# Zastosowania Procesorów Sygnałowych

dr inż. Grzegorz Szwoch
greg@multimed.org
p. 732 - Katedra Systemów Multimedialnych

# Filtry IIR

# Wstęp

Na tym wykładzie: IIR – drugi typ filtrów cyfrowych.

- Jak projektować i implementować je na DSP.
- Jak chronić się przed efektami przepełnienia.
- Jak skorzystać z asemblerowych implementacji z biblioteki DSPLIB.
- W jakich sytuacjach korzystniej jest używać filtrów IIR zamiast FIR.

# Filtry IIR

IIR – *Infinite Impulse Response*

- Filtr cyfrowy o nieskończonej odpowiedzi impulsowej.
- Filtry rekursywne – wejściem są zarówno nowe próbki sygnału, jak i poprzednie wyniki filtracji.
- Transmitancja filtru IIR rzędu *N*:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}}$$

# Filtry IIR

- Filtr IIR implementuje się na podstawie równania różnicowego.
- Filtr rzędu 2 (dwukwadratowy, *biquad*):

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

- Równanie różnicowe:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)$$

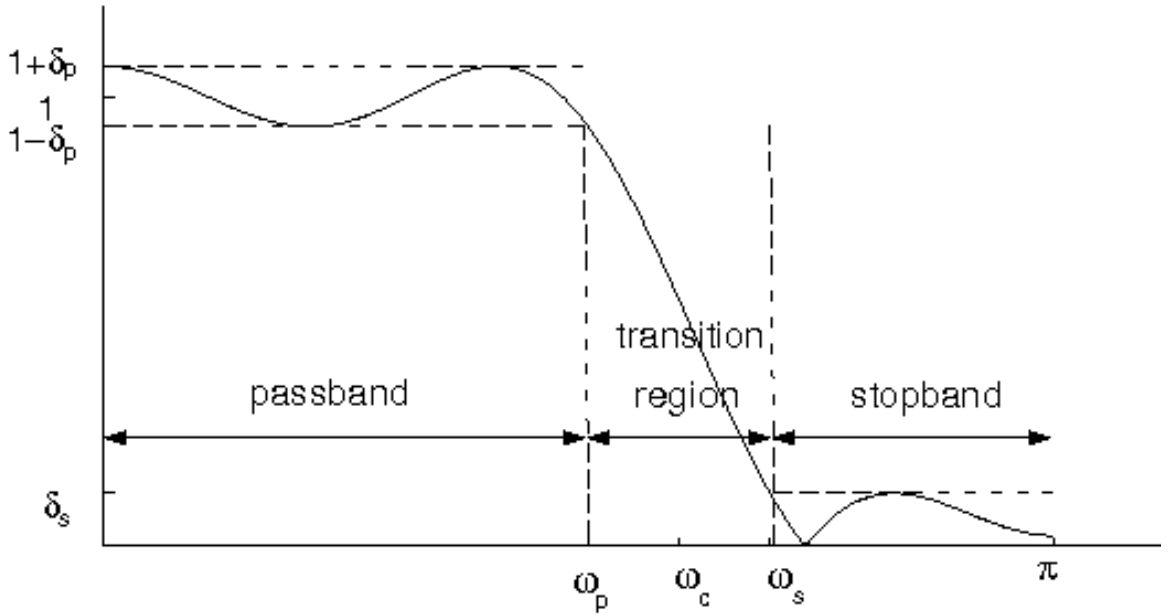
# Projektowanie filtrów IIR

- Cyfrowe filtry IIR są odpowiednikami filtrów analogowych.
- Projektuje się filtry analogowe, a następnie przekształca się je do filtru cyfrowego, zwykle za pomocą przekształcenia dwuliniowego.
- Wynikiem projektu są współczynniki $b$, $a$.
- Obecnie używamy do tego celu oprogramowania komputerowego (Matlab, python+scipy, itp.).

