

# Laboratorium Cyfrowego Przetwarzania Sygnałów

## *Preskrypt do v.2016Z* *ćwiczenie nr 0 z 10 (Matlab i Labview)*

*na prawach rękopisu*

Jacek Misiurewicz      Krzysztof Kulpa      Piotr Samczyński  
Mateusz Malanowski      Piotr Krysik      Łukasz Maślikowski  
Damian Gromek      Artur Gromek  
Marcin K. Bączyk

**Zakład Teorii Obwodów i Sygnałów**  
**Instytut Systemów Elektronicznych**  
**Wydział Elektroniki i Technik Informatycznych**  
**Politechnika Warszawska**

Uwagi redakcyjne prosimy zgłaszać do: [jmisiure@elka.pw.edu.pl](mailto:jmisiure@elka.pw.edu.pl) tel. 5441

Warszawa, 26 października 2016, 10:05

# Spis treści

<b>0</b>	<b>Laboratorium zerowe – Matlab, LabView i narzędzia</b>	<b>3</b>
0.1	Wprowadzenie do zajęć . . . . .	3
0.2	Wprowadzenie do pakietu MATLAB® . . . . .	3
0.2.1	Matlab – rozpoznanie walką . . . . .	4
0.3	Wprowadzenie do pakietu LABVIEW . . . . .	11
0.3.1	Przyrząd złożony z gotowych bloków uniwersalnych . . . . .	11
0.3.2	<i>Zadanie extra</i> Programowanie w LabView – prosty przyrząd wirtualny	15

# Ćwiczenie 0

## Laboratorium zerowe – Matlab, LabView i narzędzia

Opiekun ćwiczenia: KK

### 0.1. Wprowadzenie do zajęć

W pierwszej części ćwiczenia 0 studenci zapoznają się z regulaminem porządkowym/BHP oraz zasadami zaliczenia przedmiotu. Regulaminy te znajdują się w oddzielnym dokumencie, dostępnym na stronie internetowej przedmiotu.

W ćwiczeniu 0 student wyjątkowo nie tworzy protokołu. Warto jednak sformułować ustnie odpowiedzi na pytania występujące w instrukcji i przedyskutować je z kolegami oraz z prowadzącym.

### 0.2. Wprowadzenie do pakietu MATLAB®

MATLAB® jest jednym z podstawowych środowisk komputerowych do obliczeń numerycznych. W laboratorium LCPS jest wykorzystywany do przetwarzania sygnałów i wizualizacji wyników.

MATLAB® zbudowany jest na podstawie prostego języka skryptowego, który jest językiem interpretowanym. Oznacza to, że z linii poleceń MATLAB®'a można wykonać bezpośrednio pojedyncze polecenie, i wywołać gotowy skrypt, działając na zmiennych globalnych uruchomionego środowiska. Polecenia i skrypty mogą korzystać z funkcji – standardowych lub definiowanych przez użytkownika.<sup>1</sup>

Przykładowo wykonanie z linii komendy polecenia

```
2+2
```

```
spowoduje natychmiastową odpowiedź
```

```
ans =
```

```
4
```

---

<sup>1</sup>W warunkach domowych studenci mogą korzystać z darmowego pakietu Octave który używa języka praktycznie identycznego z językiem MATLABa®. Octave dostępny jest na licencji GNU i działa równie dobrze w systemach Linux i Windows.

W dalszym ciągu ćwiczenia kolejne właściwości MATLABa® będą wprowadzane na przykładach praktycznych poleceń wykonywanych przez studenta.

## 0.2.1. Matlab – rozpoznanie walką

“Rozpoznanie walką (ang. reconnaissance in force) - rodzaj natarcia, a także sposób przeprowadzenia rozpoznania taktycznego [...] ma na celu zdobycie najbardziej wiarygodnych informacji.[...] Ten rodzaj natarcia powinien zmusić przeciwnika do ujawnienia rzeczywistych pozycji lub punktów oporu,[...] Rozpoznanie walką prowadzi się w czasie przygotowywania działań lub w trakcie ich trwania, gdy zdobycie pożądaných informacji innymi sposobami jest niemożliwe”. (cytat z Wikipedii).

### 0.2.1.1. Podstawowe obiekty, operatory i polecenia

- U uruchom MATLABa® i poczekaj na zgłoszenie linii komendy<sup>2</sup>. Wykonaj z linii komendy polecenie z poprzedniego przykładu (2+2). Zwróć uwagę na okienko Workspace, w którym pojawi się nowa zmienna<sup>3</sup>.

- Sprawdź wartość zmiennej ans, a potem pomnóż ją przez 3

```
ans
ans*3
```

Podstawowym obiektem w MATLAB®ie jest zmienna, która może przechowywać liczbę (skalar), wektor lub macierz (w ogólnym przypadku wielowymiarową). Zmienne tworzone są przy pierwszym użyciu, a ich typ określany jest automatycznie. Liczby zespolone traktowane są w sposób naturalny. Jednostkę urojoną można oznaczać jak wszyscy: i albo jak elektronicy: j.

Znak % oznacza początek komentarza, rozciągającego się do końca wiersza.

- Utwórz zmienne o różnych typach (nie przepisuj komentarzy z poniższego kodu):

```
X = 1 % podstawienie liczby 1 do zmiennej X
Y = [1, 2, 3] % utworzenie wektora poziomego
Z = [4;5;6] % utworzenie wektora pionowego
M = [1,2;3,4] % utworzenie macierzy kwadratowej
C = [sqrt(-1), 1+2i, -3i] % utworzenie wektora zespolonego
txt = 'Ala ma kota' % utworzenie zmiennej tekstowej
```

Elementy wektorów i macierzy można indeksować stałą lub zmienną; użycie symbolu : (dwukropek) zamiast indeksu oznacza wybranie wszystkich wierszy lub kolumn. W MATLABie® indeksy zaczynają się od wartości 1 (inaczej niż np. w C). Użycie dwukropka jako pojedynczego indeksu do macierzy oznacza przekształcenie jej w wektor kolumnowy.

- Pobierz różne elementy macierzy

```
Y(2) % da nam wynik 2
```

<sup>2</sup>Nie opisujemy jak to zrobić, polegamy tu na ogólnym doświadczeniu studenta z nowoczesnymi systemami operacyjnymi. W chwili pisania tego skryptu, w zainstalowanym w laboratorium systemie należy odnaleźć odpowiednią ikonę i wykonać dwuklik myszą. W przyszłości być może będzie trzeba wyobrazić sobie intensywnie ikonkę Matlabu i następnie wypowiedzieć w myśli jakieś zaklęcie.

<sup>3</sup>Jeśli okienko nie jest widoczne, użyj menu Desktop→Workspace

```

Z(3) % da nam wynik 6
M(2, 1) % da nam wynik 3
M(2, X) % też da nam 3
M(2, :) % da nam wektor [3,4]
M(:, 1) % da nam wektor kolumnowy [1;3]
M(:) % da nam wektor kolumnowy [1;3;2;4]

```

Wektory można generować za pomocą wyrażenia z operatorem postępu arytmetycznego (w tej roli znów : czyli dwukropek).

Słowo kluczowe end użyte w wyrażeniu indeksującym oznacza końcową wartość indeksu.

**+** Wygeneruj sobie różne postępy arytmetyczne:

```

n = 1:10 % postęp z krokiem domyślnym 1
% lub równoważnie
n = [1,2,3,4,5,6,7,8,9,10]
m = 1:10:100 % postęp z krokiem 10
Kot = 10:-1:0 % postęp z krokiem -1
KotOgonem = Kot(end:-1:1) % odwracanie...

