble, idrotDTFTMeanSq, logDTFTs,
    listpoint, listpointd, dataPerfect, perfectDTFT, idDTFT},
   Run[OpenRead["!cd '" <> NotebookDirectory[] <>
      "'; gcc -g -lm afilters.c kiss_fft.c kiss_fftr.c -o afilters; ./afilters " <>
      ToString[part] <> ";"]];
   Pause[15];
   names = Outer["filtr" <> ToString[#1 - 1] <> "-" <> ToString[#2] <> ".dat" &, nn, bits];
   data = Map[Import[#, "Data"][[All, 11]] &, names, {2}];
   trans = Map[Import[#, "Data"][[All, 1]] &, names, {2}];
   rotData = Map[RotateLeft[#, (Length@# + 1) / 2] &, data, {2}];

   data2 = Map[Abs@Fourier[#, FourierParameters → {-1, 1}] &, data, {2}];
   rotData2 = Map[Abs@Fourier[#, FourierParameters → {-1, 1}] &, rotData, {2}];

   DTFT = Map[ListFourierSequenceTransform[#, x] / Length@(#) &, data, {2}];
   rotDTFT = Map[ListFourierSequenceTransform[#, x] / Length@# &, rotData, {2}];
   dataForInterLog =
    Map[{Table[i, {i, 0, 1 - 1 / (Length@#), 2 / (Length@#)}] π, Log@#[[1 ;; (Length@# + 1) / 2]]}^T &,
     data2, {2}];
   names2 = Outer["filtr" <> ToString[#1 - 1] <> "-64.dat" &, nn];
   dataDouble = Map[Import[#, "Data"][[All, 13]] &, names2];

   dataPerfect = Map[Import[#, "Data"][[All, 11]] &, names2, {1}];
   perfectDTFT = Map[ListFourierSequenceTransform[#, x] / Length@(#) &, dataPerfect, {1}];

   (*Średni Błąd względny(szerokosc, długość): miara dyskretna*)
   Table[
