# Zastosowania Procesorów Sygnałowych

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

# Generowanie sygnałów na DSP

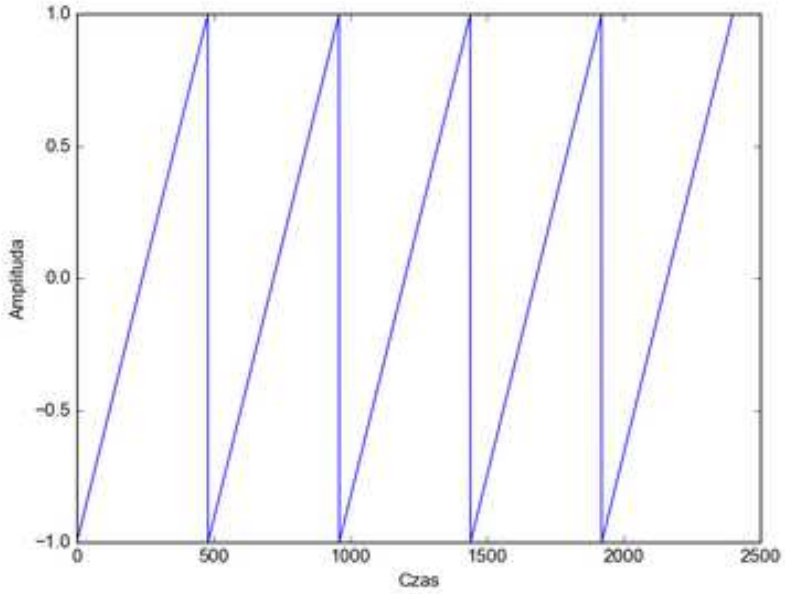
# Wstęp

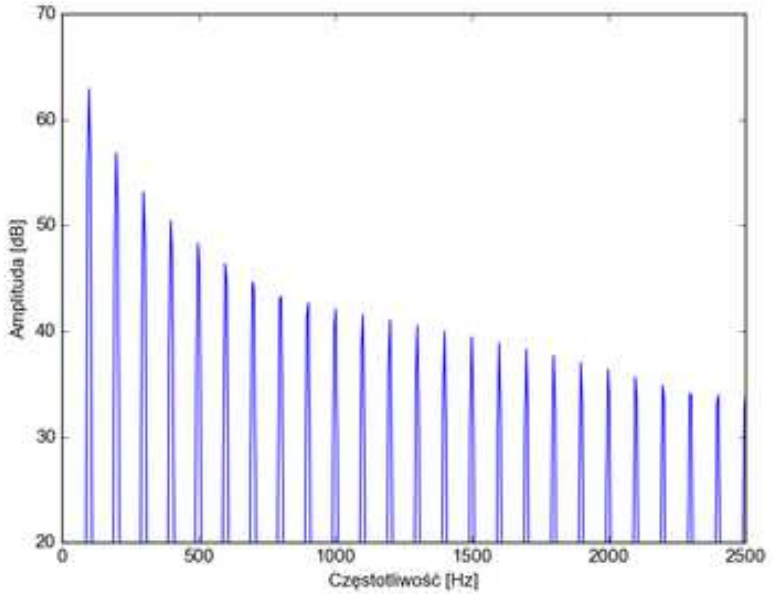
Dziś w programie: generowanie sygnałów za pomocą procesora sygnałowego:

- sygnały harmoniczne
- sygnał sinusoidalny – różne metody
- szum biały
- generowanie dowolnych sygnałów z tablicy próbek
- interpolacja

# Sygnał piłokształtny

- Sygnały harmoniczne: mają widmo, w którym występują prążki w szeregu harmonicznym: na wielokrotnościach częstotliwości podstawowej.
- Przykład: sygnał piłokształtny (*sawtooth*, „rampa”).



# Sygnał piłokształtny

- Amplituda sygnału zmienia się liniowo.
- Inicjalizacja:

```
int amplituda = 0;
int krok = ???;
```

- Dla każdej próbki, wyjście *y*:

```
y = amplituda;
amplituda = amplituda + krok;
```

- Ile wynosi *krok*?