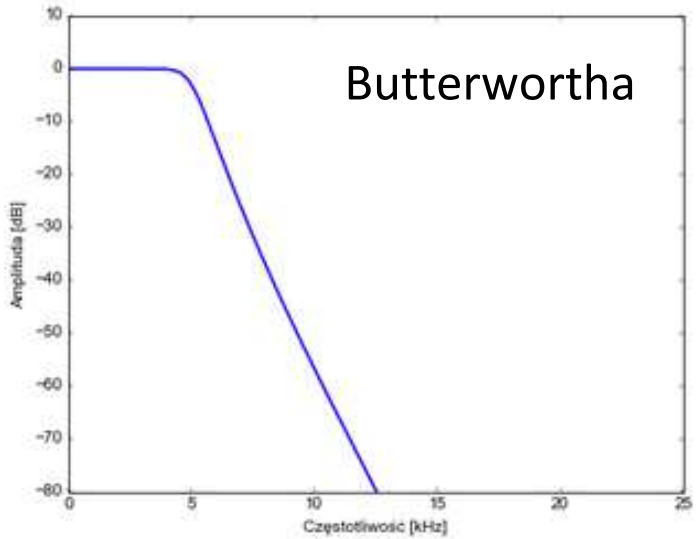
# Parametry filtru

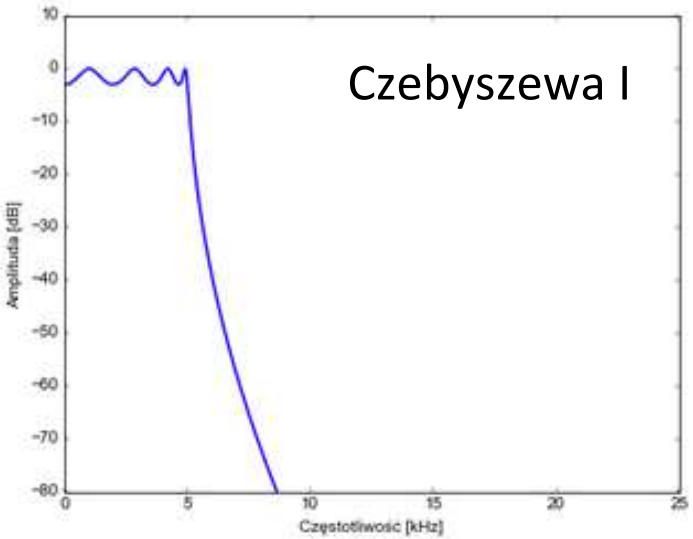
Parametry projektowe filtru:

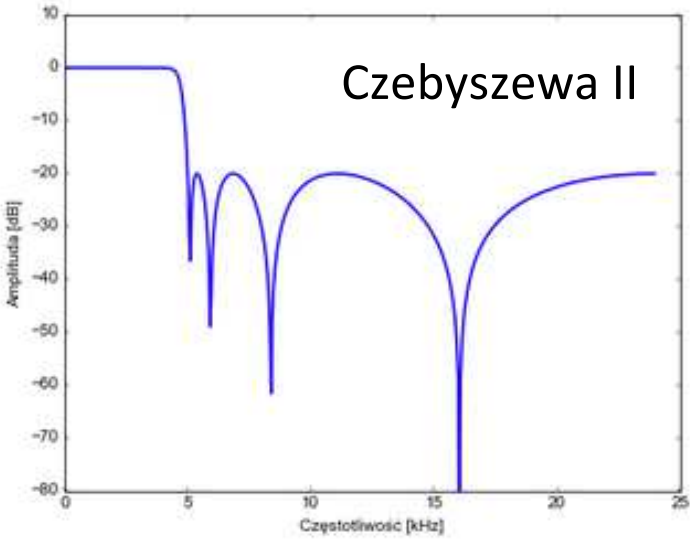
- częstotliwości graniczne
- szerokość pasma przejściowego – rząd filtru
- poziom zafalowań w paśmie:
  - przepustowym
  - zaporowym

