GENEROWANIE SYGNAŁÓW

na procesorach sygnałowych

Opracowanie: Grzegorz Szwoch

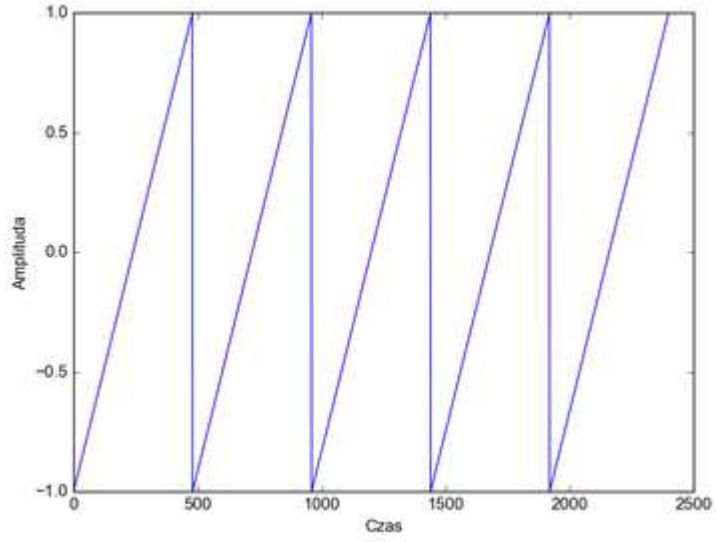
Politechnika Gdańska, Katedra Systemów Multimedialnych

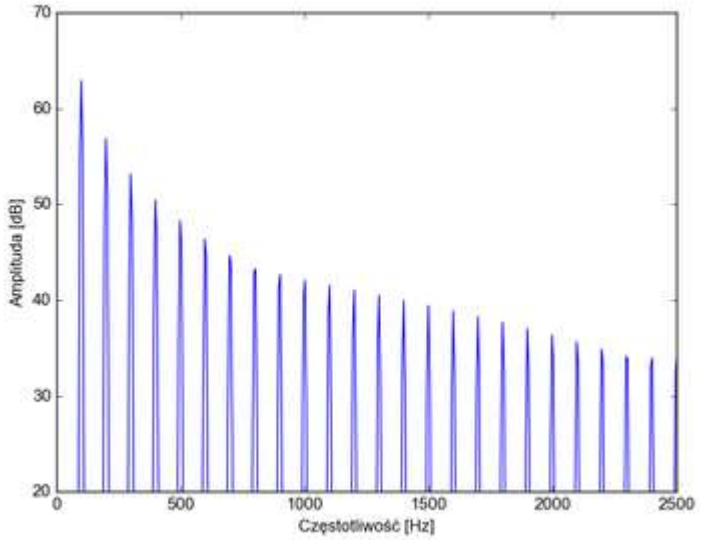
Wprowadzenie

- Procesory sygnałowe są zwykle stosowane do przetwarzania sygnałów podawanych na jego wejście.
- Możemy także wykorzystać procesor do generowania sygnałów – wejście nie jest wykorzystywane.
- Na tym wykładzie omówimy:
  - generowanie cyfrowych sygnałów harmonicznych,
  - generowanie sygnału sinusoidalnego różnymi metodami,
  - generowanie sygnału pseudoprzypadkowego,
  - generowanie dowolnych sygnałów z tablicy próbek,
  - interpolacje próbek zapisanych w tablicy.

Sygnał piłokształtny

- Sygnały harmoniczne: mają widmo, w którym występują składowe w szeregu harmonicznym, tzn. na wielokrotnościach częstotliwości podstawowej.
- Przykład: sygnał piłokształtny (*sawtooth*, „rampa”). Postać czasowa i widmowa:





Sygnał piłokształtny

- Amplituda sygnału zmienia się liniowo.
- Do wygenerowania sygnału użyjemy tzw. akumulatora – sumujemy kolejne wartości.
- Inicjalizacja:

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

- Dla każdej próbki, wyjście *y*:

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

- Ile wynosi *krok*?