    {part, len + 4, Sqrt[Total[Abs[(data2[[len, bit]] - dataDouble[[len]])]^2] /
       ((Total[Abs[dataDouble[[len]]]^2] ) )]}, {len, 1, Length@DTFT}]
   (*Średni Błąd względny(szerokość, długość): miara ciągła*)
   (*Table[
    {part,len+4,
     Sqrt[NIntegrate[Abs[DTFT[[len,bit]]-perfectDTFT[[len]]]^2,{x,0,π},AccuracyGoal→5,
        MaxRecursion→20]/NIntegrate[Abs[perfectDTFT[[len]]]^2,{x,0,π},AccuracyGoal→5,
        MaxRecursion→20]]},{len,1,Length@DTFT}]*)
   (*Średni Błąd względny(bit, długość): miara dyskretna*)
   (*Table[
    {2bit,len+4,Sqrt[Total[Abs[(data2[[len,bit]]-dataDouble[[len]])]^2]/
       ((Total[Abs[dataDouble[[len]]]^2] ) )]},{len,1,Length@DTFT}]*)
  ];
```

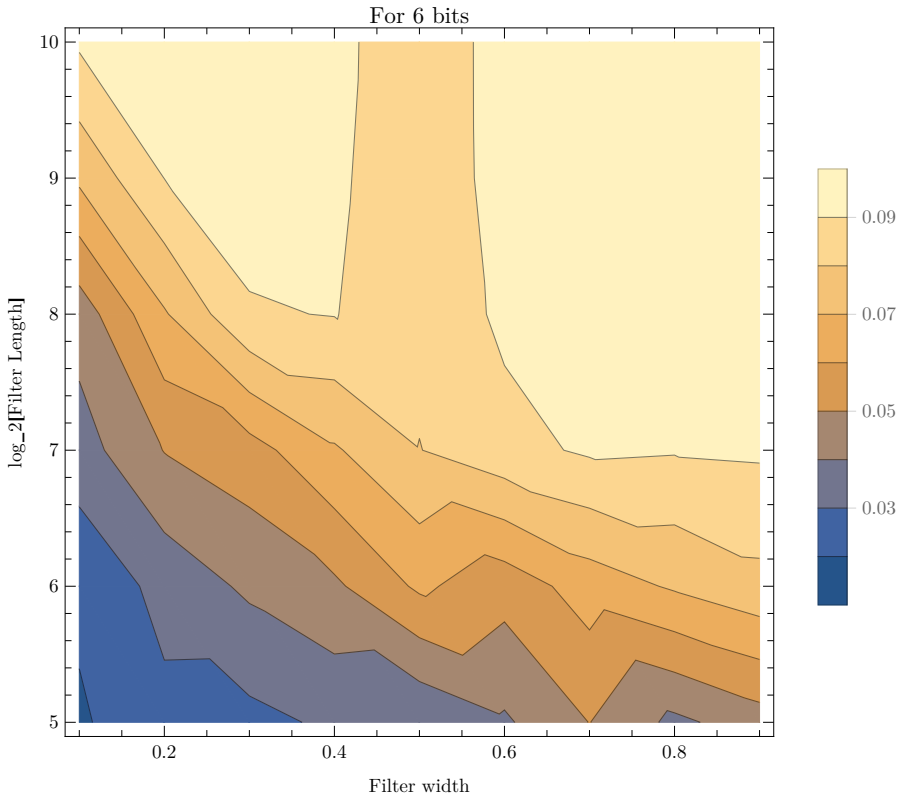
Za pomocą tego skryptu oraz:

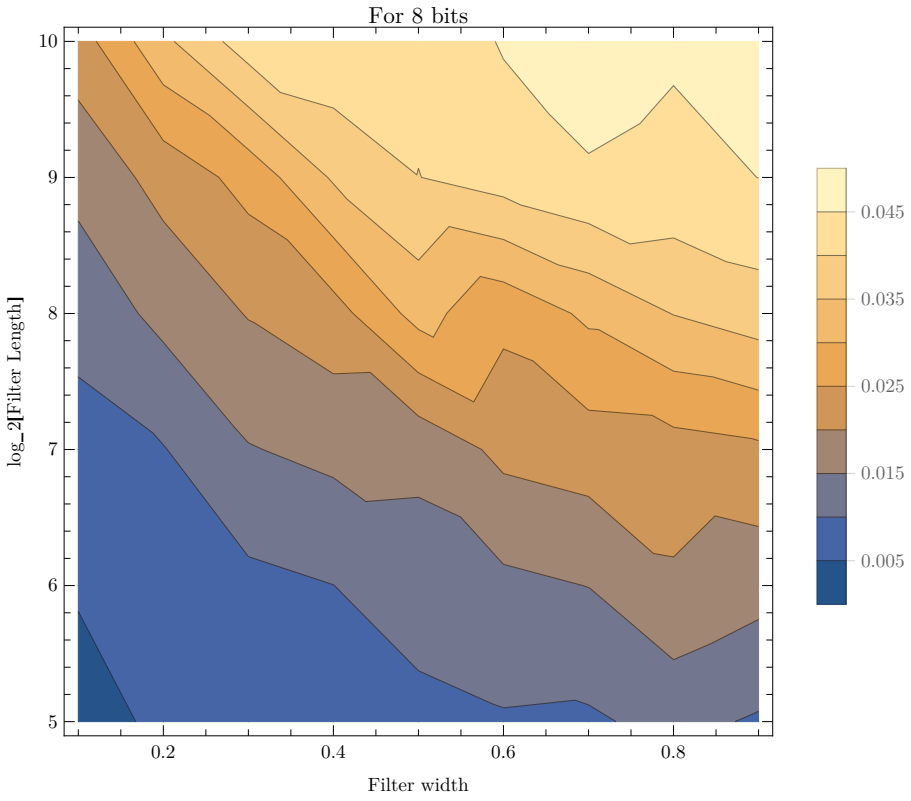
```
bit = 2; (*również dla 3,4,5,6*)
wyniki = Flatten[Table[N@CalcErrors[widt, bit], {widt, 0.1, 0.9, 0.1}], 1];
ListContourPlot[wyniki, PlotLegends → Automatic, LabelStyle → Directive[Bold],
 FrameLabel → {"Filter width", "log_2[Filter Length]"},
 PlotLabel → "For " <> ToString[bit * 2] <> " bits"]
```

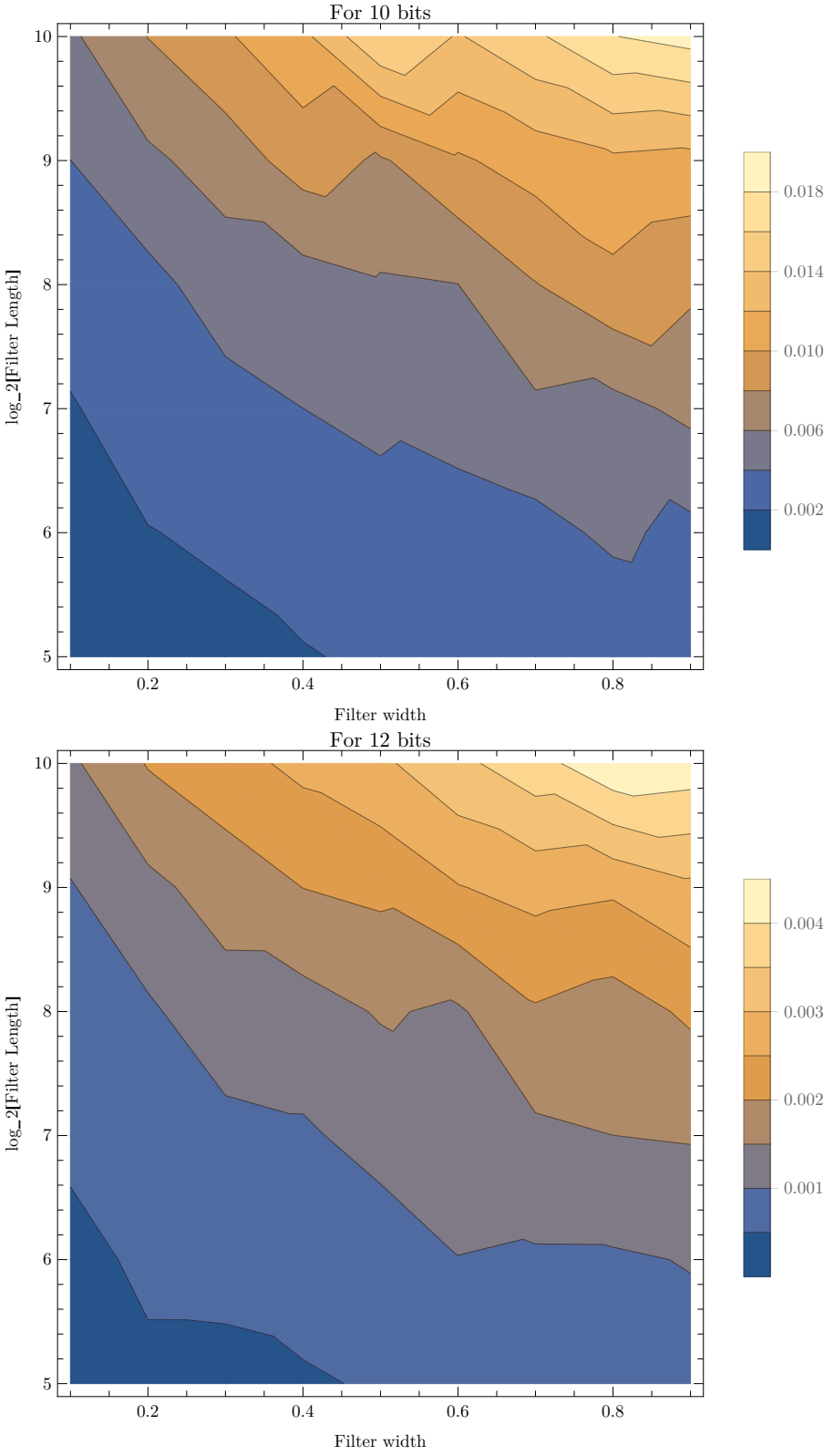
Obliczono średnie błędy względne dla różnych liczb bitów zależące od szerokości i długości filtru:

3.4.1 Dla miary dyskretnej

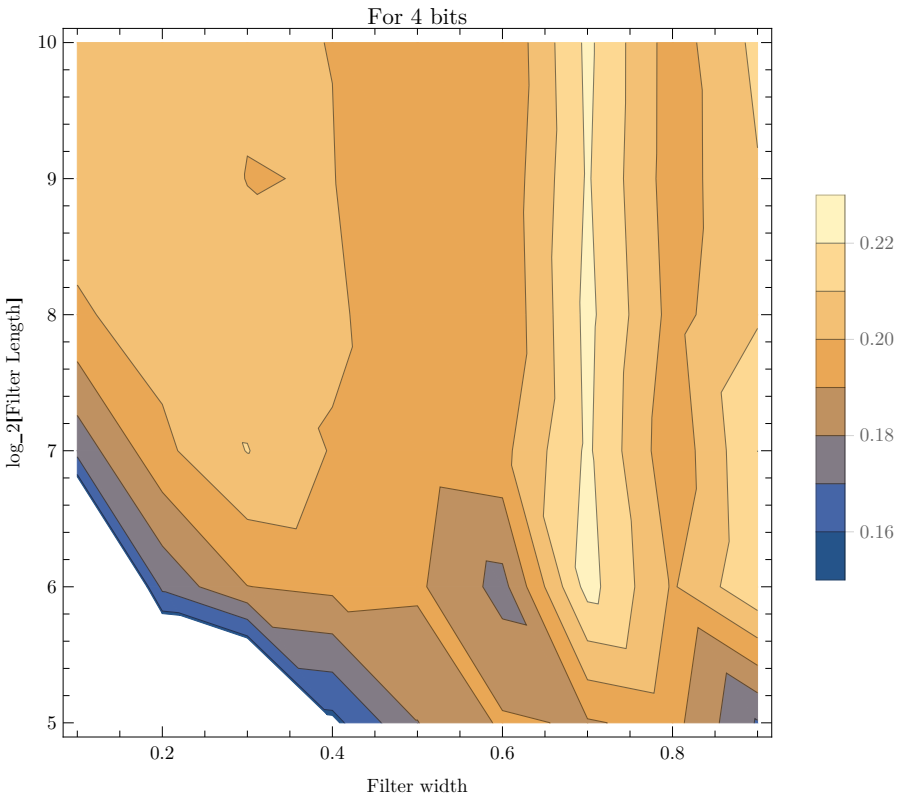


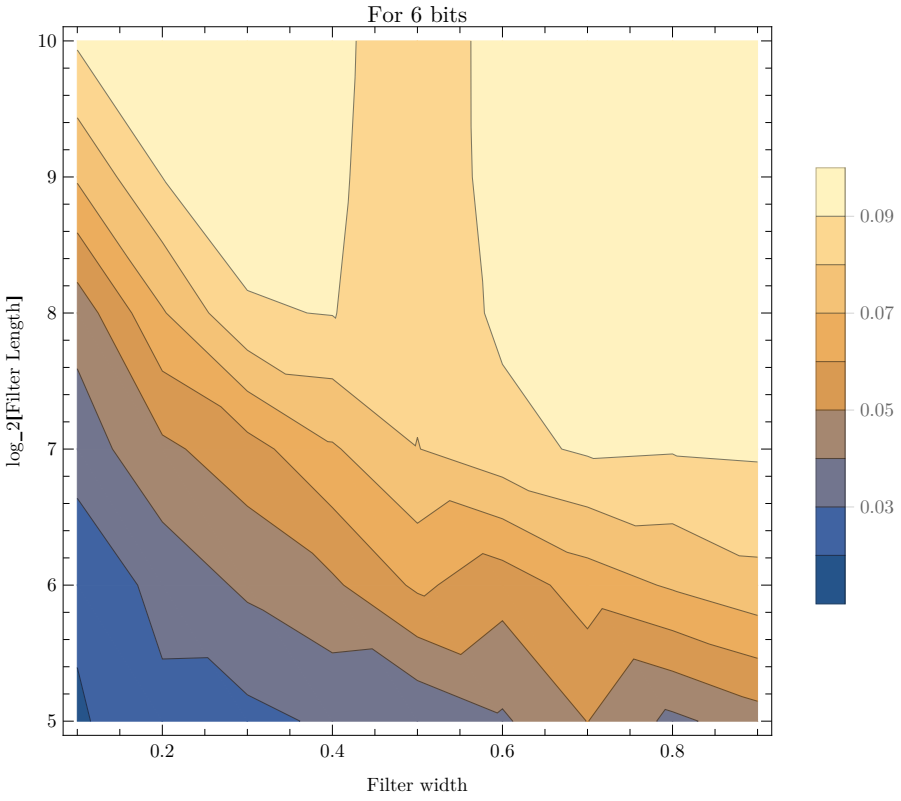

For 6 bits