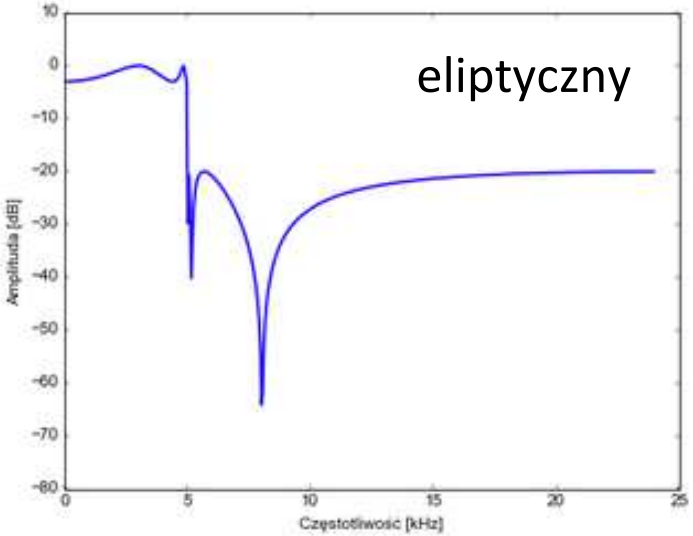


# Typy filtrów IIR



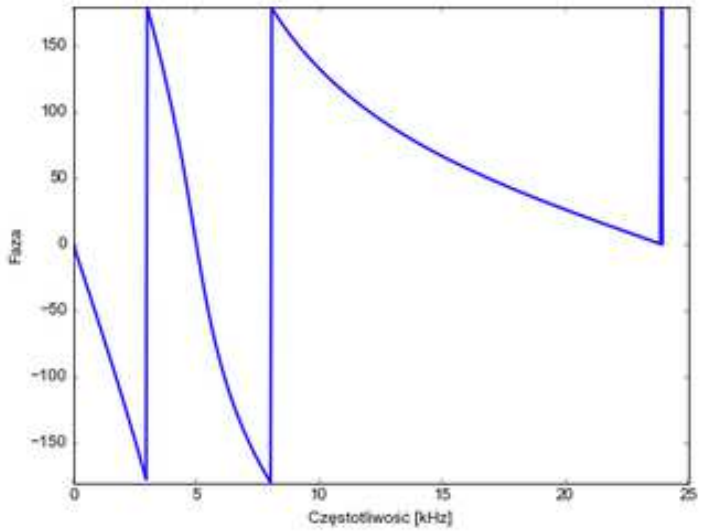



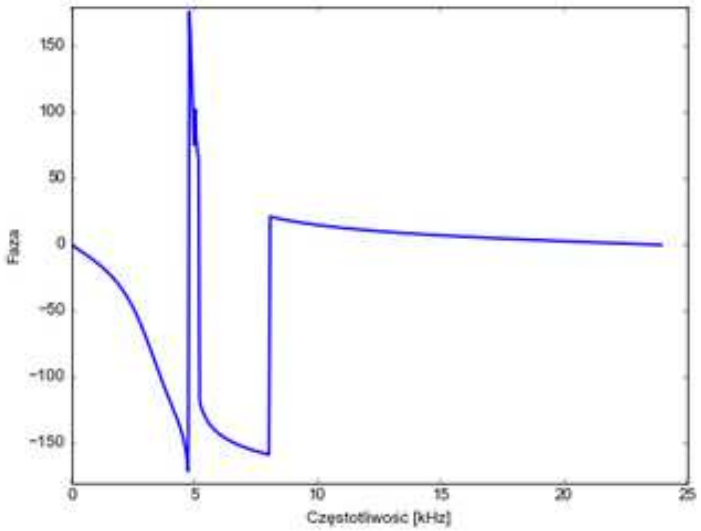
# Nieliniowość fazy

Filtry IIR nie zachowują liniowej charakterystyki fazowej

F. Butterwortha:




F. eliptyczny:



# Kwantyzacja współczynników

- Pierwszy problem. Wynik obliczenia współczynników dla filtru Butterwortha DP 2. rzędu, fc = 1 kHz:

```
0.0039, 0.0078, 0.0039, -1.8153, 0.8310
```

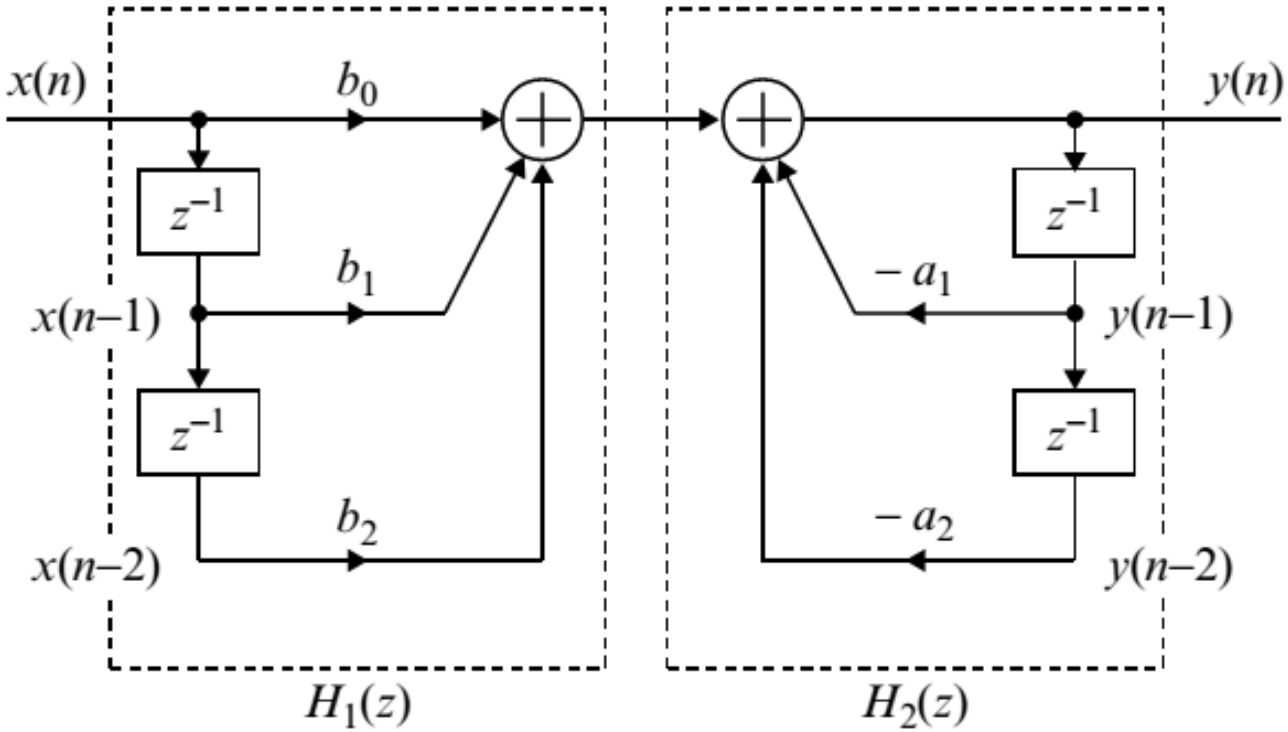
- Jeden ze współczynników zawsze nie zmieści się w zakresie (-1, 1).
- Musimy więc zapisać współczynniki filtru w formacie Q1.14, mnożąc je przez 16384.

```
const int wsp_iir[] = {64, 128, 64, -29743, 13615};
//                      b0,  b1, b2,     a1,    a2
```