# Obliczenie kroku amplitudy

- Załóżmy częstotliwość 1 Hz, fs = 48 kHz.
- Potrzeba 48 000 próbek aby amplituda zmieniła się od -32768 do 32768.
- Zmiana amplitudy na jedną próbkę wynosi:

$$d = \frac{2 \cdot 32768}{48000} = 1{,}365333...$$

- A jeżeli chcemy częstotliwość 100 Hz?

$$d = \frac{2 \cdot 32768}{48000/100} = 136{,}5333...$$

# Obliczenie kroku amplitudy

- Dla dowolnej częstotliwości *f*, krok amplitudy jako liczba Q15 wynosi (*round* dokonuje zaokrąglenia):

  $d = \text{round}( f * 1{,}36533)$

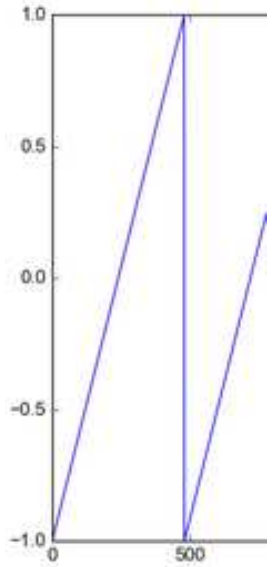
- Np. $f = 440$ Hz $\rightarrow d = 601$
- Jeżeli potrzebujemy obliczyć ten krok w kodzie:

$$d = f\frac{65536}{48000} = f\frac{2 \cdot 22368}{32768} = (f * 22368) >> 14$$

- Błędy zaokrąglenia powodują zniekształcenia, szczególnie dla małych częstotliwości.

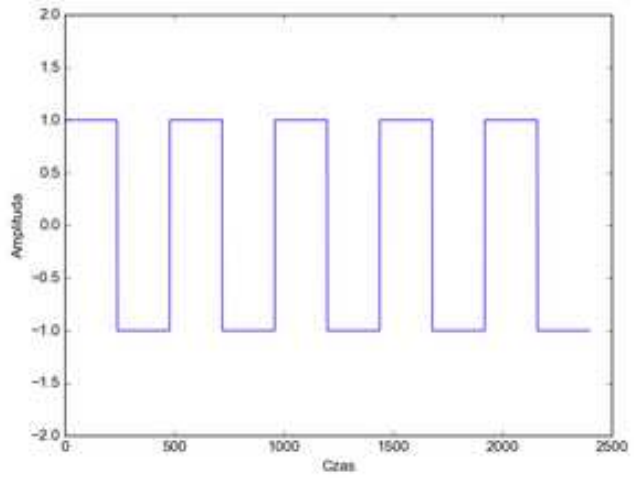
# Przepełnienie przy generowaniu

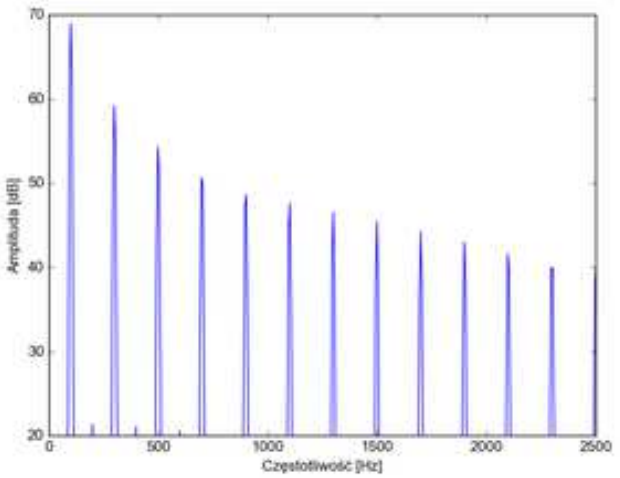
- Ważna uwaga: przy dodawaniu kroku wystąpi oczywiście przepełnienie, np:
  32750 + 25 = „32775” = –32761
- Amplituda „zawija się” na wartości ujemne
  - właśnie o to nam chodzi!
- Jest to jeden z nielicznych przypadków, w których wykorzystujemy efekt przepełnienia do własnych celów.