```

Tworzenie typowych wektorów i macierzy ułatwiają funkcje:

```

size(x) – podaje rozmiar zmiennej x
zeros(m,n) – tworzy macierz  $m \times n$  wypełnioną zerami; można też wywołać
zeros(size(x))
ones(m,n) – student sam odgadnie albo sprawdzi co ta funkcja robi
eye(n) – tworzy macierz jednostkową  $I_{n \times n}$  (przeczytaj na głos po angielsku
eye oraz I)
rand(m,n) – tworzy macierz wypełnioną (pseudo)losowymi liczbami z rozkładem
równomiernym na odcinku (0, 1)
randn(m,n) – tworzy macierz wypełnioną (pseudo)losowymi liczbami z rozkładem
normalnym

```

Wektory można sklejać (*konkatenować*) “w pionie”  $C=[A;B]$  i “w poziomie”  $R=[A,B]$

Zakończenie polecenia średnikiem wyłącza wydruk wyniku – wynik oczywiście pozostaje w odpowiedniej zmiennej w przestrzeni roboczej.

**+** Utwórz wektor wierszowy r zawierający 3000 elementów, z czego pierwsze 1000 to zera, kolejne to jedynki, a na końcu – liczby losowe o rozkładzie Gaussa.

Uproszczoną dokumentację (“help”) operatorów, funkcji i typów zapewnia polecenie help <temat>

**+** Dowiedz się więcej na temat:

```

– zmiennej ans
– dwukropka
help colon

```

Operacje na zmiennych są realizowane podobnie jak w innych językach komputerowych, przy czym, podobnie jak w językach obiektowych, podstawowe operatory działają różnie w zależności od obiektów jakie są użyte w operacji. Podstawowe operatory algebraiczne działają zgodnie z definicją operacji wektorowych i macierzowych.

Operatory działające na elementach macierzy – przy ich pomocy np. można dodawać (po elementach) obiekty o tej samej wymiarowości lub dodawać liczbę (skalar) do innych obiektów:

- + operator dodawania
- operator odejmowania
- .\* operator mnożenia po elementach
- ./ operator dzielenia po elementach

Operatory macierzowe – działają na macierzach lub gdy jeden z argumentów jest skalarom:

- \* operator mnożenia macierzowego
- / operator dzielenia macierzowego (prawostronnego) ( $C/A = C \cdot \text{inv}(A)$ )
- \ dzielenie lewostronne  $A \setminus B = \text{inv}(A) \cdot B$
- ' apostrof – operator transpozycji macierzy (lub wektora); jest to transpozycja hermitowska  $A^H$  tj. dla zmiennych zespolonych transpozycji towarzyszy sprzężenie
- .' operator transpozycji macierzy (lub wektora) bez sprzężenia

Warto zauważyć funkcję kropki . jako modyfikatora, zazwyczaj (ale nie zawsze) wskazującego że dany operator działa na elementach macierzy (podczas gdy wersja bez modyfikatora działa na całych macierzach).

- 🔧 Użyj operatora mnożenia na wektorach Y i Z aby otrzymać

- ich iloczyn skalarny
- iloczyn macierzowy którego wynik ma rozmiar  $3 \times 3$
- iloczyn skalarny wektora Y przez siebie

*Przypomnij sobie z algebry schemat mnożenia dwóch macierzy i wymagania na ich rozmiarach.*

- 🔧 **Zadanie extra** Oblicz iloczyn macierzy A i wektora x:  $y = Ax$ . Następnie spróbuj obliczyć x znając y oraz A. **Wskazówka:** Najpierw zastanów się (na papierze) nad wymiarami macierzy, żeby to miało sens; potem użyj (wciąż na papierze) mnożenia przez macierz odwrotną; na koniec w Matlabie wykonaj to korzystając z operatora dzielenia macierzy.

- 🔧 Utwórz wektor próbek sygnału sinusoidalnego o częstotliwości unormowanej (z zakresu (0, 0.5)) wybranej poprzez podzielenie numeru Twojego stanowiska przez 30.

Najprostszą metodą tworzenia wektora próbek sygnału jest:

- 1) Wygenerowanie wektora czasowego  
`n = 0:100; % sto jeden chwil czasowych`

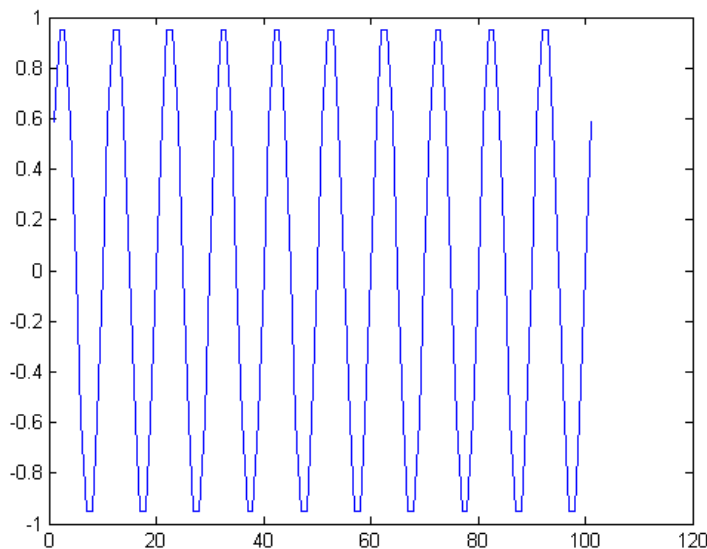
- 2) Zastosowanie odpowiedniej funkcji do wygenerowania przebiegu

Np. do wygenerowania sinusoidy o częstotliwości unormowanej 0.1 można użyć sekwencji  
`n = 0:100; f = 0.1; x = sin(2*pi*f*n);`

### 0.2.1.2. Proste funkcje graficzne

Do tworzenia wykresów najczęściej używa się standardową funkcję `plot()`. Funkcja może przyjmować różną liczbę argumentów.

- narysuj wykres sygnału  $x$  (utworzonego przed chwilą)  
`plot(x)` Powinieneś otrzymując wykres podobny do rys. 1 (ale na każdym stanowisku inny – warto porównać swój wykres z sąsiadami).



Rys. 1: Przebieg sygnału w czasie.

Użyj narzędzia powiększania (przycisk lupy) aby dokładnie obejrzeć obszar wykresu wokół punktu zerowego.

Zwróć uwagę, że funkcja `plot()` z jednym argumentem rysuje wartości elementów wektora względem ich indeksu (biegnącego od 1).

Przy dwóch argumentach (koniecznie o równej długości) pierwszy będzie zinterpretowany jako współrzędna pozioma (oś odciętych), a drugi – pionowa (oś rzędnych).

Trzeci argument o typie tekstowym wybiera kolor i rodzaj linii.

Przy rysowaniu przebiegów dyskretnych w czasie można stosować przykładowo:

'\*' – znaczniki (gwiazdki) w wartościach próbek

'-\*' – znaczniki połączone linią ciągłą

'r-o' – znaczniki (kółeczka) połączone linią ciągłą, w kolorze czerwonym

Poprawnym teoretycznie (z punktu widzenia rysowania przebiegów dyskretnych w czasie) wariantem funkcji `plot()` jest funkcja `stem()`.

Użycie polecenia `figure` powoduje przejście do nowego okna wykresu – jest to przydatne do porównywania kilku przebiegów. Można też narysować kilka przebiegów na jednym wykresie, podając wiele par (lub trójek) argumentów.

- Narysuj wykres sygnału  $x$  względem indeksu chwili czasowej  $n$  (zwróć uwagę, że ten indeks biegnie od zera).

