Temat: Obliczanie odpowiedzi systemów cyfrowych na zadane wymuszenie numeryczne z wizualizacją

Imię i nazwisko: Łukasz Hajec, Kacper Szepielak.

1. Wstęp

Technika cyfrowa zapewnia dostęp do technologii, która była dana tylko wybranym oraz pozwala na wykonywanie coraz to bardziej pracochłonnych i skomplikowanych operacji we własnym zaciszu domowym.

Naszym zadaniem było obliczenie odpowiedzi systemów cyfrowych na zadane wymuszenie numeryczne wraz z wizualizacją. Oznacza to, że musieliśmy zasymulować działanie filtru cyfrowego o danej odpowiedzi impulsowej. Ale co to jest filtracja cyfrowa? W skrócie jest to proces przetwarzania dokonywany na spróbowanym sygnale dyskretnym w celu uzyskania odpowiedniego sygnału wyjściowego również w postaci dyskretnej.

Transmitancja filtru może być skończona(filtr o skończonej odpowiedzi impulsowej-SOI) lub nieskończona (filtr o nieskończonej odpowiedzi impulsowej-NOI).

Transmitancje w filtrze SOI wyraża się następująco:

$$H(z) = \sum_{k=1}^{N} b_k z^{-k-1}$$

N- rząd filtru (ile próbek wstecz),
b_k- współczynniki filtru SOI (N+1 współczynników).

Odpowiedź systemu na sygnał wejściowy otrzymamy poprzez zaimplementowanie równania różnicowego:

$$y(n) = b_1 x(n) + b_2 x(n-1) + b_3 x(n-2) + \ldots + b_n x(n-N)$$

W filtrze NOI transmitancje zapiszemy następująco:

$$H(z) = \frac{(\sum_{k=1}^{N} b_k z^{-k-1})}{(1 + \sum_{k=2}^{N} a_k z^{-N})}$$

Natomiast równanie różnicowe:

$$y(n) = b_1 x(n) + b_2 x(n-1) + \ldots + b_N x(n-N) - a_2 y(n-1) - a_3 x(n-2) - \ldots - a_N y(n-N)$$

N- rząd filtru,
a,b- współczynnik filtru NOI.

Dla NOI występuje tak zwane sprzężenie zwrotne układu to znaczy poprzednie próbki wpływają na dalsze wyniki impulsu wyjściowego.

Celem naszej pracy było odpowiednie zaimplementowanie powyższych wzorów w środowisku MATLAB, aby można było obserwować wizualizację odpowiedzi systemu na zadane wymuszenie. Program ma działać następująco:

- wygodnie wprowadzamy transmitancję, jako wektory współczynników,
- rozpoznaje czy użytkownik sprawdza działanie filtru SOI czy NOI,
- pyta o odpowiednie warunki początkowe, jeśli użytkownik nie chce podawać takowych, program domyślnie przyjmuje zera,

- dzięki implementacji równań różnicowych program oblicza odpowiedź układu na wymuszenie i przedstawia jego wizualizację na odpowiednim wykresie.

2. Implementacja.

Główną część naszego projektu realizuje m-funkcja „odpowiedz2.m", która korzysta z dwóch funkcji wewnętrznych, tj. „odp_skonczona", odp_nieskonczona", w których realizowana jest implementacja równań różnicowych oraz wizualizacja odpowiedzi systemu.

```
2   function y = odpowiedz2(b0,x,a0)
3   % do funkcji mozesz wprowadzic lub (b0,x) dla SOI lub (b0,x,a0) dla NOI  %...%
8
9   switch nargin
10      case 0
11          error("Nie podałeś potrzebnych danych.")
12      case 1
13          error("Podałeś za mało potrzebnych danych.")
14      case 2
```

Do funkcji możemy wprowadzić współczynniki 'b' i wektor wymuszeń lub współczynniki 'b', wektor wymuszeń i współczynniki 'a', program dzięki funkcji *nargin*, która zwraca liczbę argumentów wprowadzonych do funkcji, oraz instrukcji warunkowej *switch* może realizować kolejne instrukcje dla obu rodzajów transmitancji.

Zdecydowaliśmy, że dopiero później program pyta użytkownika o warunki początkowe, które można wprowadzić korzystając z funkcji *input*. Następnie program, z odpowiednimi parametrami wejściowymi, wywołuje funkcje *odp_skonczona* lub *odp_nieskonczona,* które korzystając z *nargin*: realizują odpowiedź systemu na wymuszenie przyjmując zadane warunki początkowe lub przyjmując domyślne warunki początkowe – zera.