# Sygnał prostokątny / impulsowy

- Inny przykład sygnału harmonicznego: sygnał prostokątny (*square wave*) lub impulsowy (*pulse*).
- Sygnał ma tylko dwie wartości: -*A* i +*A*.
- Szerokość impulsu (*pulse width*): proporcja długości części dodatniej do całego okresu (0 do 1).



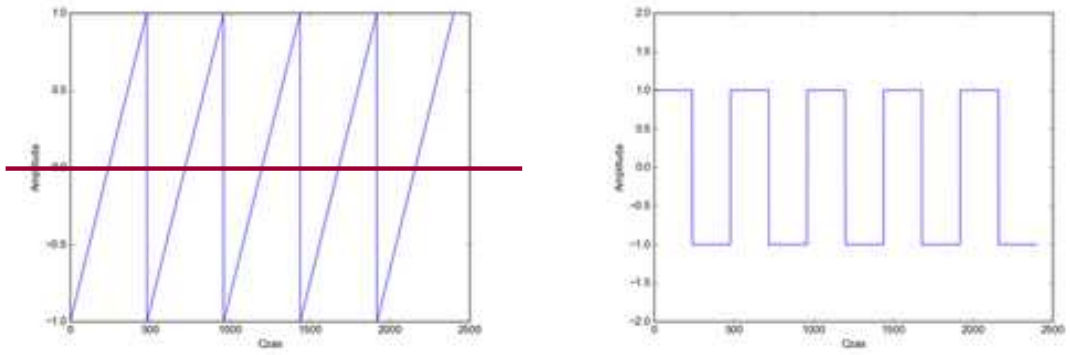



# Sygnał prostokątny / impulsowy

- Sygnał prostokątny można uzyskać z piłokształtnego:

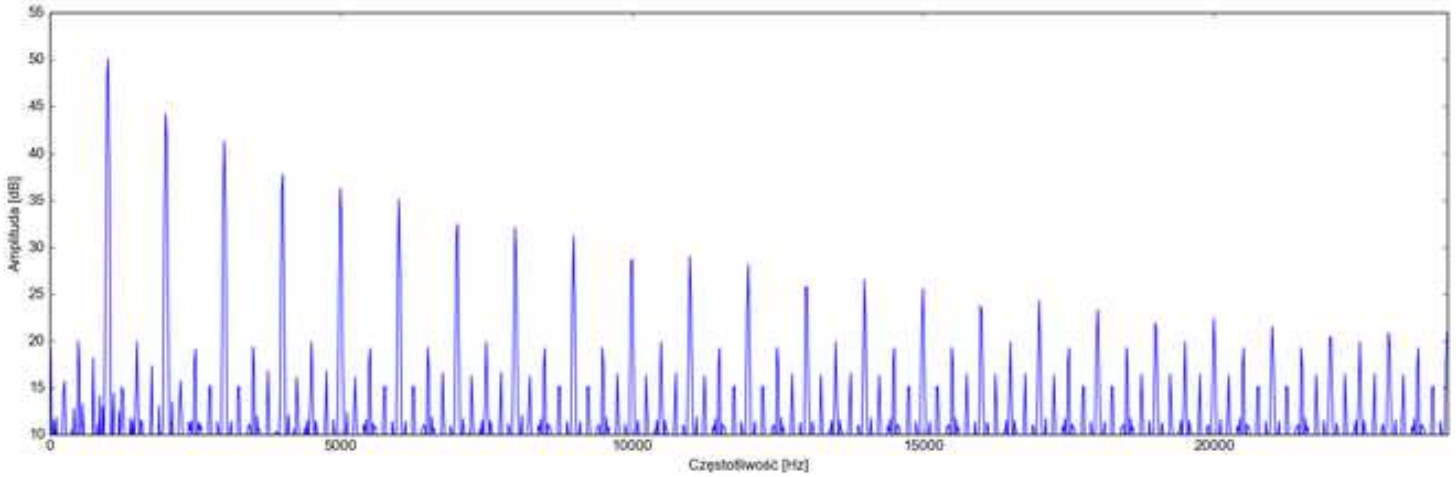
```
if (amplituda < prog)
   y = 32767;   // lub inna wartość amplitudy
else
   y = -32768;
```

- Wartość *prog* zależy od szerokości impulsu:
  *prog* = 2 * *szerokosc* – 1



# Problem aliasingu

- Sygnały harmoniczne mają nieskończone widmo.
- Generując sygnały harmoniczne „z definicji” wytworzymy sygnały zawierające alias widma.
- Problem wystąpi szczególnie dla większych cz.
- Powstanie sygnał nieharmoniczny.