- Narysuj ten sam wykres w nowym oknie używając funkcji `stem()` oraz linii `'-*`'. Który sposób rysowania daje bardziej czytelny wykres?<sup>4</sup>

Osie wykresu opisuje się używając funkcji `xlabel('<tekst>')`, `ylabel('<tekst>')`, a tytuł dodaje się przy pomocy funkcji `title('<tekst>')`. Siatkę włącza się poleceniem `grid`. Legendę dla wielu krzywych na jednym wykresie dodaje się funkcją `legend('<legenda1>', '<legenda2>', ...)`. Funkcje te działają na ostatnim aktywnym wykresie.

- Do ostatniego wykresu dodaj opis osi, tytuł i siatkę.

Funkcja `plot()` może zostać wywołana z jednym argumentem zespolonym – wykreśli wtedy jego wartości we współrzędnych ( $\Re$ ,  $\Im$ ). W innych przypadkach argumenty powinny być rzeczywiste.

- Wykonaj polecenie:

```
plot(exp(j*0.01*2*pi*[0:100])) i uzasadnij otrzymany wykres.
```

### 0.2.1.3. Podstawowe operacje na sygnałach dyskretnych

- Utwórz w zmiennej `d` impuls jednostkowy  $\delta_n$  o czasie trwania 100 próbek, przyjmując, że chwila  $n = 0$  odpowiada dwudziestej próbce; zastosuj następujący trick:

- utwórz zmienną `n=-19:80`
- wypełnij zmienną `d` zerami
- na dwudziesty element podstaw 1

Narysuj przebieg sygnału za pomocą funkcji `stem()`.

- Utwórz analogicznie skok jednostkowy  $u(n)$  w zmiennej `u` – tym razem wykorzystaj sklejanie wektorów.

Wbudowana funkcja `filter(B,A,x)` implementuje filtr cyfrowy o współczynnikach licznika transmitancji w wektorze `B` i mianownika – w wektorze `A`.

- Przetestuj filtr o współczynnikach  $B = [-1, 2, -1]$ ,  $A = [1, 0.9]$  sygnałami  $\delta(n)$  i  $u(n)$ , przykładowo:

```
B=[-1,2,-1];
A=[1,0.9];
plot(n,d, '-*', n,filter(B,A,d), '-o');
legend('wejście', 'wyjście');
```

- Wyznacz widmo sygnału sinusoidalnego `x` z poprzedniego zadania za pomocą funkcji `fft`

```
F = fft(x);
```

Zwizualizuj wyliczone widmo za pomocą komendy

```
plot(abs(F))
```

Odpowiedz na pytanie - dlaczego używamy funkcji `abs()`?

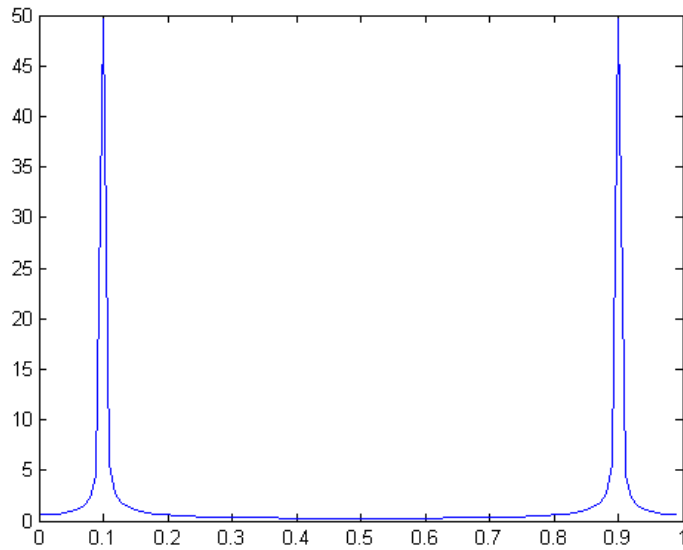
- Narysuj ten sam wykres z poprawnym wyskalowaniem osi częstotliwości unormowanych:

```
f0 = (0:(length(F)-1))/length(F); % wektor częstotliwości
plot(f0, abs(F)); % wyrysowanie widma
```

- Wykreśl widmo w skali decybelowej wraz z tytułem, siatką i opisem osi za pomocą zestawu komend:

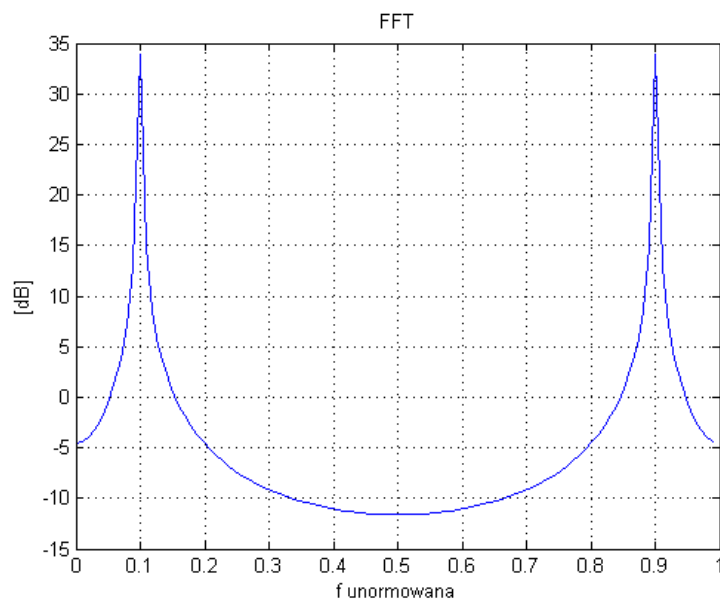
<sup>4</sup>Nie dyskutując o gustach, przed udzieleniem odpowiedzi sugerujemy zajrzenie na ekrany kolegów, którzy będą badali sygnały o różnych częstotliwościach.





Rys. 2: Moduł widma sygnału sinusoidalnego.

```
plot(f0, 20*log10(abs(F)));  
xlabel('f unormowana'); ylabel('moduł widma [dB]'); title('FFT'); grid  
    uzyskując efekt jak na rys. 3. W razie potrzeby użyj narzędzia powiększenia (przy-  
    cisk lupy) lub polecenia axis([0, 1, -15, 35]) do zmiany zakresu osi. Dowiedz się,  
    co oznaczają cztery liczby w argumencie axis().
```



Rys. 3: Moduł widma sygnału sinusoidalnego w skali decybelowej.

### 0.2.1.4. Tworzenie funkcji

Funkcje tworzy się w edytorze wywołując np. wbudowany edytor MATLABa®

```
edit plot_fft.m
```

Nazwa funkcji powinna być zgodna z nazwą pliku w którym jest zapisana. Pierwsze wiersze komentarza stanowią skróconą dokumentację – zostaną wyświetlone w wyniku polecenia

```
help plot_fft
```

✚ Wpisz w edytorze kod przykładowej funkcji:

```
function [F, f0] = plot_fft(x)
% tu zaczyna sie HELP
%
% [F, f0] = plot_fft(x)
% Funkcja do wyznaczania i rysowania widma w dB
%
% - funkcja zwraca wektor widma F i wektor częstotliwości f0
% - jeżeli nie wykorzystujemy wektora F funkcja rysuje widmo
% to jest ostatnia linia HELP

% to jest już tylko komentarz

F = fft(x); % wyznaczenie widma
f0 = (0:(length(F)-1))/length(F); % wyznaczenie punktów częstotliwości

% rysowanie
plot(f0, 20*log10(abs(F)));
xlabel('f unormowana'); ylabel('moduł amplitudy widma [dB]'); title('FFT');
grid

end % koniec funkcji
```

✚ Przetestuj wywołanie funkcji:

```
plot_fft(x); % bez pobrania parametrów wyjściowych (tylko plot)

F = plot_fft(x); % z pobraniem pierwszego parametru (widma)

[F, f0] = plot_fft(x); % ... i obu parametrów (widma i częstotliwości)
```

Matlab jest środowiskiem bardzo rozbudowanym. Jego funkcjonalność znacznie przekracza to co tu omówiono, ale przedstawiona wiedza wystarcza do posługiwania się nim w zakresie potrzebnym w laboratorium.