Obliczenie kroku amplitudy

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

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

- A jeżeli chcemy częstotliwość 100 Hz (okres 0,01 s)?

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

Obliczenie kroku amplitudy

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

  $d = \text{round}(f * 1.36533)$

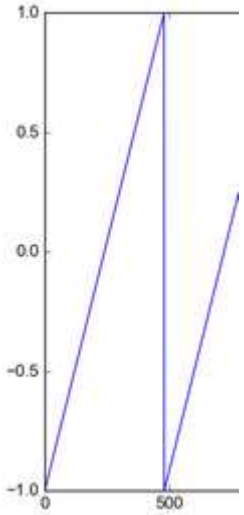
- Np. $f = 440$ Hz $\rightarrow d = 601$
- Jeżeli potrzebujemy obliczyć ten krok w kodzie:

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

- Pamiętajmy, że nie możemy dokładnie zapisać dowolnej częstotliwości stosując format stałoprzecinkowy.

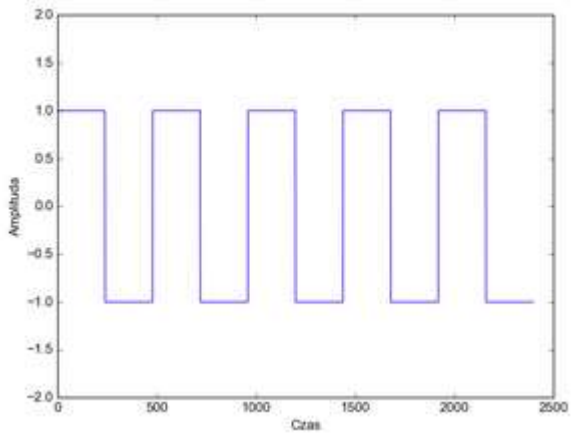
Przepełnienie przy generowaniu

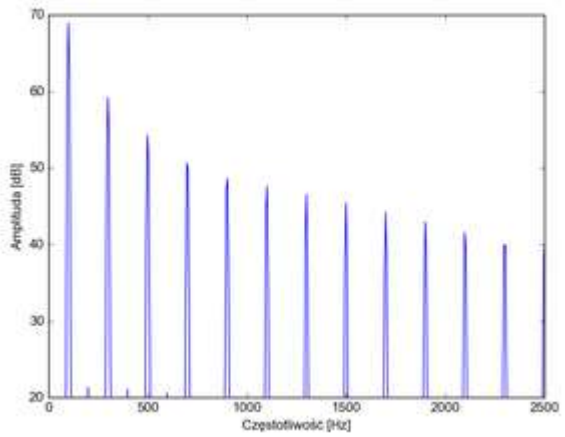
- Ważna uwaga: przy dodawaniu kroku wystąpi oczywiście przepełnienie zakresu, np.:
  32750 + 25 = „32775” = –32761
- Amplituda „zawija się” na wartości ujemne – właśnie o to nam chodzi!
- Jest to jeden z nielicznych przypadków, w których efekt przepełnienia jest korzystny.



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).
- Postać czasowa i widmowa dla szerokości 0,5:



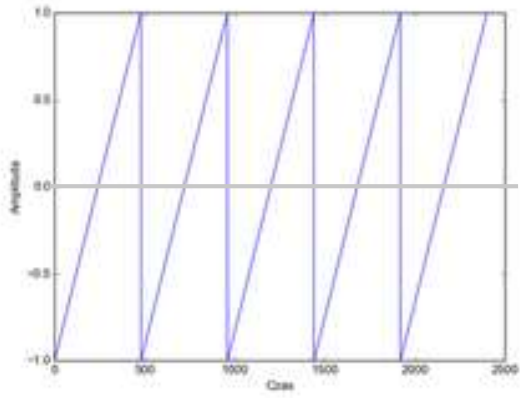


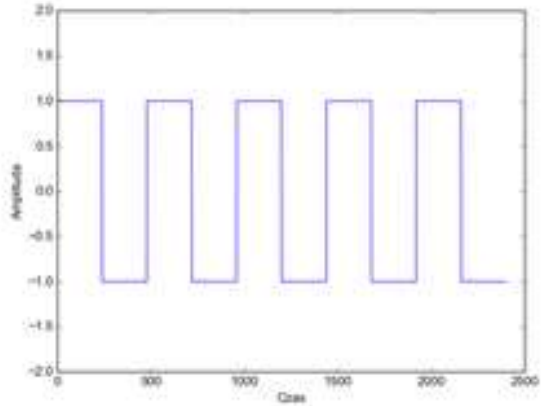
Sygnał prostokątny / impulsowy

- Sygnał prostokątny można uzyskać z piłokształtnego:

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

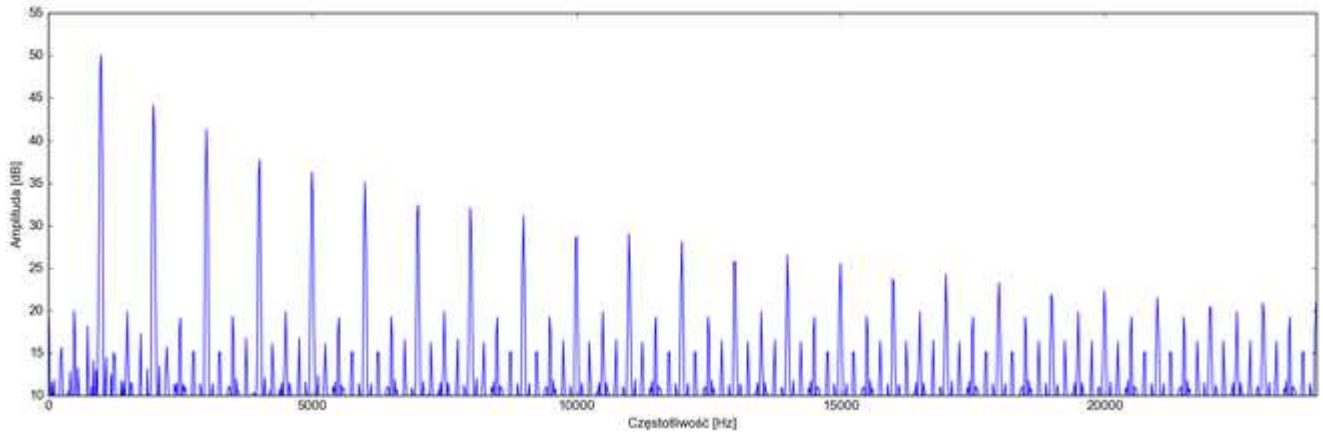
- Wartość *prog* zależy od szerokości impulsu:
  prog = 2 * *szerokosc* – 1
- „Klasyczny” sygnał prostokątny (proporcje 50/50): próg = 0.





Problem aliasingu

- Sygnały harmoniczne (analogowe) , takie jak prostokątny lub piłokształtny, mają nieskończone widmo.
- Generując cyfrowo sygnały harmoniczne „z definicji” spowodujemy aliasing widma.
- Problem wystąpi szczególnie dla większych częstotliwości.
- Powstanie sygnał nieharmoniczny.



Problem aliasingu

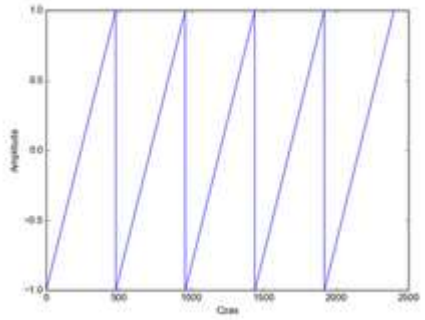
Są różne metody radzenia sobie z aliasingiem przy generowaniu.