# Problem aliasingu

Są różne metody radzenia sobie z aliasingiem.

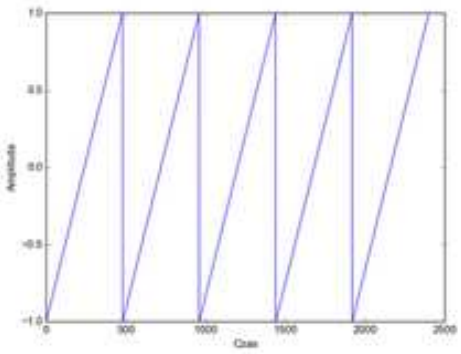
- Generowanie sygnału z większą częstotliwością próbkowania (nadpróbkowanie) – omówimy na kolejnym wykładzie.
- Obliczanie z szeregu Fouriera, np. dla „piły”:

$$x(n) = \frac{A}{2} - \frac{A}{\pi} \sum_{k=1}^{N} (-1)^k \frac{\sin(2\pi k n f / fs)}{k}$$

  - dla $k \cdot f$ leżących poniżej cz. Nyquista,
  - zniekształcamy postać czasową sygnału

# Generowanie sinusa

- Charakterystyka fazowa sygnału sinusoidalnego:



Wiemy już jak wygenerować sygnał piłokształtny. Musimy teraz zamienić fazę na amplitudę.

- Obliczanie z szeregu Taylora:

$$\sin(x) \cong x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots$$

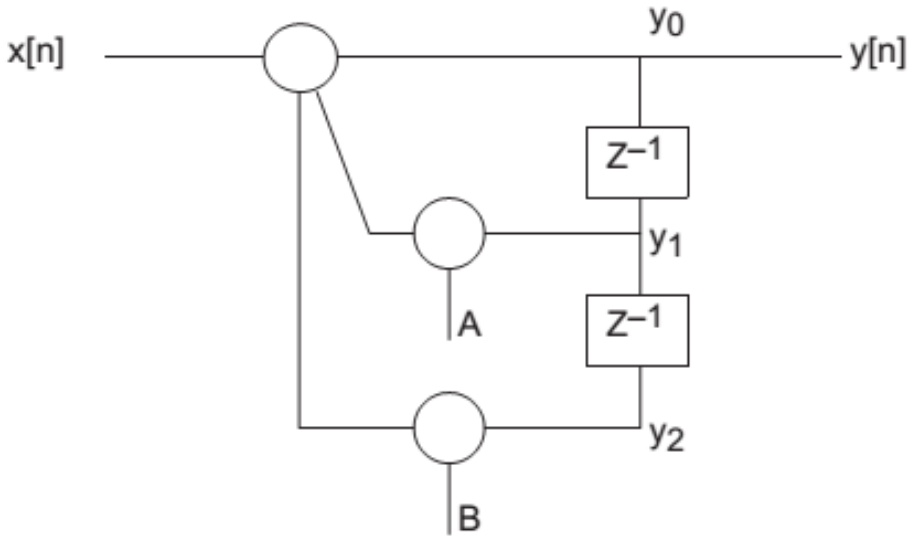
# Generowanie sinusa z DSPLIB

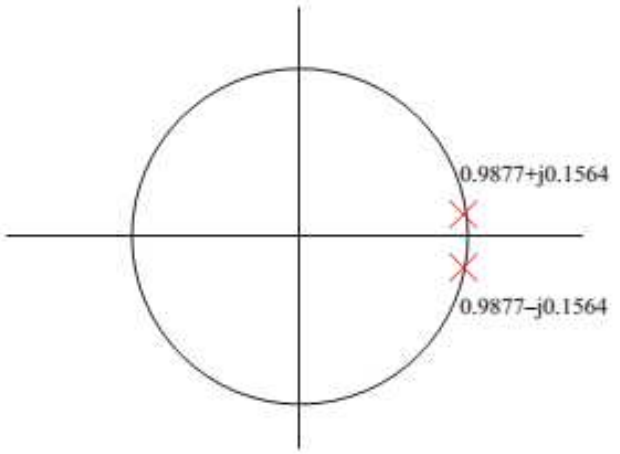
| sine | *Sine* |
| --- | --- |
| **Function** | ushort oflag = sine (DATA *x, DATA *r, ushort nx) |