## 0.3. Wprowadzenie do pakietu LABVIEW

LabVIEW (LV) jest graficznym środowiskiem programowania stworzonym przez firmę National Instruments (NI). Służy ono do szybkiej obsługi urządzeń i przyrządów pomiarowych, przetwarzania zarejestrowanych danych oraz prezentacji wyników. W laboratorium LCPS jest wykorzystywany głównie do obsługi karty akwizycji danych NI PCIe-6321 wraz z panelem złącz BNC-2120.

Podstawowym obiektem w LabView jest przyrząd wirtualny nazywany skrótowo VI<sup>5</sup> (*ang. Virtual Instrumentation*), czyli moduł programowy który może przechowywać, przetwarzać oraz prezentować nagromadzone dane. Dane mogą być zorganizowane w postaci wektorów, macierzy (w ogólnym przypadku tablic wielowymiarowych) lub też struktur.

Przyrząd wirtualny VI<sup>6</sup> wyglądem i działaniem przypomina pracę urządzenia rzeczywistego. Każdy moduł programowy VI opisany jest w sposób graficzny na trzech poziomach szczegółowości:

1) *panel czołowy* – stanowiący graficzny interfejs użytkownika (rys. 4). Składa się z KONTROLK (WEJŚĆ) oraz INDYKATORÓW (CZYLI WYJŚĆ) modułu.

2) *schemat blokowy* – stanowiący graficzny kod źródłowy programu tzw. diagram przepływu informacji (rys. 7). Obiekty panelu czołowego tj. kontrolki oraz indykatory widoczne są w postaci zacisków wyprowadzeń (*ang. terminals*).

3) *porty wejścia/wyjścia* – zaciski wyprowadzeń VIa (rys. 6). Typy argumentów reprezentowane są w sposób info-graficzny poprzez tzw. terminale.

Każdemu VIowi przyporządkowana jest ikona stanowiąca jego miniaturę graficzną (rys. 5). Ikona mapuje jednoznacznie *porty wejścia/wyjścia* VIa, umożliwiając „enkapsulację” kodu do postaci proceduralnej (*ang. subVI*<sup>7</sup>) (rys. 6).

Przyrządy wirtualne VI mają budowę hierarchiczną i modułową. Dzięki zamknięciu do podprocedur, subVIe mogą być używane jako samodzielne programy lub podprogramy innych programów (aplikacji).

### 0.3.1. Przyrząd złożony z gotowych bloków uniwersalnych

W tym rozdziale stworzymy przyrząd pomiarowy z gotowych bloków uniwersalnych. Bloki te dostępne są w sekcjach “Express” palet funkcji i kontrolki. Jest to bardzo szybka metoda prototypowania przyrządu – niewielkim kosztem otrzymamy działające urządzenie.

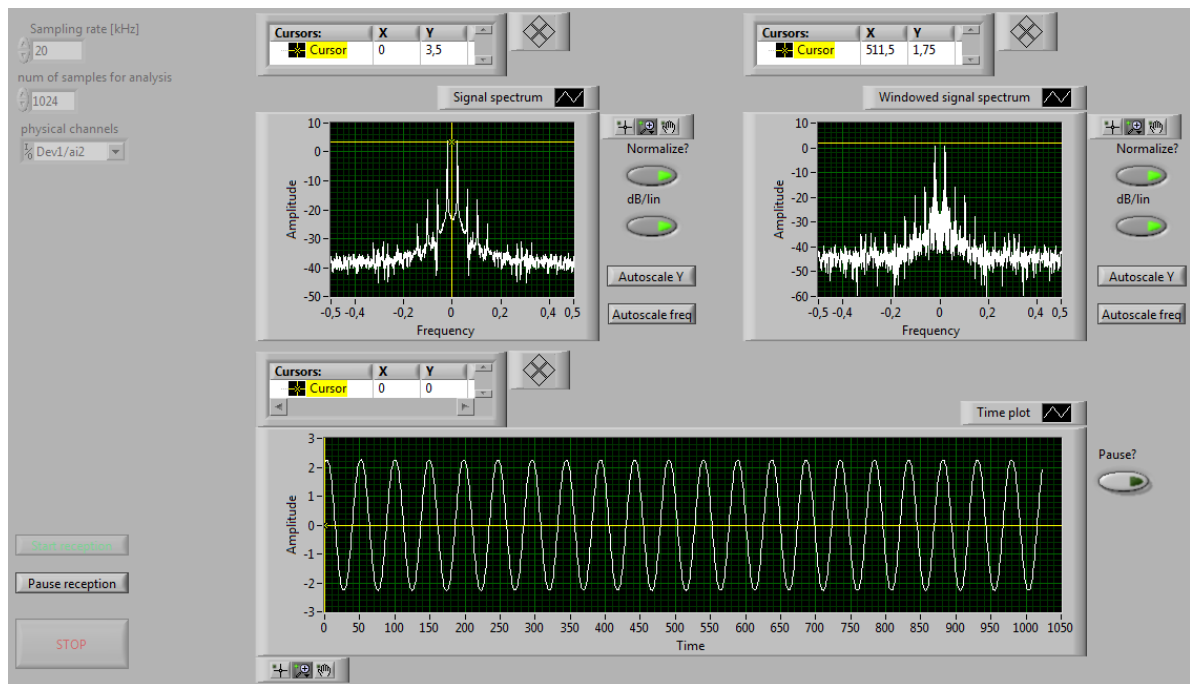
W następnym rozdziale poznamy bardziej żmudną metodę konstruowania przyrządu (ale dającą większą kontrolę i zazwyczaj większą pewność działania urządzenia) – poprzez programowanie go z wykorzystaniem obiektów elementarnych.

- ➊ Uruchom okno startowe LABVIEW(®) za pomocą ikony na pulpicie)
- ➋ Z okna startowego LV wybierz **Create Project** → **Blank VI** i przejdź do edycji nowo otwartych okien pustego VIa. Pierwsze z okien (szare) jest widokiem PANELU CZOŁOWEGO,

<sup>5</sup>czyt. *vi-aj*

<sup>6</sup>Skrótu tego używamy dalej zgodnie z nomenklaturą *NI* w odniesieniu do przyrządu wirtualnego zrealizowanego w postaci modułu programowego; bez entuzjazmu godzimy się z faktem, że po polsku brzmi on fatalnie i równie fatalnie wygląda w druku.

<sup>7</sup>czyt. *sab-vi-aj*



Rys. 4: Widok panelu czołowego VI.



Rys. 5: Ikona VI.