# Implementacja (forma I)

- Implementacja w formie I (*direct form I)*
  - bezpośrednio z równania różnicowego

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)$$



# Implementacja (forma I)

- Dla filtru rzędu *N* potrzebujemy bufora na 2*N*+1 próbek: *N*+1 wejściowych i *N* wyjściowych.
- Dla filtrów niskiego rzędu możemy użyć bufora liniowego (wygodniej).
- Należy pamiętać o wypełnieniu bufora zerami przed pierwszą filtracją – inaczej podamy śmieci jako stan początkowy.

```
const int wsp_iir[] = {64, 128, 64, -29743, 13615};
int bufor[5];
```

# Implementacja (forma I)

```
// przesuwanie bufora:
bufor[2] = bufor[1];       // x(n-2) – poprzednia x(n-1)
bufor[1] = bufor[0];       // x(n-1) – poprzednia x(n)
bufor[0] = probka;         // x(n) – nowa próbka wejściowa

// filtracja
long suma = 0;
suma = _smac(suma, wsp_iir[0], bufor[0]);  // b0 * x(n)
suma = _smac(suma, wsp_iir[1], bufor[1]);  // + b1 * x(n-1)
suma = _smac(suma, wsp_iir[2], bufor[2]);  // + b2 * x(n-2)
suma = _smas(suma, wsp_iir[3], bufor[3]);  // - a1 * y(n-1)
suma = _smas(suma, wsp_iir[4], bufor[4]);  // - a2 * y(n-2)

int wynik = (int)((suma + 16384) >> 15);
bufor[4] = bufor[3];       // y(n-2) – poprzednia y(n-1)
bufor[3] = wynik;          // nowe y(n-1)
```

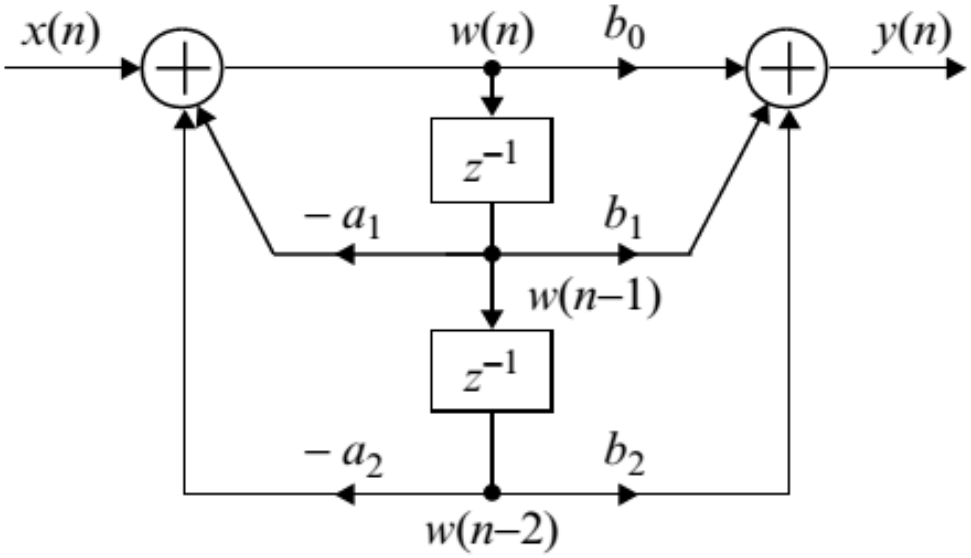
# Implementacja (forma I)

Uwagi do kodu.

- Mnożymy liczbę Q15 (próbka) przez Q1.14 (współczynnik) – wynik Q1.29. Tryb ułamkowy zamienia wynik na Q1.30.
- Potrzebujemy wyniku Q15, więc musimy go przesunąć o 30 – 15 = 15 miejsc. Dodanie 16384 zaokrągla wynik.
- Zamiast robić ręcznie 5 operacji, możemy użyć pętli z instrukcją _*smac*, jeżeli zapiszemy współczynniki *a* z przeciwnym znakiem.

# Implementacja – forma II

Alternatywna forma – *Direct form II*




$$w(n) = x(n) - a_1 w(n-1) - a_2 w(n-2)$$
$$y(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2)$$