- Generowanie sygnału z większą częstotliwością próbkowania (nadpróbkowanie), a następnie decymacja.
- 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 częstotliwości Nyquista,
  - zniekształcamy postać czasową sygnału (brak składowych o wysokiej częstotliwości).

Generowanie sinusa



- Charakterystyka fazowa sygnału sinusoidalnego ma postać taką, jak sygnał piłokształtny.
- Wiemy już jak wygenerować sygnał piłokształtny. Musimy teraz zamienić fazę na amplitudę.
- Jedna z możliwości: aproksymacja funkcji *sin* za pomocą szeregu Taylora:

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

- Funkcja *sine* z DSPLIB dla C55x wykorzystuje tę metodę.

Generowanie sinusa za pomocą DSPLIB

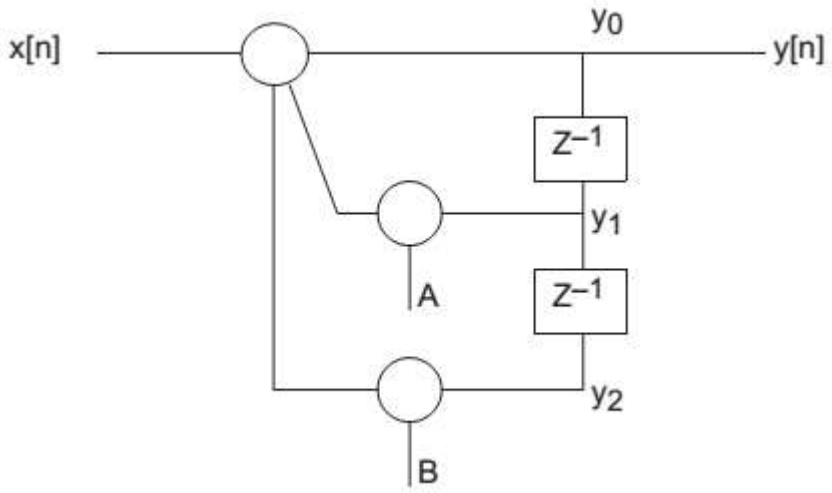
sine	*Sine*
Function	ushort oflag = sine (DATA *x, DATA *r, ushort nx)

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

Uwaga: funkcja *sine* sama z siebie nie wygeneruje sygnału sinusoidalnego, ona wyłącznie oblicza sinus podanego argumentu (kąta fazowego).

Sinus z układu IIR

- Alternatywna metoda generowania sinusa: stosujemy układ IIR 2. rzędu na granicy stabilności.
- Pobudzamy filtr impulsem: $y(0) = -\sin(2\pi f/fs)$, $y(1) = 0$
- Układ wchodzi w oscylacje – generuje wartości przy zerowym sygnale wejściowym.
- Implementacja na stałoprzecinkowym DSP jest problematyczna (powstają błędy)



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

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

Generowanie szumu białego

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

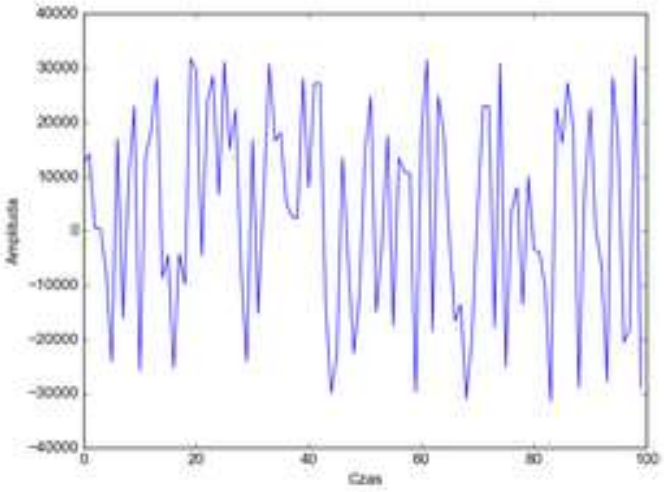
$$y(n) = \left[a \cdot y(n-1) + b\right] \bmod M$$