log_2[Filter Length]
Filter width
10
9
8
7
6
5
0.2
0.4
0.6
0.8
0.09
0.07
0.05
0.03

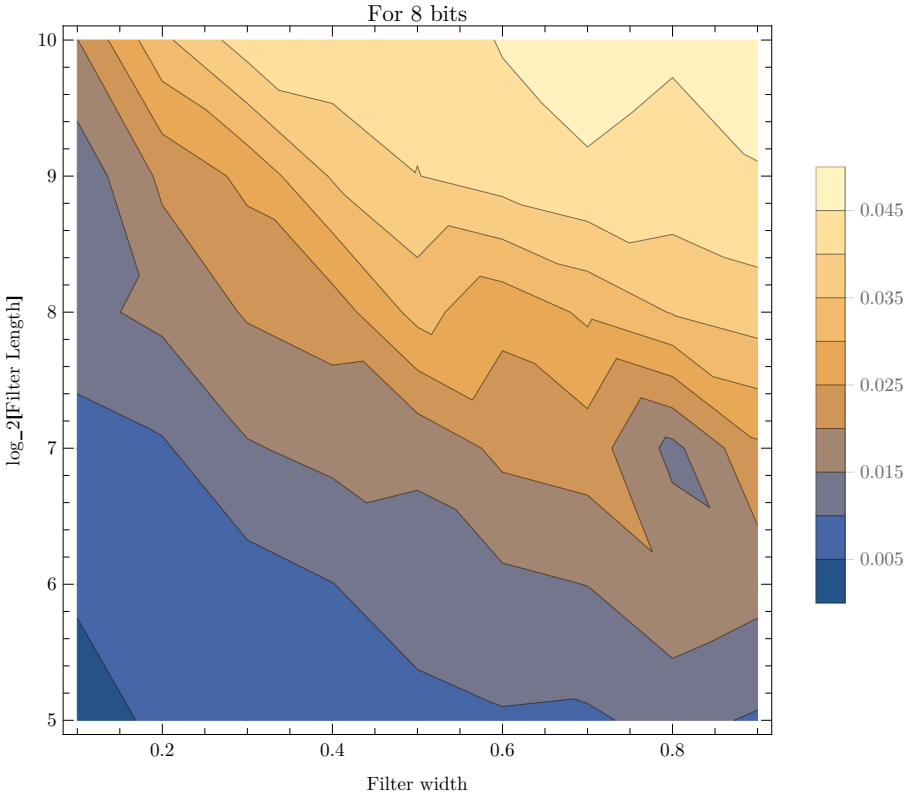

For 8 bits
log_2[Filter Length]
Filter width
10
9
8
7
6
5
0.2
0.4
0.6
0.8
0.045
0.035
0.025
0.015
0.005

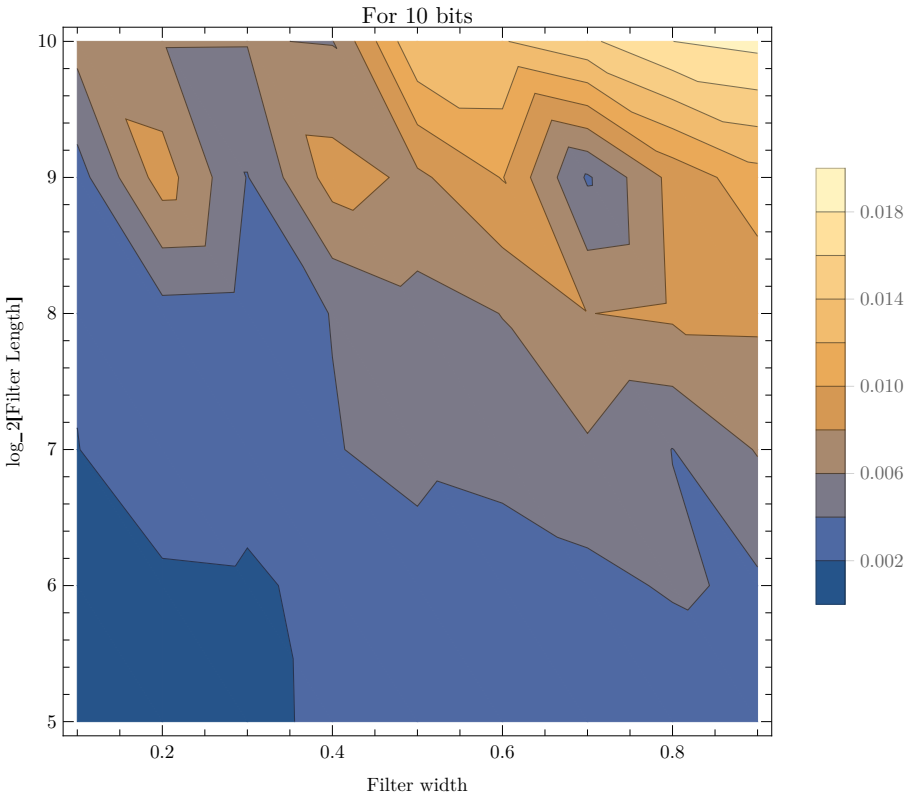


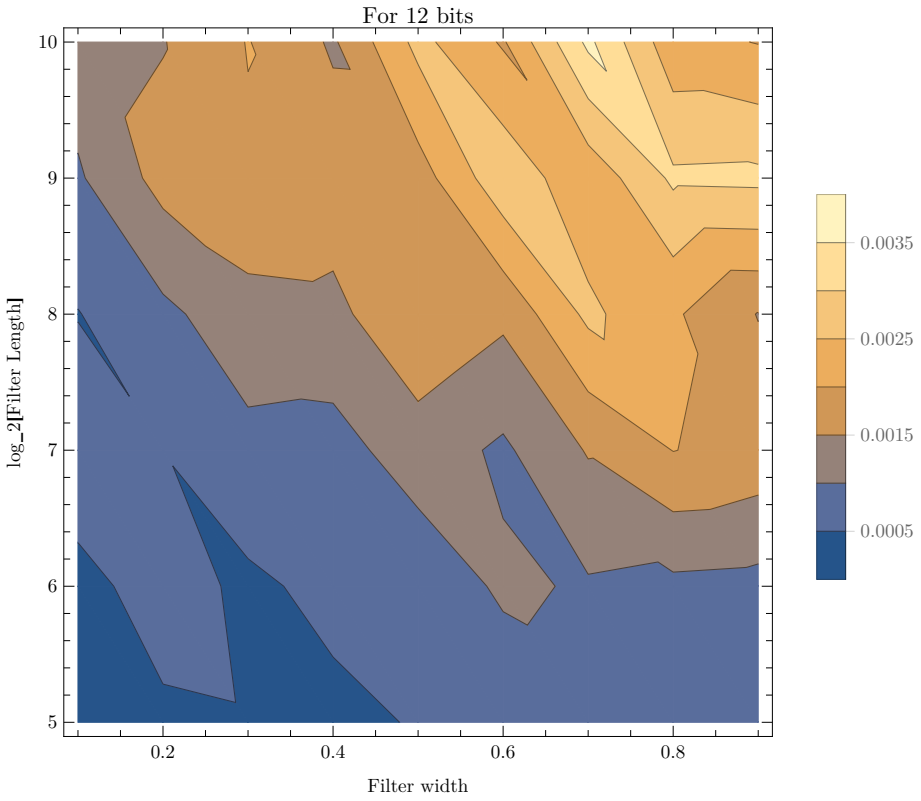
3.4.2 Dla miary ciągłej


For 4 bits
log_2[Filter Length]
Filter width
10
9
8
7
6
5
0.2
0.4
0.6
0.8
0.22
0.20
0.18
0.16


For 6 bits
log_2[Filter Length]
Filter width
10
9
8
7
6
5
0.2
0.4
0.6
0.8
0.09
0.07
0.05
0.03


For 8 bits
log_2[Filter Length]
10
9
8
7
6
5
0.2
0.4
0.6
0.8
Filter width
0.045
0.035
0.025
0.015
0.005


For 10 bits
log_2[Filter Length]
10
9
8
7
6
5