Implementacja równania różnicowego dla filtru SOI:

```
60  for i=1:(length(x)+(M-1)) %główna pętla korzystajaca z równanania różnicowego
61              y(i) = 0;
62
63              for k=1:(M-1)
64                  bufor(k) = bufor(k+1);
65              end
66
67              if i >= length(x)+1
68              bufor(M) = 0;
69              else
70              bufor(M) = x(i);
71              end
72
73              for j=1:M
74                  h =  b(j)*bufor(M+1-j);
75                  y(i) = y(i) +h;
76              end
77  end
```

Zmienna *bufor*, została przygotowana wcześniej, by odpowiednio wykorzystywać warunki początkowe, pozwala także wygodnie używać kolejnych elementów wektora wymuszeń. Instrukcja *if* (linia 67) jest potrzebna, aby uzyskać warunki końcowe (po wygaśnięciu wymuszeń) dla odpowiedzi filtru, ponieważ podając X-elementowy wektor wymuszeń na filtr SOI powinniśmy uzyskać X+M-1 elementów wektora odpowiedzi - linia 60 (gdzie M to ilość współczynników 'b' filtru, a zarazem ilość dostępnych buforów).

Implementacja równania różnicowego dla filtru NOI:

```
119 -       for i =1:length(x)      %petla do czesci ze wspolczynnikami b transmitan
133 -       for i =1:length(x)      %petla do czesci ze wspolczynnikami a transmitan
134 -           y_(i) = 0;
135
136 -           for k=1:(N-1)
137 -               bufor_(k) = bufor_(k+1);
138 -           end
139
140 -           if i == 1
141 -               bufor_(N) = bufor_(N);
142 -           else
143 -               bufor_(N) = y(i-1);
144 -           end
145
146 -           for j=2:(N+1)
147 -               roznica =  a(j)*bufor_(N+2-j);
148 -               y__(i) = y__(i) +roznica;
149 -           end
150
151 -           y(i) = y_(i) - y__(i);
152 -       end
```

Część równania zawierająca współczynniki ‘b’ została zaimplementowana analogicznie jak dla filtru SOI. Zmienna *bufor_* o szerokości N (liczba współczynników ‘a’) pozwala realizować wcześniej wspomniane sprzężenie zwrotne. Dla filtru NOI liczba elementów wektora odpowiedzi systemu jest równa liczbie elementów wektora wymuszeń, stąd ilość iteracji głównej pętli to *length(x)*. W linii 151 tworzymy ostateczną postać kolejnych elementów wyjściowych odejmując od część obliczonej wcześniej, dla współczynników ‘b’ - *y_* , część ze współczynnikami ‘a’ - *y__*.

Dla obu przypadków realizujemy wizualizacje odpowiedzi danego systemu cyfrowego na wymuszenie korzystając z funkcji *stem*. Na czerwono powinniśmy zobaczyć wymuszenia, a kolor czarny będzie reprezentował odpowiedź filtru.

```
figure(2)                %wizualizacja
hold on; grid on;
stem(x,'ro','fill')
stem(y,'kd','fill')
legend('wymuszenie numeryczne','odpowiedź na wymuszenie');
```

3. Wyniki, wnioski.

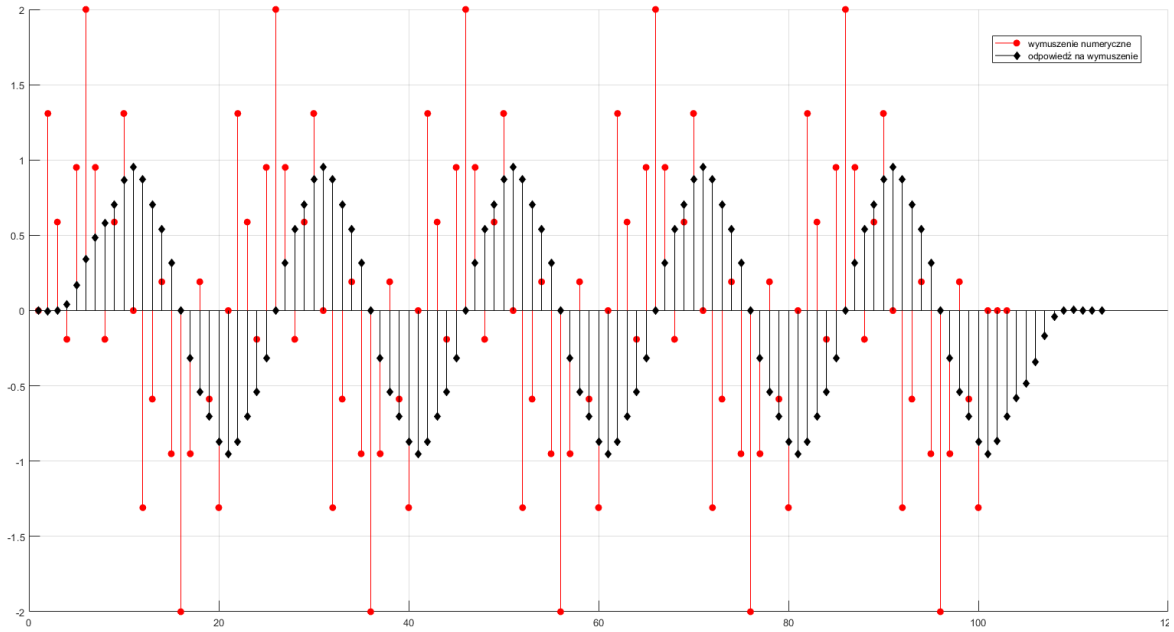
Plik „odpowiedz_wywolanie.m” stanowi niejako podsumowanie naszego projektu, pozwala na wybranie przykładowej symulacji działania funkcji „odpowiedz2.m” dla filtru SOI lub NOI, przy odpowiednio dobranych parametrach transmitancji i wektora wymuszeń. Program działa prawidłowo, tzn. :