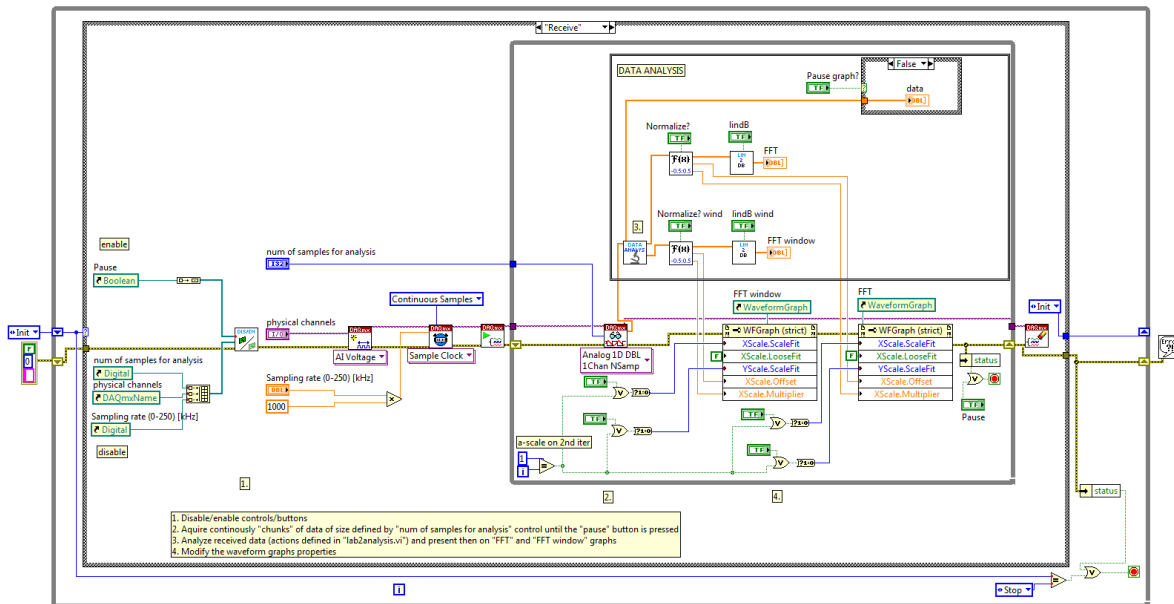


Rys. 6: Mapa wyprowadzeń zacisków VI; wejścia – strona lewa; wyjścia – strona prawa; kolory oznaczają typy argumentów;

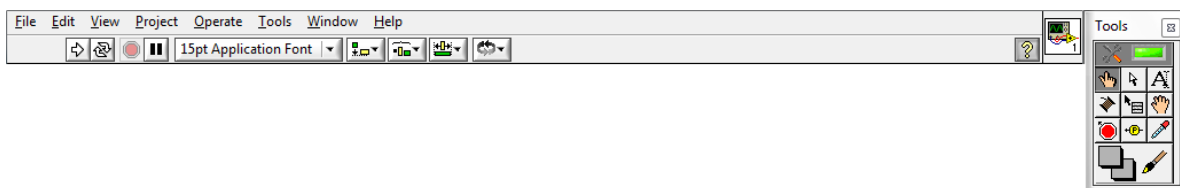
drugie (białe) jest oknem widoku DIAGRAMU BLOKOWEGO, oba tworzą kod źródłowy VI.

W dowolnym momencie edycji można przełączyć się między widokami w panelu czołowym na diagram blokowy wybierając z górnego paska stanu okna opcję **Window** → **Show Block Diagram** i na odwrót **Window** → **Show Front Panel** (można użyć skrótu klawiszowego **Ctrl+E**). Po uruchomieniu VIa, użytkownik widzi tylko panel czołowy zaś diagram blokowy jest ukryty – “zaszyty” pod panelem czołowym.

Przyciski widoczne na paskach stanu obu okien są zgromadzone w trzech grupach (rys. 8, rys. 9). Zaczynając od lewej widzimy przyciski związane z uruchamianiem, „debugowaniem” i wstrzymywaniem VIa, następnie pośrodku jest grupa przycisków związanych z porządkowaniem jego widoku i wreszcie na końcu po prawej stronie przycisk pomocy kontekstowej (pytajnik) oraz widok ikony VIa. Pomoc kontekstowa wyświetla informacje o obiektach kodu wskazanych kursorem myszy, i dostępna jest także pod skrótem **Ctrl+H**.



Rys. 7: Schemat blokowy VI; wejścia – strona lewa; wyjścia – strona prawa;



Rys. 8: Pasek górnego menu i narzędzi dostępny w oknie widoku panelu czołowego.

PRACA ZE SCHEMATEM BLOKOWYM VIa polega na dosłownie „narysowaniu kodu aplikacji” tj. wybieraniu (z odpowiedniej palety) „bloczków” potrzebnych do zrealizowania programu, a następnie połączeniu ich tworząc właściwy diagram przepływu informacji/danych w programie. Dostęp do palety funkcyjnej uzyskujemy w ten sam sposób co uprzednio, poprzez kliknięcie prawym przyciskiem myszy w pustym obszarze okna schematu blokowego lub wybraniu **View** → **Functions Palette** z górnego paska menu.

- Zaczynij od edycji schematu blokowego, na którym umieścisz generator sygnału. Kliknij w białym polu prawym klawiszem myszy i z menu wybierz **Express** → **Input** → **Simulate sig**. Przeciągnij wybrany obiekt na schemat. Obiekty z menu “Express” stanowią gotowe, dość uniwersalne fragmenty kodu (lub generatory kodu), które wstępnie konfiguruje się w chwili umieszczania na schemacie.
- W oknie konfiguracji generatora nie musisz nic zmieniać - zakończ konfigurację [OK]. Wygeneruje się VI generatora, opatrzonej portami we/wy (terminalami).





Rys. 9: Pasek górny narzędzi dostępny w oknie widoku diagramu blokowego.

W schemacie blokowym możemy wyróżnić następujące elementy składowe:

- *porty wejścia i wyjścia* - terminale, jedyne elementy widoczne na zewnątrz tj. w panelu czołowym
- *subVie* – czyli podprocedury zamknięte w pojedynczym module VI
- *funkcje* – realizujące podstawowe zadania odwzorowania/przekształcania danych
- *stałe* – wartości niezmiennie w czasie wykonywania programu
- *struktury* – danych, organizujące informacje w programie
- *połączenia* – symboliczne kablowe, przesyłające dane pomiędzy blokami

- ➊ Umieść na panelu czołowym wyświetlacz, rysujący przebieg czasowy sygnału. Z menu kontekstowego wybierz **Express** → **Graph Indicator** → **Graph**. Przeciągnij wybrany obiekt na schemat. Ten obiekt nie wymaga konfiguracji.

Po umieszczeniu niezbędnych kontrolerek/indykatorów na panelu czołowym VIa, mamy możliwość dostosowania ich parametrów – położenia, rozmiaru, zakresu przyjmowanych wartości i wielu innych właściwości – klikając prawym przyciskiem myszy na wybranym obiekcie i wybierając z menu kontekstowego opcję **Properties**.

- ➋ Przełącz się na schemat blokowy i zauważ, że pojawił się na nim terminal wejściowy wyświetlacza.
- ➌ Na obrysie obiektu generatora znajdują się jego terminale. Znajdź terminal wejściowy zadający częstotliwość – wykorzystaj podpowiedzi kontekstowe, pojawiające się przy najechaniu kursorem myszy, albo rozciągnij ikonę w pionie, aż pojawią się podpisy terminali. Na terminalu naciśnij prawy klawisz myszy i stwórz odpowiednią kontrolkę (**Create** → **Control**). Na panelu czołowym automatycznie pojawi się widget kontrolki dostosowany do typu danych terminala.
- ➍ Narysuj połączenie terminala wyjściowego generatora z wejściem wyświetlacza.
- ➎ Uruchom przyrząd przyciskiem pracy ciągłej  *Run Continuously*. Zmieniaj częstotliwość operując myszką na kontrolce panelu czołowego. Zatrzymaj działanie VIa przyciskiem „Stop”  *Abort Execution*.
- ➏ Dodaj teraz nowy wyświetlacz i wyświetl na nim widmo sygnału. Widmo oblicz przy użyciu bloku **Express** → **Signal Analysis** → **Spectral**. Zastosuj domyślną konfigurację bloku obliczeniowego (chyba, że jesteś odważnym eksperymentatorem... ). Przetestuj działanie.
- ➐ Zwróć uwagę, że wyświetlacz sygnału ma poprawną skalę czasu – wynika to z tego, że razem z blokiem próbek sygnału przekazywana jest informacja o okresie próbkowania.
- ➑ Zwróć uwagę, że wyświetlacz widma ma porównaną skalę częstotliwości, ale niepoprawny opis osi – jak widać, są pewne niedoróbki... Możesz to poprawić w menu właściwości (“Properties”) wyświetlacza.
- ➒ Zastąp symulator sygnału rzeczywistym wejściem analogowym. Umieść na schemacie obiekt “DAQ Assistant” (z menu **Express** → **Input**). W konfiguracji wstępnej obiektu wybierz “acquire”, “Analog input”, “Voltage”. Pojawi się możliwość wybrania urządzenia - jeśli pojawią się do wyboru dwa urządzenia, wybierz pierwsze - i koniecznie **[+]** **rozwiń** drzewko i wybierz z niego tylko kanał A0 (za chwilę fizycznie, “w realu”, podłączysz sygnał do gniazda BNC oznaczonego A0). Teraz można nacinać [Finish].

