Politechnika Wrocławska

Przetwarzanie sygnałów IV

Autor: mgr. inż. Piotr Lampa

Wstęp

Projekt polega na stworzeniu programu, który będzie generował sygnał (funkcję matematyczną), generował losowy szum, zapisywał i odczytywał sygnały z pliku oraz filtrował szum z sygnału.

Poniżej znajdują się zadania projektu oraz omówienie zagadnień koniecznych do ich zrealizowania. Cały projekt powinien zostać skończony do następnych zajęć.

Projekt: Przetwarzanie sygnałów – część IV

Napisz program, który zawiera:

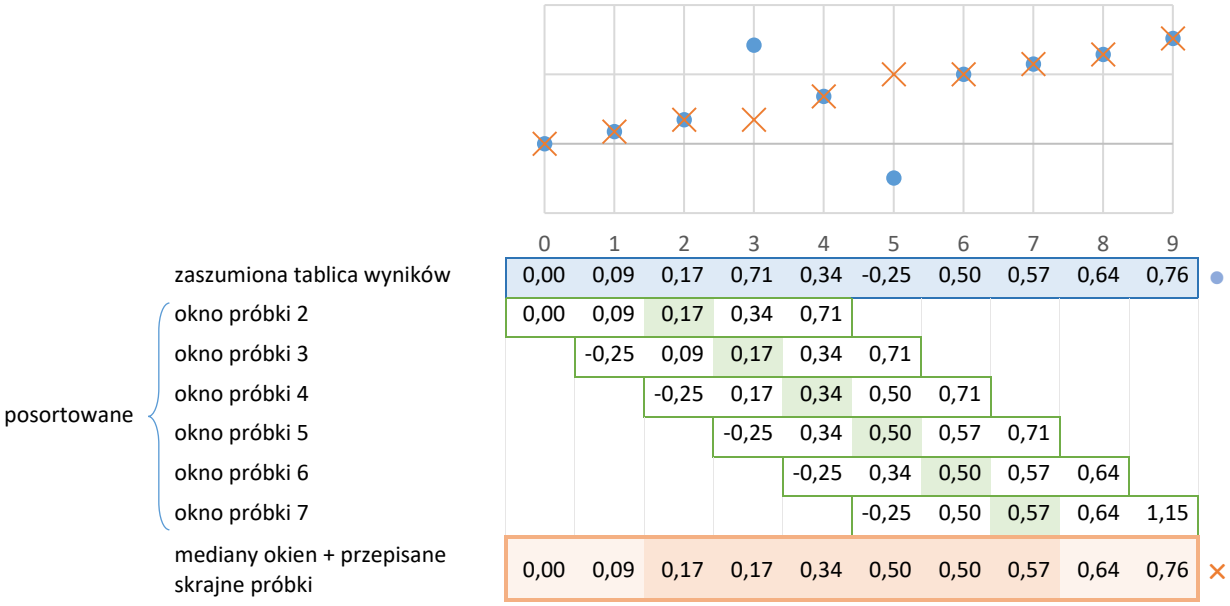
1. Funkcję generującą przebieg funkcji matematycznej (będzie podany dla każdego indywidualnie). Jako parametry wejściowe funkcji wchodzą: tablica współczynników funkcji, dwa parametry określające dziedzinę, tablica wynikowa oraz wielkość tablicy wynikowej.
   DLA AMBITNYCH: wygeneruj funkcję piłokształtną lub trójkątną;
2. Funkcję wczytującą z konsoli współczynniki funkcji (jako tablicę);
3. Funkcję wczytującą zakres dziedziny;
4. Funkcję generującą losowy szum (zakłócenia) na sygnale i zapisującą zaszumiony sygnał w osobnej tablicy. Szum powinien być losowy w kilku dziedzinach, na przykład: zaszumienie z określonym prawdopodobieństwem, amplituda szumu wylosowana z określonych granic itp.
5. Funkcję zapisującą tablicę (podstawowa, zaszumioną, a później również odfiltrowaną) z wartościami funkcji do pliku .CSV z możliwością otwarcia w arkuszu kalkulacyjnym;
6. Funkcję wczytującą rozmiar tablicy wynikowej;
7. Dynamiczną alokację pamięci dla tablicy wynikowej za pomocą funkcji **`malloc()`** lub **`calloc()`**.
8. Funkcję odczytującą dane (sygnał) z pliku .CSV.
   DLA AMBITNYCH: z użyciem funkcji **`realloc()`** – bez sprawdzania długości pliku;
9. Funkcję filtrującą wygenerowany lub wczytany sygnał z szumu (filtr medianowy i średniej ruchomej).
   DLA AMBITNYCH: okno filtra o dowolnej szerokości;
10. Zabezpieczenia programu przed błędnym wprowadzaniem danych;
11. Menu użytkownika – możliwość generowania, wczytania, zaszumienia i filtracji sygnału, zapisania sygnału w każdym z tych stanów itd.

Algorytmy filtracji sygnału

W ramach zajęć do zaimplementowania są dwa proste filtry odszumiające sygnał z losowych zakłóceń: filtr medianowy i filtr średniej ruchomej.