- po wywołaniu funkcji z odpowiednimi parametrami, ustala jaki filtr zamierza sprawdzić użytkownik oraz pyta o odpowiednie warunki początkowe,

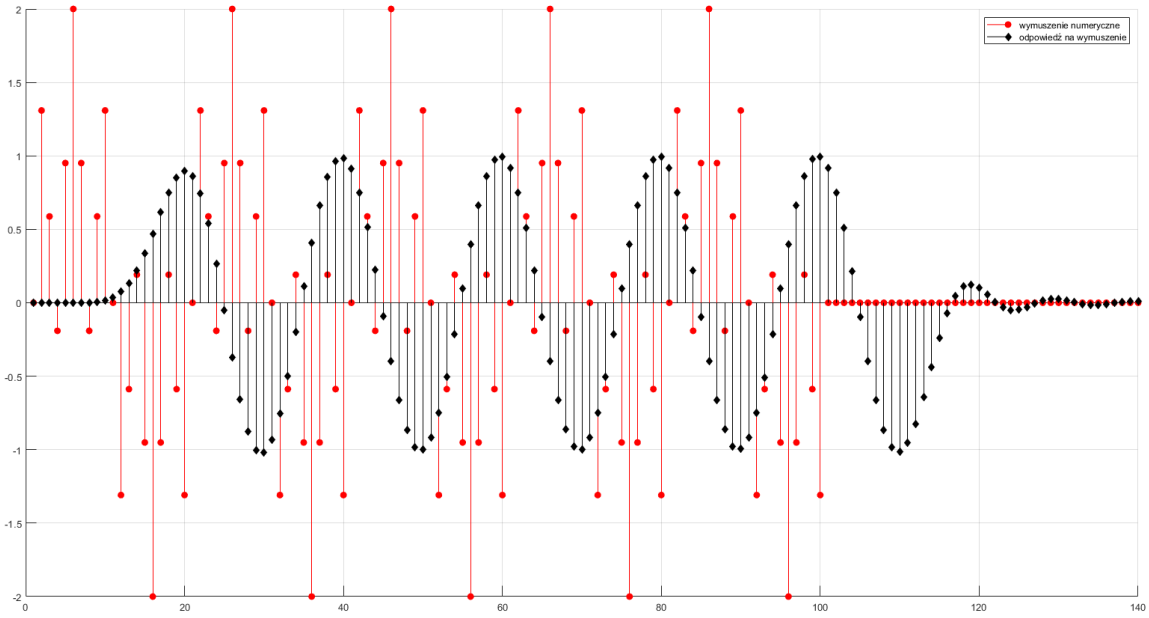
```
x = [0 1.30901699437495 0.58778
a = [1 -8.18492991535752 31.903
b = [1.27777060737744e-08 1.788
y2 = odpowiedz2(b,x,a);

Podaj warunki poczatkowe b w formacie "[x(n-1) x(n-2) ...]" ones(1,14)

wb =

     1     1     1     1     1     1     1     1     1     1     1     1     1     1

Podaj warunki poczatkowe a w formacie "[y(n-1) y(n-2) ...]" 2*ones(1,14)

wa =

     2     2     2     2     2     2     2     2     2     2     2     2     2     2
```

- następnie po rozwiązaniu równań różnicowych i stworzeniu wektora odpowiedzi systemu, odpowiednio go wizualizuje:

Wynik działania programu dla SOI, warunki początkowe domyślne.



Wynik działania programu dla NOI, warunki początkowe domyślne.



Dzięki naszemu projektowi możemy wygodnie, szybko znaleźć i obserwować odpowiedź układu cyfrowego o zadanej transmitancji na wymuszenie numeryczne, co byłoby niezmiernie trudne bez użycia komputera.

Otrzymane wyniki zgadzają się z teoretycznymi założeniami, tzn. dla filtru SOI, po wygaśnięciu wymuszeń możemy jeszcze obserwować warunki końcowe (10 próbek wyjściowych, ponieważ wektor współczynników ‘b’ miał 11 elementów); dzięki kilku zerowym wartością elementów x na końcu widzimy, że odpowiedź filtru SOI szybko wygasa. Natomiast dla filtru NOI odpowiedź układu, nawet dla kilkudziesięciu elementów x równych 0 na końcu wektora wymuszeń, oscyluje wokół zera - teoretycznie nigdy nie powinna go osiągnąć, jednak ze względu na dokładność obliczeń i bardzo małe wartości tej odpowiedzi po jakimś czasie ciężko to obserwować.

Działanie naszego programu zweryfikowaliśmy przy użyciu funkcji *conv* oraz *filter*, co jest ostatnią częścią pliku „odpowiedz_wywołanie”. Przebiegi nakładają się, dlatego możemy ostatecznie stwierdzić, że zrealizowaliśmy odpowiednio założenia naszego projektu.