- ⊕ Na drugim etapie konfiguracji ustaw częstotliwość próbkowania (“Rate”) na 10 kHz, a liczbę próbek (“Samples to read”) na 1000. Zanim zakończysz konfigurację, wyjmij z szuflady kable oraz trójnik i podłącz sygnał z generatora na wejście “AI 0” (te czynności wykonaj “w realu”!!!) i pobierz blok próbek (przycisk “RUN” w konfiguratorze). Sprawdź, czy na pewno widzisz ten sygnał, który jest podłączony (np. zmień jego amplitudę i ponownie uruchom “Run”).

| Zawsze warto sygnał z generatora podejrzeć też – przez trójnik – na oscyloskopie.

- ⊕ Po zakończeniu konfiguracji [OK] i wygenerowaniu przez LabView obiektu dołącz go zamiast symulatora i przetestuj działanie całości.
- ⊕ Dodaj według uznania kontrolki umożliwiające zmiany parametrów urządzenia.
- ⊕ Zbadaj jak wygląda sygnał i jego widmo, gdy częstotliwość sygnału przekroczy częstotliwość próbkowania.

Karta akwizycji nie posiada filtru antyaliasingowego i jej układ próbkujący ma pasmo do pojedynczych MHz. To oznacza, że praktycznie – przy sygnałach z zakresu do dziesiątek kiloherców – pobrana próbka jest “punktowa” a nie uśredniona za okres (odstęp) próbkowania  $T$ , co umożliwia pracę z podpróbkowaniem (undersampling).

### 0.3.2. Zadanie extra Programowanie w LabView – prosty przyrząd wirtualny

Na przykładach omówimy krok po kroku jak samemu, z podstawowych elementów, stworzyć prosty VI.

Zrealizujemy przyrząd, który będzie pełnił rolę uniwersalnego generatora sygnałów dyskretnych i analizatora ich widma.

- 1) Zamknij okna poprzedniego przyrządu VI, i z okna głównego stwórz nowy VI
- 2) Wejdź do okna schematu blokowego

#### 0.3.2.1. Tworzenie wektorów sygnału

Poniżej przedstawiono jeden ze sposobów realizacji zadania.

W celu wytworzenia wektora danych potrzebujemy w pierwszej kolejności skonstruować wektor (zdyskretyzowanego) czasu  $n=[0:N]$  a następnie zastosować pewne odwzorowanie/funkcję czasu na próbki  $f[n]$ . Na wstępie przykładu wytworzymy trzy różne wektory danych: narastających liniowo, zmieniających się losowo oraz uformowanych harmonicznie.

- 1) W oknie schematu blokowego dodaj niezbędne elementy wybierając je z palety funkcyjnej. Kliknij prawym przyciskiem myszy i wybierz strukturę pętli FOR: **Programming** —> **Structures** —> **For Loop**<sup>8</sup>, wstaw ją do schematu blokowego obrysowując dookoła obszar/układu (jak na rys. 11)<sup>9</sup>. Pętla zawiera licznik obrotów [N] oraz wskaźnik numeru aktualnie wykonywanej iteracji/obrotu [i] (indeksowany od zera).

<sup>8</sup>możesz również skorzystać z opcji tzw. szybkiego wybierania „Quick Drop” wciskając kombinację klawiszową Ctrl+Space poczym wpisać w polu wyszukiwania żądane słowo (nazwę poszukiwanego elementu) a następnie wybrać go z listy i przeciągnąć na schemat

<sup>9</sup>można dowolnie modyfikować rozmiar pętli pociągając wskaźnikiem myszy za jej krawędź lub też będąc w obszarze pętli naciskając klawisz Ctrl+lewy przycisk myszy zarysować obszar – wykorzystywane często do robienia miejsca w gęsto zabudowanym diagramie



2) Dodaj terminale wejściowe i wyjściowe VIa poprzez klikanie prawym przyciskiem myszy na elementach pętli<sup>10</sup>.

Terminal wejściowy dla licznika pętli [N] definiujący długość wektora danych: kliknij prawym przyciskiem myszy na liczniku pętli [N] i wybierz z menu kontekstowego **Create** → **Control**.

Terminal wyjściowy wygenerowanego wektora danych: ustaw kursor myszy na indeksie pętli [i]; pojawi się wtedy port wyjściowy indeksu. Port ten połącz z krawędzią bloku pętli: kliknij na porcie i na krawędzi (tu konieczne jest dość dokładne trafienie w krawędź). Wynik powinien przypominać rys. 11).

Zauważ że, na obwodzie pętli tam gdzie dochodzi połączenie/kabel pojawił się tzw. „tunel”, dzięki czemu na wyjściu (po wykonaniu się pętli) otrzymamy wektor danych.


Tunelami przekazywane są dane z i do struktur programistycznych, struktura wykona (lub zakończy) się jeśli wszystkie dane są obecne w tunelu(ach) - pełna synchronizacja. Domyślnie tunele przechodzące przez krawędź pętli podlegają auto-indeksacji, można ją wyłączyć z menu kontekstowego tunelu opcją **Disable Indexing**.

3) Dodaj wskaźnik danych z tunelu wyjściowego, klikając nań prawym przyciskiem myszy wybierz z menu kontekstowego **Create** → **Indicator**. Zmień jego typ z integer na double, wybierając z menu kontekstowego **Representation** → **DBL** i zauważ na ikonie marker rzutowania typów danych.

4) Uporządkuj rozmieszczenie kontrolki i indykatorów na schemacie po czym przejdź do panelu czołowego.

Dobłą praktyką jest aby na schemacie blokowym wejścia umieszczać z lewej strony, a wyjścia z prawej; schemat pojedynczego VIa powinien mieścić się na jednym ekranie.

Kontrolki i indykatory automatycznie pojawiły się na panelu czołowym VIa. Opisz je sensownie i uporządkuj wygląd interfejsu graficznego (GUI) (rys. 10). Wstaw początkowe wartości kontrolki (i/lub indykatorów) wpisując je doń z klawiatury, korzystając z menu kontekstowego/właściwości (naklikując prawym przyciskiem myszy na obiekcie) ustaw jako domyślne **Data Operations** → **Make Current Value Default**. Rozciągnij indykator wektora danych na kilkanaście pól znaczących pociągając myszką za jego dolną krawędź (patrz rys. 10).

5) Uruchom VI przyciskiem „strzałki” pojedynczego wykonania  *Run* z górnego paska narzędzi okna<sup>11</sup>, na wyświetlaczu zaobserwuj wynik.