Filtr medianowy

Polega na przeskanowaniu wszystkich próbek z tablicy wyników filtrem - oknem o szerokości będącej nieparzystą liczbą $2n+1$ (załóżmy szerokość 5 elementów). Dla każdej próbki do okna kopiowana jest dana próbka oraz dwie poprzednie i dwie następne (w sumie 5). Następnie w oknie określana jest mediana - próbki w oknie są sortowane[1] w kolejności od najmniejszej do największej i wybierana jest środkowa próbka. Zostaje ona umieszczona w **nowej** tablicy na tej samej pozycji, z której była pobrana. Graficznie działanie filtru przedstawiono na rysunku 3. Efektem sortowania w oknach próbki znacznie odstające od reszty nie są w ogóle brane pod uwagę.

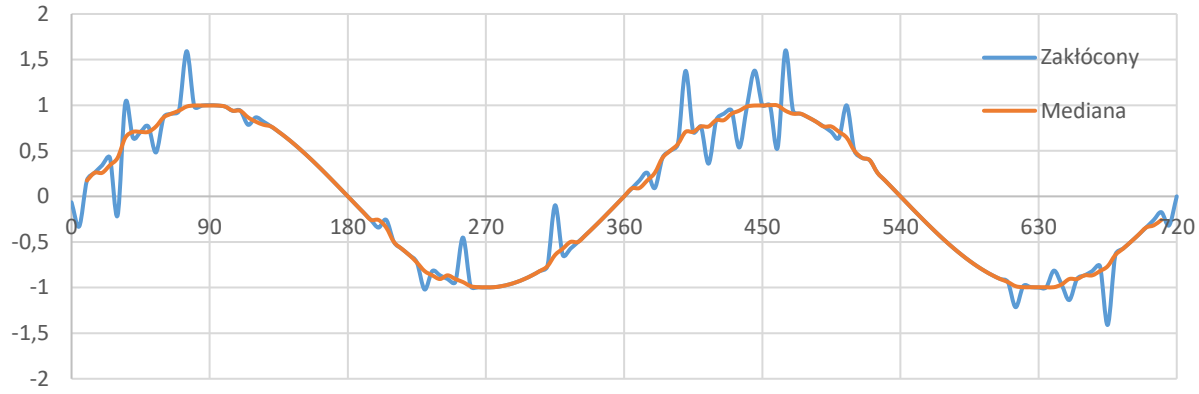


Rys. 3. Idea działania filtra medianowego

- Dlaczego odfiltrowane próbki muszą być wpisywane do nowej tablicy?
- Jakie wady niesie ze sobą stosowanie filtra medianowego?

Efekt działania filtra na funkcję sin(*x*) z losowymi zakłóceniami przedstawia rysunek 4.

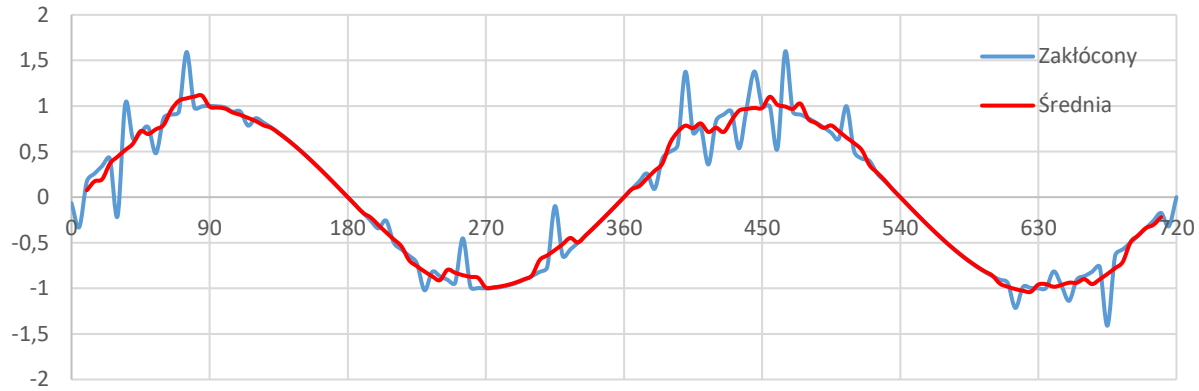
[1] Sposób sortowania jest dowolny, jednak ze względu na prostotę i niewielki zbiór wartości polecam sortowanie bąbelkowe: https://pl.wikipedia.org/wiki/Sortowanie_b%C4%85belkowe



Rys. 4. Efekt działania filtra medianowego

Filtr średniej ruchomej

Zasada działania podobna jak w filtrze medianowym, z okna $2n+1$ przesuwającego się po tablicy próbek wyliczane są wartości średnie, które następnie wpisywane są do nowej tablicy. Efekt działania filtra średniej ruchomej na zakłóconej funkcji sin(x) na poniższym rysunku:



Rys. 5. Efekt działania filtra średniej ruchomej

- Jaka jest różnica w efekcie działania obu filtrów?
- Jaki efekt przyniesie wielokrotne filtrowanie tego samego sygnału?
- Jaki wpływ na efekt ma szerokość okna?
- Co należy zrobić z próbkami na krawędziach sygnału?