0.2
0.4
0.6
0.8
Filter width
0.018
0.014
0.010
0.006
0.002



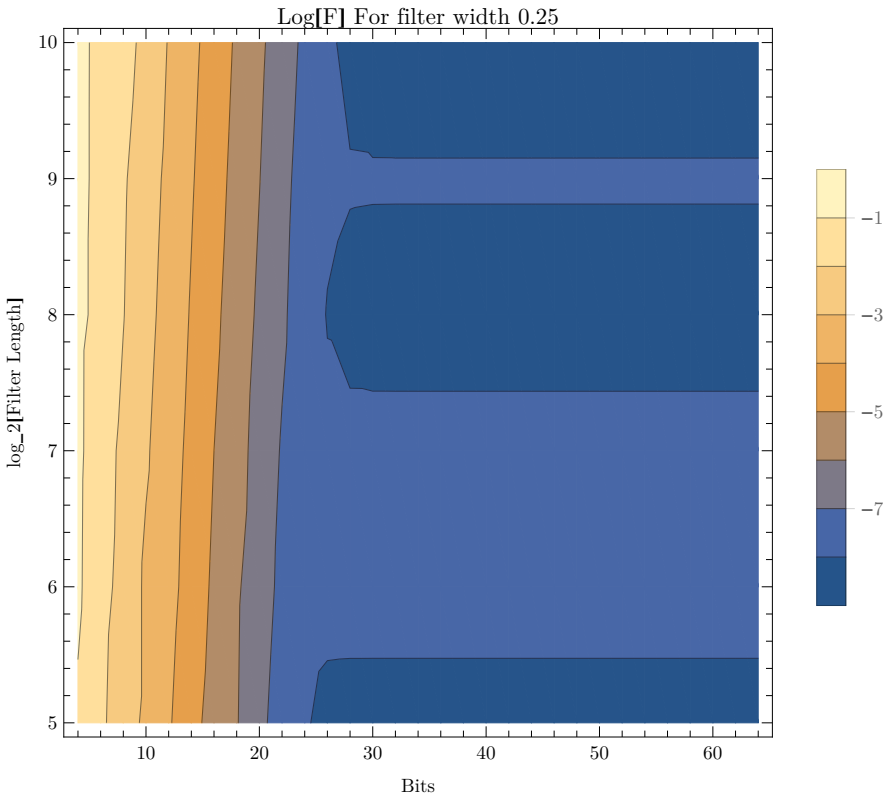
Z poniższych wykresów widać, iż obie miary dają zgodne wyniki.

3.5 Zanik błędu z rosnącą liczbą współczynników

Za pomocą podobnego skryptu:

```
wyniki4 = Flatten[Table[N@CalcErrors[0.25, bit], {bit, 2, 32}], 1];
wyniki5 = {#[[1]], #[[2]], Log@Sqrt[#[[3]]]} & /@ wyniki4;
ListContourPlot[wyniki5, PlotLegends → Automatic, LabelStyle → Directive[Bold],
 FrameLabel → {"Bits", "log_2[Filter Length]"}, PlotLabel → "Log[F] For filter width 0.25"]
```

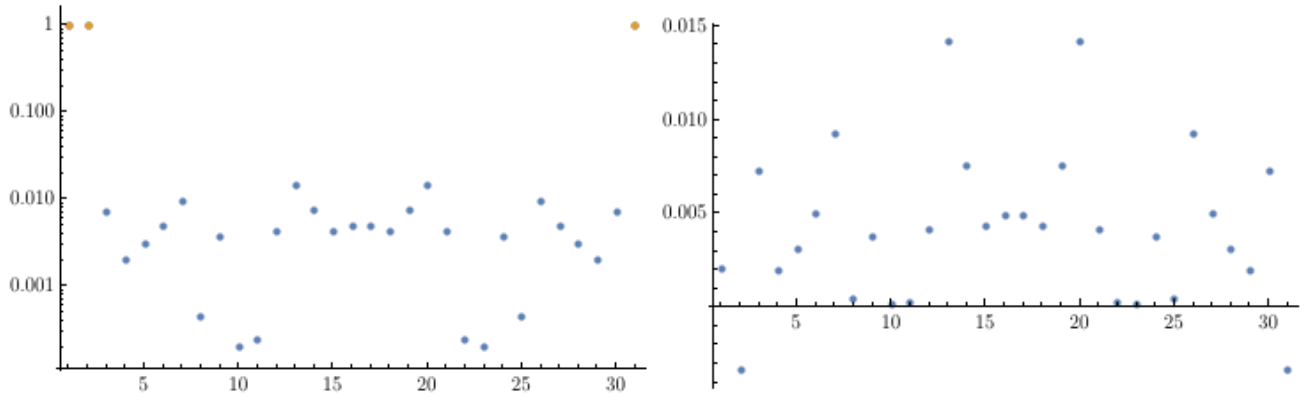
Obliczono zależność średniego błędu względnego od długości filtru i liczby bitów użytej przy dyskretyzacji, otrzymując:

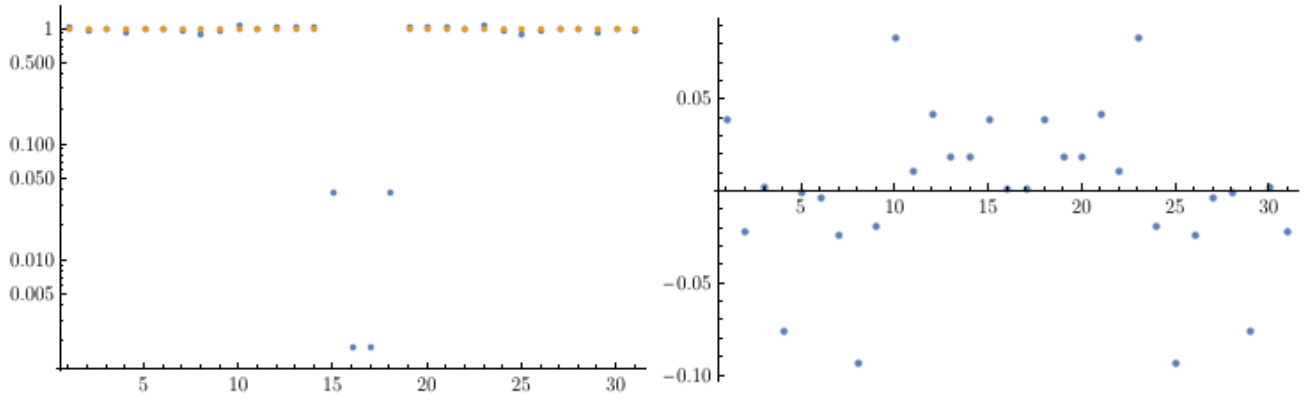


Wykres wskazuje (na spodziewany) zanik błędu z rosnącą liczbę bitów.

3.6 Problem rosnącego błędu z szerokością filtru

Nieco zastanawiający okazał się wzrost błędu przy rosnącej szerokości filtru, zwłaszcza (jak widać) w przypadku niskiej ilości bitów.
Do tej pory nie znaleziono zadowalającego wytłumaczenia tego fenomenu.
Prosta heurystyka, przeprowadzona przy 6 bitach i mierze dyskretnej daje następujące wyniki dla filtrów 10% (pierwszy wiersz) i 90% (drugi wiersz):





Pierwsza kolumna przedstawia spektrum idealne i spektrum zdyskretyzowane na jednym wykresie, zaś druga przedstawia różnicę między nimi.
Licznik równania (3.1) dla przypadku filtru 10% wynosi 0.00104495, a mianownik 3.
Dla filtru 90%, licznik wynosi 0.0555405, a mianownik 27.
Zatem błąd zdecydowanie staje się większy gdy próbujemy utworzyć szerszy filtr.

4 Podsumowanie

Przedstawiona tutaj metoda oraz zestaw skryptów pozwala na analizę jakości dowolnych filtrów z udziałem różnych miar – zarówno ciągłych jak i dyskretnych.
Dzięki uzyskanym wynikom można, w prosty sposób dopasować parametry pracy filtru FIR w zależności od możliwości dostępnego sprzętu.
Nasza wiedza jest jednak nie pełna. Do tej pory nie udało się wytłumaczyć, dlaczego średni błąd względny na jeden współczynnik rośnie wraz z długością filtru.