Zauważ że, skonstruowany wektor chwil czasowych zaczyna się od zera a chcielibyśmy aby zaczynał się od jedynki (jak dla przykładów w Matlab'ie). Należy zatem zwiększyć wartość indeksu pętli o jeden: wróć do diagramu blokowego, kliknij prawym przyciskiem myszy na połączeniu [i] < -- > tunel i wybierz **Insert** → **Numeric Palette** → **Increment** (patrz

<sup>10</sup>klikając prawym przyciskiem myszy na konektorach poszczególnych struktur i bloków, LV w menu kontekstowym podpowiada nam różne możliwości wyboru m.in. właściwość zakończenia/terminala **Create** → **Constant, Control, lub Indicator**

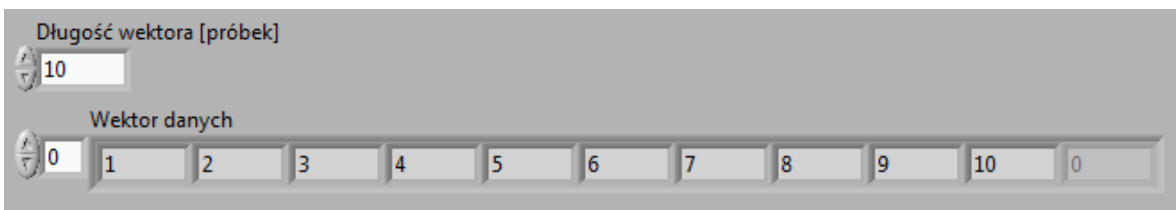
<sup>11</sup>Ctrl+R - skrót klawiszowy uruchamiający wykonanie VIa



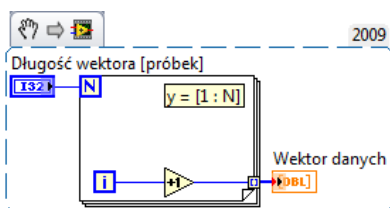
rys. 11).

6) Uruchom ponownie VI, tym razem w trybie pracy ciągłej wciskając przycisk „obiegu zamkniętego”  *Run Continuously*. Kontrolę nad działającym VIem sprawujemy z pomocą użycia myszki i klawiatury, zweryfikuj działanie VIa zmieniając długość wektora. Zatrzymaj działanie VIa przyciskiem „czwonego kółka”  *Abort Execution*.

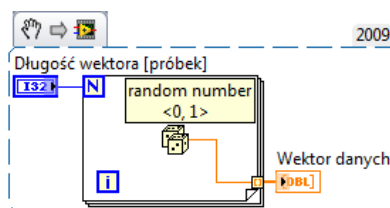
Dobłą praktyką jest używanie przycisku *Abort Execution* wyłącznie w sytuacjach awaryjnych – każda aplikacja powinna być wyposażona w programowy przycisk STOPu wstrzymujący jej działanie



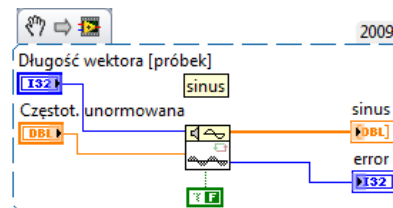
Rys. 10: Widok panelu czołowego generatora wektorów z indeksami biegnącymi od jedynki; wejścia – góra; wyjścia – dół;



Rys. 11: Diagram blokowy generatora wektorów liniowych; wejścia – strona lewa; wyjścia – strona prawa;



Rys. 12: Diagram blokowy generatora wektorów losowych; wejścia – strona lewa; wyjścia – strona prawa;



Rys. 13: Diagram blokowy generatora wektorów harmonicznych; wejścia – strona lewa; wyjścia – strona prawa;

7) Teraz zaimplementuj generator wektorów losowych. Usuń funkcję **Increment** zaznaczając ją (klikając) lewym przyciskiem myszy, po czym naciśnij klawisz Delete. Dodaj i podłącz obiekt generacji losowej **Programming** → **Numeric** → **Random Number** (jak na rys. 12), usuń złamane połączenie(a)<sup>12</sup> klikając nań raz lewym przyciskiem myszy i Delete. Sprawdź, czy generator działa zgodnie z oczekiwaniami.

8) Zaimplementuj generator sygnału sinusoidalnego. Usuń pętlę FOR tak jak poprzednio usuwałeś elementy schematu, dodaj blok generacji sinusa **Signal Processing** → **Signal Generation** → **Sine Wave**. Obejrzyj jego pomoc kontekstową. Kliknij dwa razy na bloku aby otworzyć jego panel czołowy, użyj Ctrl+E aby obejrzeć wewnętrzny schemat blokowy. Po pobieżnym zapoznaniu się zamknij te okna – blok będzie wykorzystywany jako sub-VIi sterowany za pomocą konektorów.

Na wejściu bloku generacji dodaj kontrolkę częstotliwości unormowanej<sup>13</sup> oraz indykator

<sup>12</sup>Ctrl+B - skrót klawiszowy służący do czyszczenia schematu ze złych połączeń

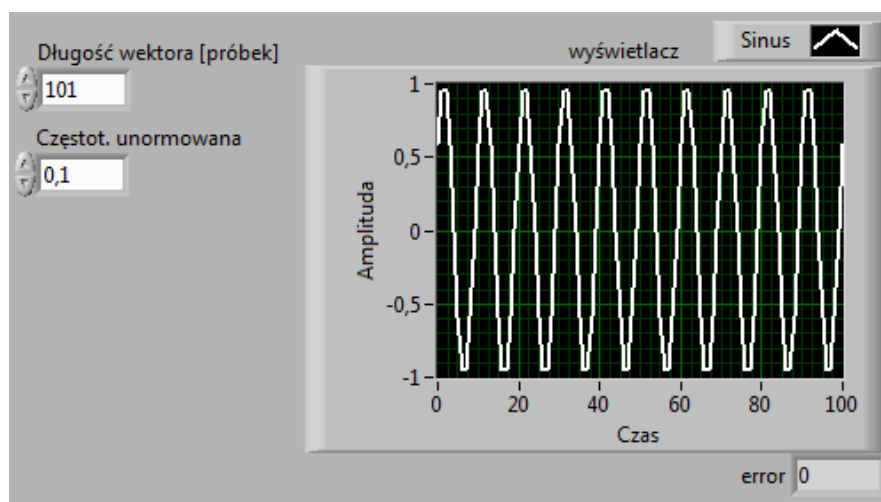
<sup>13</sup>z menu kontekstowego właściwego jej konektora, patrz również okno pomocy kontekstowej

błędu generacji<sup>14</sup>, połącz elementy<sup>15</sup> jak na (rys. 13)

### 0.3.2.2. Wyświetlanie wykresów – widmo sygnału

1) Wizualizacja widma sygnału: zanim tego dokonasz zmień na panelu czołowym VIa (przykładu z rys. 13) indyktor na graf **Replace** → **Graph** → **Waveform Graph**. Opisz osie X i Y grafu jako osie czasu i amplitudy – we właściwościach grafu. Tamże wybierz rysowanie linią ciągłą ze znacznikami w pozycjach próbek.

Ustaw wartość 101 jako domyślną długość sygnału oraz 0,1 dla częstotliwości unormowanej (zwróć uwagę na *przecinek* dziesiętny – zgodnie z ustawieniami lokalnymi...). Zobacz jak wygląda w czasie wygenerowany sygnał (rys. 14) i porównaj go z wynikami uzyskanymi w MATLAB®'ie (rys. 1).