- *x* – wskaźnik do bufora zawierającego wartości fazy. Tu wpisujemy wartości Q15 sygnału piłokształtnego o danej częstotliwości.
- *r* – wskaźnik do bufora, w którym zostaną zapisane wartości sinusa
- *nx*- liczba wartości w buforze

# Sinus z filtru IIR

- Alternatywna metoda generowania sinusa: stosujemy filtr IIR na granicy stabilności.



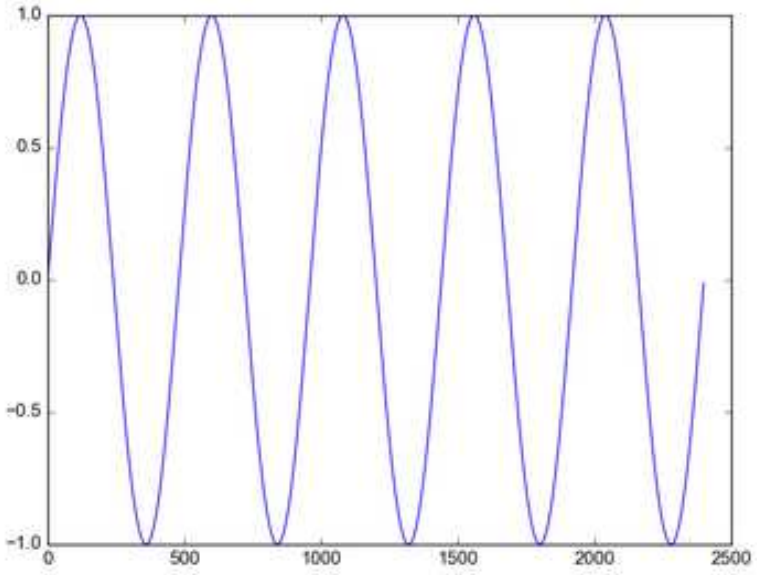




$$y(n) = a \cdot y(n-1) - y(n-2)$$

$$a = 2\cos\left(\frac{2\pi f}{fs}\right)$$

# Sinus z filtru IIR

- Pobudzamy filtr impulsem:
  $y(0) = -\sin(2\pi f/fs)$, $y(1) = 0$
- Filtr wpada w oscylacje – generuje próbki przy zerowym sygnale wejściowym.
- Implementacja na stałoprzecinkowym DSP jest problematyczna (powstają błędy)



# Generowanie szumu białego

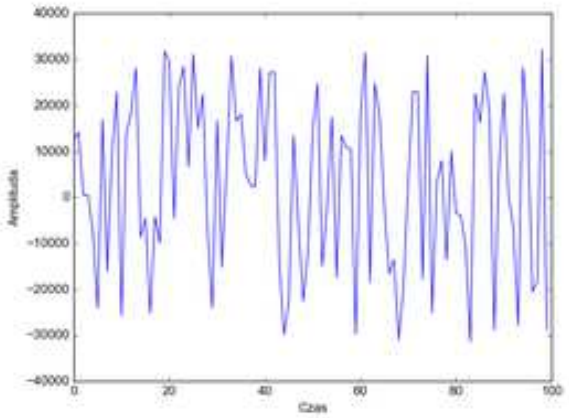
- Szum biały (*white noise*) – sygnał losowy o równomiernym rozkładzie widmowym.
- Do generowania cyfrowego szumu stosuje się generatory liczb pseudolosowych (*RNG*).
- Próbki są obliczane przez algorytm.
- LCG – liniowy generator kongruentny:

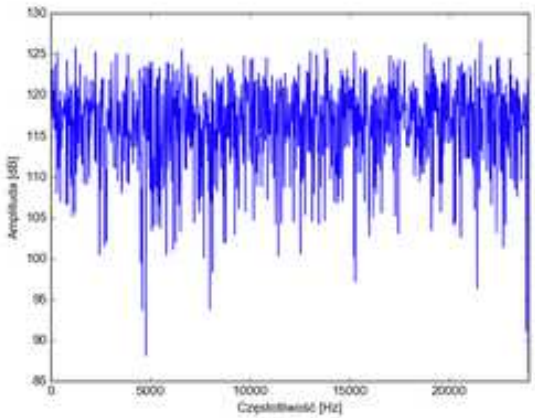
$$y(n) = \left[a \cdot y(n-1) + b\right] \bmod M$$

*mod* – modulo, reszta z dzielenia przez *M*

# Generowanie szumu białego

- Wartość początkowa $y(0)$ to „ziarno” (*seed*). Podając to samo ziarno, dostaniemy zawsze taką samą sekwencję liczb!
- W praktyce: ustawiamy ziarno na zmienną liczbę, zazwyczaj aktualny czas.
- Przykład: $a = 2045$, $b = 0$, $M = 2^{20}$, $y(0) = 12345$



# Generowanie szumu białego z DSPLIB

- Inicjalizacja – tylko raz na początku programu:

```
rand16init();
```

- Wypełnienie bufora *r* o długości *nr* próbkami:

| **rand16** | *Random Number Generation Algorithm* |
|---|---|
| **Function** | ushort oflag= rand16 (DATA *r, ushort nr) |

```
rand16(bufor, 2048);
```

- LCG: $a = 31821$, $b = 13849$, $M = 65536$

# Sygnał z tablicy próbek

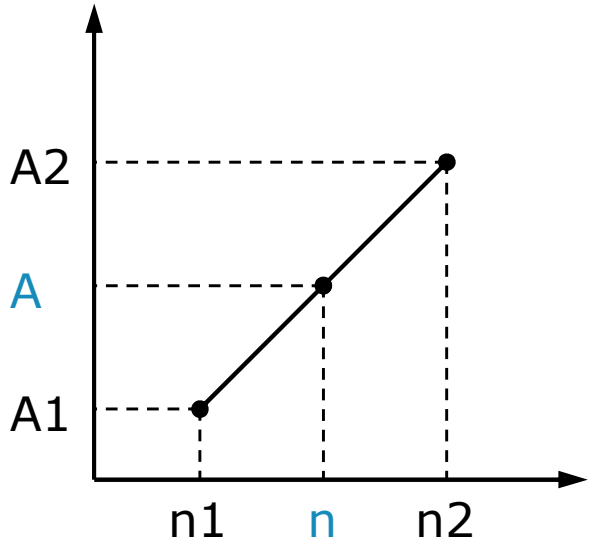
- Dowolny sygnał można wygenerować odczytując jego próbki zapisane w tablicy (*wavetable*).
- Np. możemy zapisać w tablicy 480 próbek okresu sygnału sinus.
- Odczytując je z prędkością 48 kHz dostaniemy sinus o częstotliwości 100 Hz.
- Jeżeli odczytamy tylko co drugą próbkę, mamy 240 próbek na okres, czyli $f$ = 48000 / 240 = 200 Hz
- Zapętlając odczyt dostajemy ciągły sygnał.

# Sygnał z tablicy próbek

- Przypadek ogólny: chcemy dostać dowolną cz.
- Krok, o jaki przesuwamy pozycję odczytu:
  $s = f \cdot N / fs$ ($N$ – liczba próbek w tablicy).
- Np. dla $f$ = 456 Hz i $N$ = 4800: $s$ = 45,6
- Zazwyczaj krok $s$ będzie niecałkowity.
- Musimy czytać „pomiędzy próbkami”.
- Interpolacja próbek: „zgadywanie” wartości pomiędzy zapisanymi próbkami sygnału.