# Forma I vs. forma II

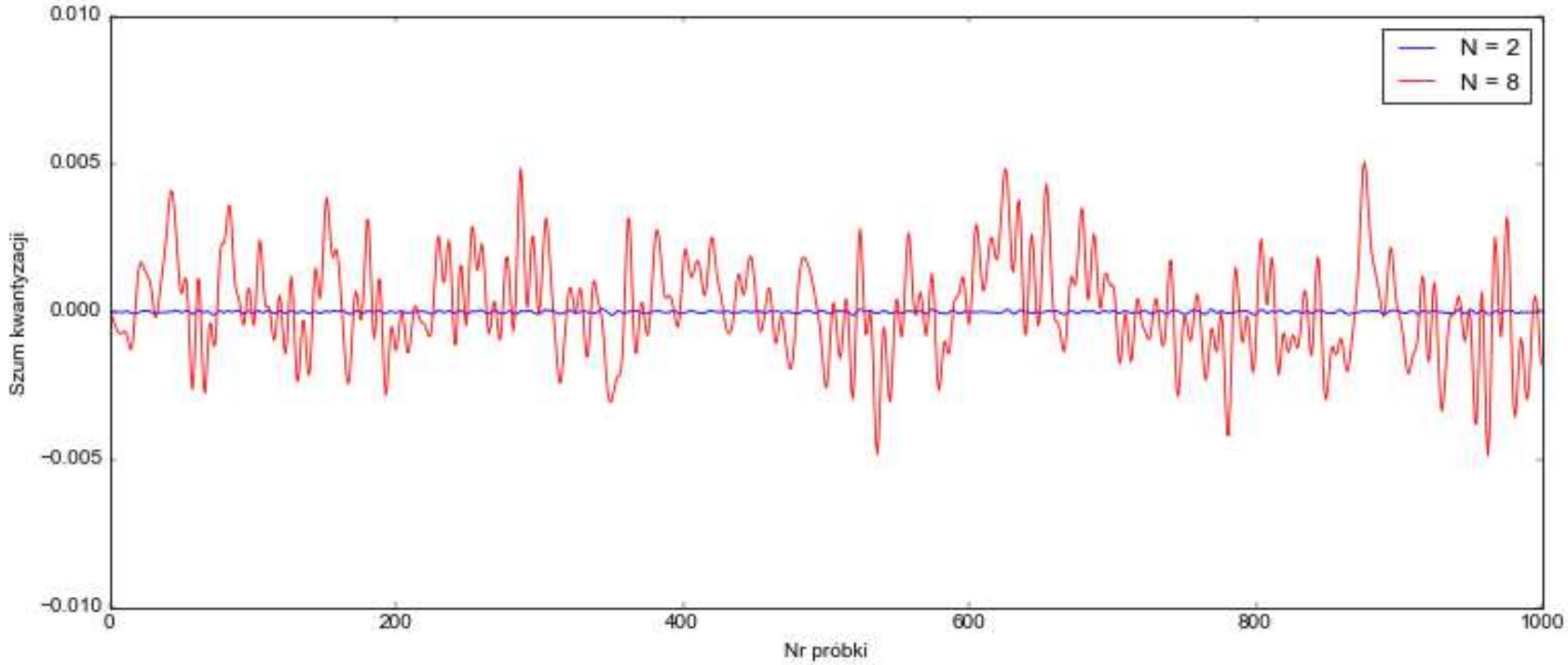
- W implementacji zmiennoprzecinkowej, forma II jest lepsza – wymaga mniejszych buforów.
- W implementacji stałoprzecinkowej mamy problem przepełnienia zakresu w trakcie obliczeń.
- Przepełnienie powoduje zniekształcenie wyniku, co ma wpływ na kolejne próbki (rekursywność!).
- Forma I jest mniej wrażliwa na efekty przepełnienia, jest zatem zalecana dla implementacji stałoprzecinkowej.

# Szum kwantyzacji

- Współczynniki filtru są kwantowane, zatem różnią się one od obliczonych w projekcie.
- Ma to wpływ na charakterystykę filtru.
- Filtr IIR jest rekursywny, zatem błąd kwantyzacji ma udział w obliczaniu kolejnych próbek – efekty kwantyzacji kumulują się.
- Wpływ szumu kwantyzacji w filtrach IIR jest znacznie większy niż w przypadku filtru FIR.

# Szum kwantyzacji

Porównanie szumu kwantyzacji dla filtrów IIR rzędu 2 (niebieski) i 8 (czerwony):



# Stabilność filtru IIR

Inny sposób zapisu transmitancji filtru IIR:

$$H(z) = G\frac{(z-r_0)(z-r_1)\ldots(z-r_N)}{(z-p_1)(z-p_2)\ldots(z-p_N)}$$

- *r*: zera – miejsca zerowe licznika
- *p*: bieguny (*poles*) – miejsca zerowe mianownika
- *G*: wzmocnienie układu

Filtr IIR jest stabilny, jeżeli bieguny nie leżą na zewnątrz okręgu jednostkowego: $|p_k| \leq 1$

# Stabilność filtru IIR

Co w praktyce oznacza stabilność filtru IIR?

- Filtr stabilny: po wyłączeniu sygnału wejściowego, sygnał wyjściowy zeruje się po maks. *N* próbkach.
- Filtr niestabilny: wyłączamy sygnał na wejściu, a filtr nadal generuje próbki na wyjściu.
- Filtr „wzbudza się” (wpada w niekontrolowane oscylacje) i działa nieprawidłowo.
- Typowy filtr IIR musi być stabilny.

# Stabilność filtru IIR

- Oprogramowanie do projektowania filtrów zwykle da nam stabilny filtr (ale należy to sprawdzić!).
- Często filtry projektuje się na granicy stabilności - bieguny na okręgu jednostkowym lub blisko niego.
- Błędy kwantyzacji mogą spowodować, że bieguny wyjdą poza okrąg jednostkowy. Filtr stanie się niestabilny!
- Zawsze należy sprawdzić położenie biegunów po kwantyzacji współczynników!