Zabezpieczenie przed błędami – „idiotoodporność”

Wiele działań, które można przeprowadzić w programie, może być powodem błędów niewykrywalnych w momencie kompilacji. Najczęstszy przykład to dzielenie przez liczbę podaną przez użytkownika. Jeśli użytkownik wprowadzi wartość 0, doprowadzi to do błędu w programie. Nie można pozwolić, by program z błędnie wprowadzonymi danymi się wyłączał. By się przed tym zabezpieczyć, należy wprowadzić mechanizm obsługi błędów. Ważne, by w programie nie wystąpił „crash” oraz aby wszystkie możliwe zadania zostały sfinalizowane (pliki zamknięte, zwolniona pamięć) a użytkownik został poinformowany, że coś poszło źle. Często w miejscu, gdzie użytkownik podaje wartość, wystarczy zastosować

zabezpieczenie, które przy błędnie wprowadzonej wartości nada komunikat i poprosi o wprowadzenie jej ponownie. Poniżej przykład pętli, która do skutku prosi o podanie liczby różnej od 0.

```
int dzielnik = 0;
while( !dzielnik ) {                //równoznaczne z while( dzielnik == 0 )
       printf( "Podaj dzielnik (rozny od 0): " );
       scanf( "%d", &dzielnik );
       ...
}
```

Listing 1. Pętla pobierająca wartość różną od 0

Zabezpieczenie należy stosować również przy otwieraniu pliku. Jeśli otwierany/tworzony plik jest otwarty w innym programie, albo plik otwierany do odczytu nie istnieje, to dalsze operacje spowodują błąd. Funkcja **`fopen()`** zwraca jednak **`NULL`**, gdy otwieranie pliku zakończy się niepowodzeniem. Jak więc zabezpieczyć program przed błędem otwarcia pliku? Wprowadź takie zabezpieczenie do projektu.

Menu użytkownika

Być może do tej pory tworzyłeś swój program liniowo, czyli tak, by po kolei realizował zaprojektowane funkcje według z góry założonego schematu. Większość programów nie działa w taki sposób – pozwalają użytkownikowi wybrać co chce robić w danym momencie, np. wczytać, zapisać, znów coś wczytać itd. W programach pisanych w konsoli jedną z możliwości realizacji nieliniowego działania jest wprowadzenie menu użytkownika.

W ramach kursu proponuję stworzenie menu opartego o znaną instrukcje **`switch()`**, poprzedzoną wyświetleniem opcji i zapytaniem o wybór. Przykład prostego menu:

```
int wybor = 0;
printf("1.Nowy\n2.Zapisz\n0.Zakoncz");          //wyswietlenie opcji
scanf("%d", &wybor);                            //wybor opcji
switch (wybor)
{
case 1:
       funkcjaNowy(...);                        //realizacja opcji 1
       break;
case 2:
       ...                                      //realizacja opcji 2
...
}
```

Listing 2. Przykład menu użytkownika

Aby móc korzystać z menu wielokrotnie w trakcie działania programu, wystarczy umieścić je w pętli z odpowiednim warunkiem. Pamiętaj, by zabezpieczyć wybór opcji przed beztroskimi użytkownikami!

Przy budowaniu konsolowego programu z menu, przydatna może być możliwość czyszczenia ekranu, do czego w systemie Windows można użyć instrukcji **`system("cls")`**, zaś w Linux **`system("clear")`**.

Debugowanie kodu

Finalizując projekt na pewno nie raz napotkasz na różne problemy, których rozwiązanie będzie wymagało „podglądu” programu w trakcie działania, czyli obserwacji jakie wartości przyjmują zmienne itp.

Możliwe, że do tej pory robiłeś to za pomocą **`printf()`**, wyświetlając w konsoli dane w celu skontrolowania. Czasami jest to dobre podejście, zwłaszcza przy programowania mikrokontrolerów, gdzie może nie być żadnych innych narzędzi.

Visual Studio oraz większość środowisk posiada jednak narzędzie zwane debuggerem, które pozwala na wykonywanie programu krok po kroku z jednoczesnym podglądem wszystkich zmiennych obecnych w aktualnie działającym bloku kodu.

Debugowania programu najłatwiej nauczyć się przez praktykę. Podstawy dobrze omówiono w videotutorialu „Korzystanie z debuggera” na http://www.student.mvlab.pl/c-tutoriale.html .

Bardziej zaawansowane możliwości opisano na http://www.student.mvlab.pl/wiedza/post91.html .

Zagadnienia na następne zajęcia

- Plik graficzny w formacie .PGM.
- Odczyt pliku pod nazwą podaną przez użytkownika.
- Wspólne przechowywanie danych różnych typów – struktury.

Literatura

Prata S (2006) Język C. Szkoła Programowania. Gliwice. Wydawnictwo Helion.

https://pl.wikibooks.org/wiki/C - Wikibooks, Kurs programowania w języku C.

http://www.cplusplus.com/reference/ - Standard C++ Library reference.