# Interpolacja liniowa

- Najprostsza interpolacja: liniowa.
- Niech *indeks* = 45,6. Interpolujemy między próbkami *x*(45) i *x*(46) – „poprzednia” i „następna”.




$$\frac{A_2 - A_1}{n_2 - n_1} = \frac{A - A_1}{n - n_1} \qquad n_2 - n_1 = 1$$

$$A = A_1 + (A_2 - A_1)(n - n_1)$$

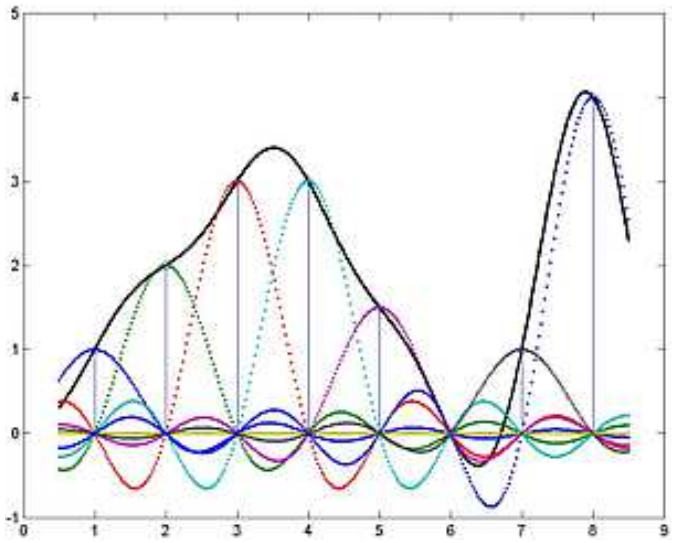
$$x(45{,}6) = x(45) + (x(46) - x(45)) \cdot 0{,}6$$

# Odczyt z tablicy z interpolacją

```
// Przykład: indeks = 45.6
int indeks_c = 45;              // część całkowita (45), int
int indeks_u = 19661;           // część ułamkowa (0.6), Q15

// Odczyt próbek z bufora
int a1 = bufor[indeks_c];     // próbka „poprzednia”
int a2 = bufor[indeks_c+1];   // próbka „następna”

// interpolacja liniowa
long a = (long)indeks_u * (a2 – a1)  // (n-n1)*(a2-a1)
a = (a >> 15) + a1                   // + a1

// wartość interpolowanej próbki
y = (int)a;
```

# Inne metody interpolacji

- Interpolacja liniowa jest prosta i mało dokładna.
- Bardziej złożone, ale dokładniejsze metody:
  - interpolacja wielomianowa – stopnia 2 (kwadratowa), 3 (kubiczna) i wyższych rzędów
  - interpolacja funkcją sin(x)/x („sinkowa”) - najczęściej używana w przetwarzaniu sygnałów



# Sygnał z tablicy próbek

Ogólne uwagi

- Im więcej próbek w buforze, tym lepiej (mniejsze błędy interpolacji).
- Możemy zapisać zupełnie dowolny sygnał.
- Duże wymagania pamięciowe.
- Metoda działa dobrze jeżeli odczytujemy i zapętlamy okres sygnału.
- Jeżeli nie zapętlamy, zmiana częstotliwości powoduje skrócenie lub wydłużenie sygnału.

# Sampler

- Przykład praktyczny: sampler – instrument muzyczny, który odgrywa próbki brzmień (sample) zapisane w pamięci. Zapętlane są tylko fragmenty w środku sampli, lub w ogóle nie zapętla się ich.
- DSP w samplerze dokonuje transpozycji – zmiany wysokości generowanego dźwięku poprzez zmienny krok odczytu próbek i interpolację.
- Powstają zniekształcenia czasowe – dźwięk skraca się lub wydłuża. Dlatego trzeba stosować zbiór sampli o różnych wysokościach.

# Podsumowanie

Co należy wiedzieć i umieć:

- jak wygenerować sinus i szum za pomocą DSPLIB,
- jak samodzielnie wygenerować sygnały harmoniczne, sinusa oraz szum,
- dlaczego w sygnałach harmonicznych powstaje aliasing i jak sobie z tym radzić,
- jak generować sygnały za pomocą tablicy próbek,
- jak użyć interpolacji do uzyskania dowolnej częstotliwości sygnału przy odczycie z tablicy.