# Struktura kaskadowa IIR

Problemy implementacji filtrów IIR wyższych rzędów na procesorze stałoprzecinkowym:

- rośnie wpływ szumu kwantyzacji,
- błąd kwantyzacji w jednym miejscu wpływa na działanie całego filtru,
- więcej współczynników rekursywnych – większa kumulacja błędów,
- wzrasta ryzyko przepełnienia zakresu.

# Struktura kaskadowa IIR

Raz jeszcze alternatywny zapis transmitancji:

$$H(z) = G\frac{(z-r_0)(z-r_1)\ldots(z-r_N)}{(z-p_1)(z-p_2)\ldots(z-p_N)}$$

Możemy to podzielić na kilka czynników:

$$H(z) = \prod_k G_k \frac{(z-r_{k0})(z-r_{k1})}{(z-p_{k0})(z-p_{k1})}$$

Struktura kaskadowa – szeregowe połączenie filtrów.

# Struktura kaskadowa IIR

Filtr IIR parzystego rzędu *N* można przedstawić jako kaskadowe połączenie *N*/2 filtrów dwukwadratowych:

$$H(z) = \prod_k g_k \frac{b_{k0} + b_{k1} z^{-1} + b_{k2} z^{-2}}{1 + a_{k0} z^{-1} + a_{k1} z^{-1}}$$

W implementacji stałoprzecinkowej:

- szum kwantyzacji ma wpływ tylko na daną sekcję,
- całkowity błąd kwantyzacji ulega zmniejszeniu,
- maleje ryzyko przepełnienia i niestabilności.

# Struktura kaskadowa IIR

- W implementacji zmiennoprzecinkowej: podział na sekcje jest bez znaczenia – mnożenie jest przemienne.
- W implementacji stałoprzecinkowej to już ma znaczenie. Jedna para zer/biegunów może spowodować przepełnienie, inna nie!
- Zbudowanie optymalnej struktury kaskadowej do implementacji na stałoprzecinkowym DSP nie jest prostym zadaniem.

# Struktura kaskadowa IIR

- Zaczynamy od układu zer, biegunów i wzmocnienia.
- Musimy:
  - podzielić wzmocnienie na sekcje,
  - ułożyć pary zer i pary biegunów,
  - połączyć zera i bieguny ze sobą w sekcje,
- w taki sposób, aby nie występowały przepełnienia.
- Wzmocnienie sygnału w każdej sekcji powinno być tak duże, jak to możliwe, ale nie powodując przepełnienia.

# Przykład projektu

- Filtr DP Butterwortha, rząd 8, fc = 1 kHz
- Dostajemy z projektu:

```
k =  2.43444901944e-10
SOS = [[ 1.   2.   1.   1.   -1.75785265  0.77302109]
       [ 1.   2.   1.   1.   -1.78875835  0.80419348]
       [ 1.   2.   1.   1.   -1.84881984  0.86477323]
       [ 1.   2.   1.   1.   -1.93365048  0.95033587]]
```

- Czasami wzmocnienie *k* jest „wciągane” do pierwszej sekcji – trzeba je „wyciągnąć”.

# Przykład projektu

- Wzmocnienie `2.43444901944e-10` jest małe – jego zastosowanie do jednej sekcji wyzeruje współczynniki.
- Spróbujmy rozłożyć to wzmocnienie na cztery sekcje:
  - $k^{1/4}$ = 0.00395002797191
- mnożymy licznik każdej sekcji przez tę wartość.

```
SOS = [[ 0.00395003  0.00790006  0.00395003  1.  -1.75785265  0.77302109]
       [ 0.00395003  0.00790006  0.00395003  1.   -1.78875835  0.80419348]
       [ 0.00395003  0.00790006  0.00395003  1.   -1.84881984  0.86477323]
       [ 0.00395003  0.00790006  0.00395003  1.   -1.93365048  0.95033587]]
```

# Przykład projektu

- Uzyskany projekt nie zapewnia, że nie wystąpi przepełnienie. Zobaczmy maksymalne wzmocnienia po przejściu przez każdą sekcję:

```
[1.04164374448  1.06627496682  1.0560301017  1.00000001587]
```

- Chcemy aby wzmocnienie nie przekraczało 1.
- Przedstawimy jedno z możliwych rozwiązań.
- Obliczenie maks. wzmocnienia dla sekcji SOS: maksimum charakterystyki amplitudowej.

# Przykład projektu

- Bierzemy pierwszą sekcję i obliczamy maksymalne wzmocnienie charakterystyki amplitudowej.
- Dzielimy współczynniki *b* tej sekcji przez obliczone wzmocnienie.
- Bierzemy pierwsze dwie sekcje (uwzględniając modyfikacje), obliczamy maks. wzmocnienie.
- Dzielimy wsp. *b* sekcji drugiej przez wynik.
- Powtarzamy dla trzeciej i czwartej sekcji.
- Przykładowy kod – w materiałach pomocniczych.