Rys. 14: Widok panelu czołowego generatora wektora harmonicznej; wejścia – strona lewa; wyjścia – strona prawa;

2) Rozbuduj przykład o analizę widmową dodając bloki obliczania FFT oraz modułu. Znajdź niezbędne elementy na palecie funkcyjnej tj. blok **FFT**<sup>16</sup> oraz **Complex To Polar**<sup>17</sup>, połącz całość jak na (rys. 15). Wyskaluj oś częstotliwości wyświetlacz tak aby była w zakresie od 0 do 1, korzystając z właściwości grafu w zakładce **Properties** → **Scales** → **Scaling Factors** → **Multiplier** wpisz 0,01 co w przybliżeniu odpowiada  $1/\text{długość\_wektora}$ .

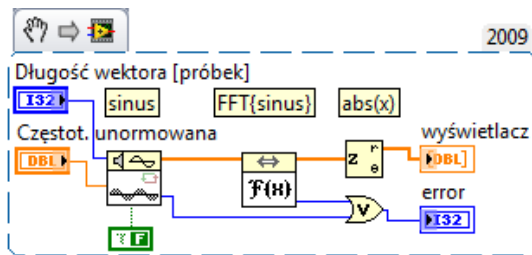
Na wykresie możemy zauważyć dwa charakterystyczne piki występujące w widmie sygnału, ich położenie na osi częstotliwości unormowanej oraz wysokość/amplitudy (rys. 17).

<sup>14</sup>na wypadek, gdyby takowy w ogóle wystąpił

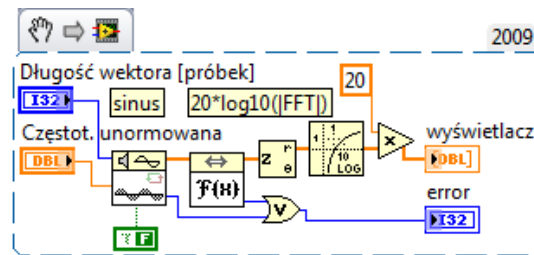
<sup>15</sup>Ctrl+Space+D uruchamia automatyczne dodawanie w LV terminali do niepodłączonych portów wejścia/wyjścia dla elementu aktualnie zaznaczonego myszą; wszystkim portom zostaną przyporządkowane odpowiednie terminale; operacja taka może jednak trwać kilka-kilkanaście sekund.

<sup>16</sup>Signal Processing → Transforms → FFT

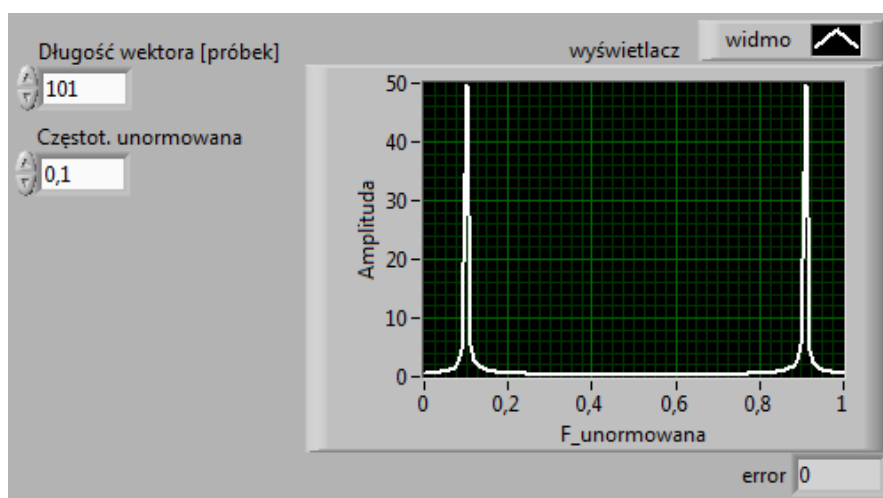
<sup>17</sup>Mathematics → Numeric → Complex → Complex To Polar



Rys. 15: Diagram blokowy plotera widma wektora danych: skala liniowa; wejścia – strona lewa; wyjścia – strona prawa;



Rys. 16: Diagram blokowy plotera widma wektora danych: skala decybelowa; wejścia – strona lewa; wyjścia – strona prawa;

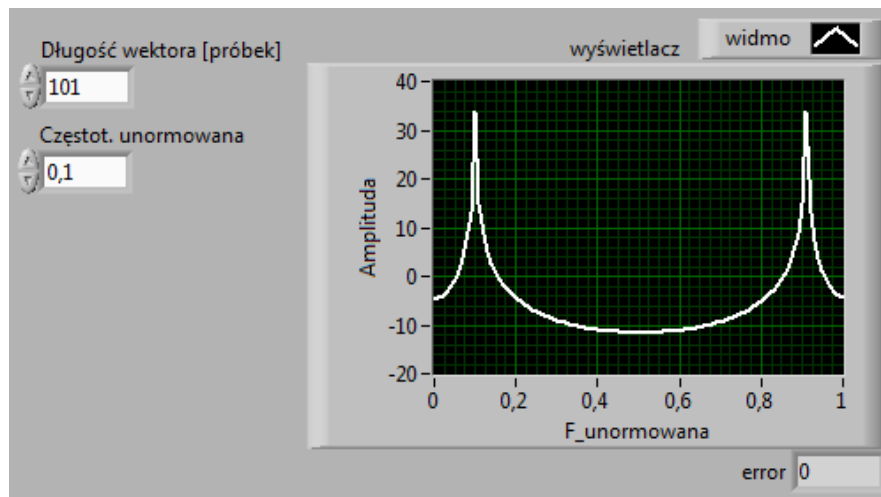


Rys. 17: Widok panelu czołowego analizatora widma sygnału harmonicznego; wejścia – strona lewa; wyjścia – strona prawa;



3) Zmodyfikuj diagram tak aby wyświetlić widmo w skali decybelowej (patrz rys. 16). Dodaj do schematu blok logarytmowania **Logarithm Base 10**<sup>18</sup> oraz mnożnik **Multiply**<sup>19</sup> do przeskalowania z logarytmu na decybele. Stałą o wartości 20 utwórz *przed* połączeniem bloku logarytmu i mnożnika – inaczej stała zostanie uznana za wektor zamiast skalar. Przeprowadź połączenia zgodnie ze schematem przykładu (rys. 16) i uruchom VI.




<sup>18</sup>Mathematics –>Elementary –>Exponentials –>Logarithm Base 10


<sup>19</sup>Mathematics –>Numeric –>Multiply



Rys. 18: Widok panelu czołowego analizatora widma sygnału harmonicznego wyskalowanego w decybelach; wejścia – strona lewa; wyjścia – strona prawa;

LabVIEW umożliwia podejrzenie kolejności wykonania poszczególnych operacji programu. W tym celu należy kliknąć przycisk „żarówki”  *Highlight Execution*, dostępny w górnym pasku narzędzi diagramu blokowego. Uruchomienie programu z załączoną opcją podglądu powoduje wykonanie go w zwolnionym tempie z animacją przepływu informacji .

Użycie mechanizmu sondy  *Probe* pozwala podejrzyć wartości danego połączenia. Można również ustawić „pułapkę”  *Breakpoint* w kodzie programu na którymś z połączeń, spowoduje to zatrzymanie wykonania VI na pułapce i przejście w tryb wstrzymania  *Pause*.

Jeśli w programie występują błędy połączeń, zostanie to zasygnalizowane złamaną linią połączeń oraz złamaną „strzałką” przycisku uruchomienia  *Run*. Po jej naciśnięciu wyskoczy okno listy błędów<sup>a</sup> VI.

<sup>a</sup>Ctrl+L - skrót klawiszowy pozwala podejrzyć listę błędów

- 4) Wprowadź umyślnie błąd w schemacie – usuń dowolne połączenie. Obejrzyj listę błędów.