# Filtr IIR w DSPLIB

Spośród implementacji IIR w Asemblerze z biblioteki DSPLIB, najlepiej nadaje się funkcja *iircas51*.

*Cascaded IIR Direct Form I (5 Coefficients per Biquad)*

```
ushort oflag = iircas51 (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nbiq,
ushort nx)
```

- *x* – wskaźnik do próbki wejściowej (*&probka*) lub do bufora próbek do przetworzenia (*bufor_wej*)
- *h* – wskaźnik do współczynników filtru, w kolejności: *b0, b1, b2, a1, a2*; sekcja po sekcji (bez *a0*!) *(DATA*)wsp_iir*

# Filtr IIR w DSPLIB

```
ushort oflag = iircas51 (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nbiq,
ushort nx)
```

- *r* – wskaźnik do próbki wyjściowej (*&wynik*) lub do bufora próbek wyjściowych (*bufor_wyj*)
- *dbuffer* – bufor o długości (4 * liczba sekcji + 1), który musimy sami utworzyć,
- *nbiq* – liczba sekcji dwukwadratowych,
- *nx* – liczba próbek do przetworzenia.
- Funkcja zwraca wartość *oflag*, która jest równa 1 jeżeli wystąpiło przepełnienie podczas obliczeń.

# Niedopełnienie

- Niedopełnienie (*underflow*) występuje wtedy, gdy wynik operacji jest zbyt mały, aby mógł być zapisany na najmniej znaczącym bicie.
- Dla 32 bitów (Q31) występuje, gdy wynik < $2^{-31}$
- Najczęściej jest spowodowany mnożeniem przez małe liczby.
- Przed niedopełnieniem można się chronić poprzez odpowiednie skalowanie danych oraz odpowiednią kolejność działań.

# Filtr IIR w DSPLIB (c.d.)

- Implementacja *iircas51* zakłada, że próbki i współczynniki są zapisane jako Q15.
- Ale nasze współczynniki są Q1.14!
- Wynik filtracji *y*(*n*) (Q1.14) jest traktowany jako Q15. Jest więc dwukrotnie za mały!
- W rezultacie:
  - filtr przetwarza błędne wyniki *y*(*n*-…),
  - może wystąpić niedopełnienie na dalszych sekcjach – zerowy wynik filtracji.

# Filtr IIR w DSPLIB (c.d.)

- Aby filtr działał prawidłowo dla współczynników Q1.14, trzeba zmodyfikować kod asemblerowy.
- Odpowiedni plik (*iircas51.asm*) należy skopiować do katalogu projektu i dodać mnożenie przez 2.
- Pliki do projektu ZPS będą już zmodyfikowane.

```
238         || RPTBLOCAL OuterLoopEnd-1                 ;outer loop: process a new input
239                 MOV     *AR0+ << #16, AC1           ; HI(AC1) = x(n)
240                 ||RPTBLOCAL     InnerLoopEnd-1      ;inner loop: process a bi-quad
241                     NOP_16                          ; CPU_116: Remark 5682
242                 || MPYM     *AR1+, AC1, AC0         ; AC0 = b0*x(n)
243                     MACM    *AR1+, *(AR6+T0), AC0   ; AC0 += b1*x(n-1)
244                     MACM    *AR1+, *AR6, AC0        ; AC0 += b2*x(n-2)
245                     MOV     HI(AC1), *(AR6+T1)      ; x(n) replaces x(n-2)
246                     MASM    *AR1+, *(AR4+T0), AC0   ; AC0 -= a0*y(n-1)
247                     MASM    *AR1+, *AR4, AC0, AC1   ; AC1 -= a1*y(n-2)
248                     SFTS    AC1, #1                 ; AC1 *= 2 (correction for Q14)
249                     MOV     rnd(HI(AC1)), *(AR4+T1) ; y(n) replaces y(n-2)
250     InnerLoopEnd:
251                 AMOV    AR7, AR1                    ;reinitialize coeff pointer
252             || MOV      rnd(HI(AC1)), *AR2+         ;store result to output buffer
```

# Filtr IIR w DSPLIB (c.d.)

Przykład wywołania funkcji *iircas51*

```
const int wsp_iir[] = {
     62,    124,     62,  -28801,  12665
     63,    126,     63,  -29307,  13176
     65,    131,     65,  -30291,  14168
     68,    137,     68,  -31681,  15570};
int bufor[17];
```

```
// int probka – wejściowa próbka do filtracji
// int wynik  – wyjściowa próbka po filtracji

iircas51(&probka, (DATA*)wsp_iir, &wynik, bufor, 4, 1);
```

# Podsumowanie – implementacja IIR

- Ustalić parametry projektowe – częstotliwość graniczną, rząd filtru, typ charakterystyki.
- Obliczyć współczynniki za pomocą oprogramowania.
- Podzielić zera i bieguny na sekcje dwukwadratowe, tak aby zminimalizować błędy kwantyzacji oraz ryzyko przepełnienia i niestabilności.
- Unormować maksymalne wzmocnienie każdej sekcji.
- Sprawdzić stabilność i maksymalne wzmocnienie.
- Zaimplementować za pomocą np. *iircas51*.

# Podsumowanie – filtry IIR

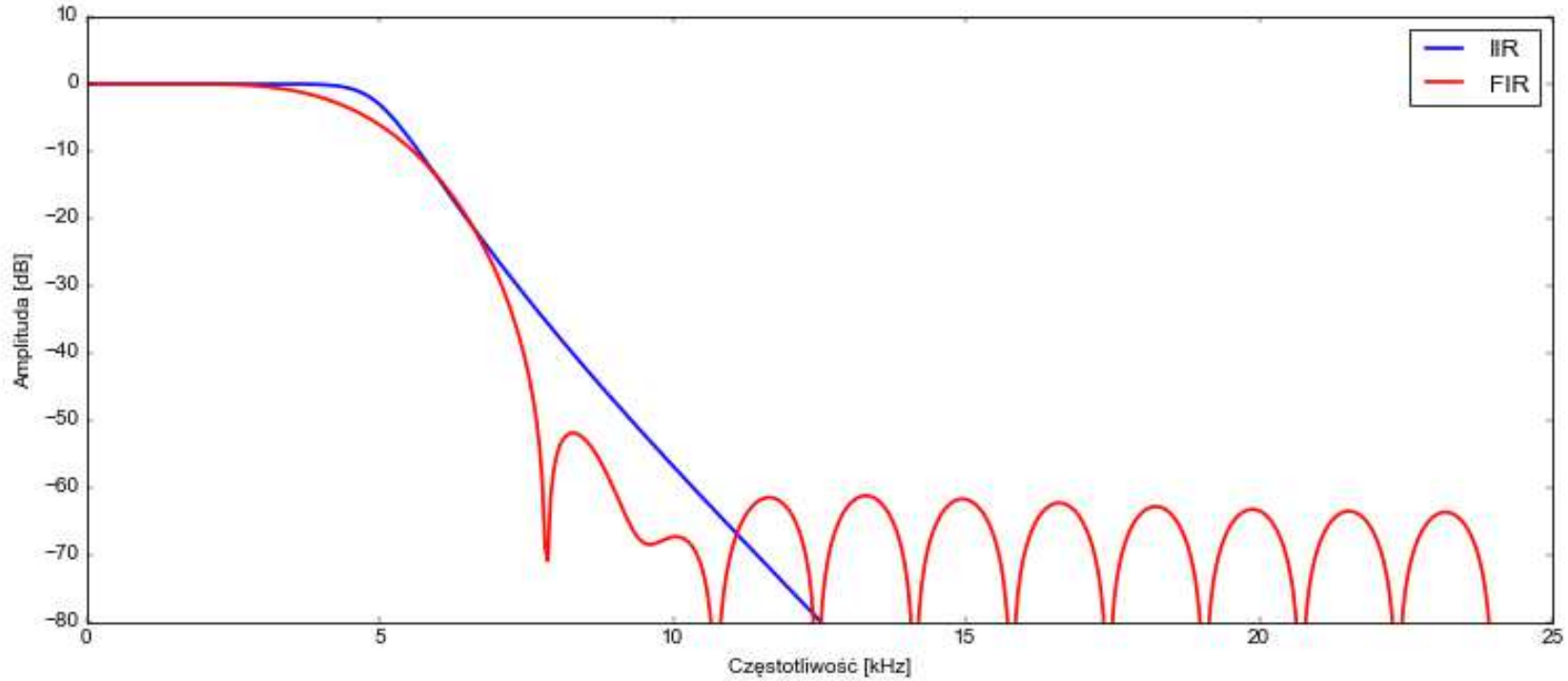
Cechy filtrów IIR:

- (–) zniekształcają fazę,
- (–) mogą stać się niestabilne,
- (–) większy wpływ szumu kwantyzacji na wyniki,
- (–) w implementacji stałoprzecinkowej: duża podatność na przepełnienie i niedopełnienie,
- (–) dużo trudniejsze projektowanie,
- (+) prosta implementacja,

# Podsumowanie – filtry IIR

- (+) większa skuteczność filtracji niż FIR – mniejsza liczba obliczeń dla tego samego efektu.

Porównanie: IIR rzędu 8 vs. FIR rzędu 30



# Podsumowanie – filtry IIR

Kiedy więc warto stosować IIR zamiast FIR?

- gdy musimy oszczędzać cykle procesora
  - IIR zużyje ich mniej,
- gdy wymóg liniowości fazy nie jest krytyczny,
- gdy potrafimy sobie poradzić z projektem IIR ☺

W większości przypadków, dopóki mamy wystarczająco dużo cykli DSP, filtry FIR są łatwiejsze w projektowaniu i w implementacji („lepiej się zachowują”).

# Podsumowanie – filtry IIR

Co musimy umieć i o czym pamiętać?

- Różnice między IIR i FIR.
- Jak zaimplementować filtr IIR w C.
- Dlaczego szum kwantyzacji jest ważniejszy w IIR.
- Jak wygląda struktura kaskadowa i dlaczego należy z niej korzystać na DSP.
- Jak chronić się przed przepełnieniem modyfikując wzmocnienie sekcji dwukwadratowych.
- Jak uruchomić filtr IIR za pomocą DSPLIB.