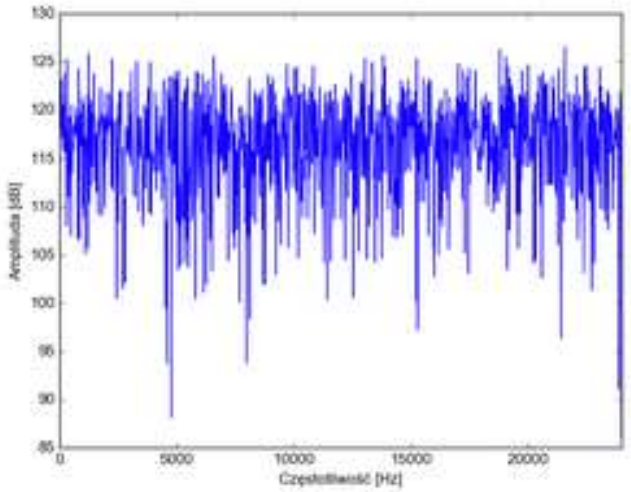
mod – modulo, reszta z dzielenia przez *M*

- Do poważnych zastosowań (np. szyfrowanie danych) wymagane są dokładniejsze metody (np. Mersenne Twister).

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 z zegara systemowego.
- Przykład: $a = 2045$, $b = 0$, $M = 2^{20}$, $y(0) = 12345$.
  Postać czasowa i widmowa:





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)

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

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

Sygnał z tablicy próbek

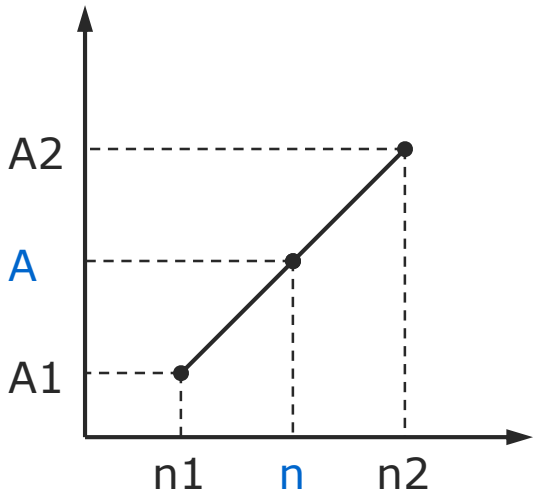
- Dowolny sygnał można wygenerować odczytując jego próbki zapisane w tablicy (*wavetable*).
- Np. możemy zapisać w tablicy 480 próbek okresu sygnału sinusoidalnego.
- Odczytując je z prędkością 48 kHz dostaniemy sygnał sinusoidalny o częstotliwości 100 Hz.
- Jeżeli odczytamy tylko co drugą próbkę, mamy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

Sygnał z tablicy próbek

- Przypadek ogólny: chcemy dostać dowolną częstotliwość.
- 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 nie będzie liczbą całkowitą.
- Musimy czytać „pomiędzy próbkami”.
- Interpolacja próbek: „zgadywanie” wartości pomiędzy zapisanymi w tablicy próbkami sygnału.

Interpolacja liniowa

- Najprostsza interpolacja: liniowa. Łączymy znane próbki linią prostą i szukamy wartości w zadanym punkcie na linii.
- Niech *indeks* = 45,6. Interpolujemy między próbkami *x*[45] i *x*[46] – „poprzednia” i „następna”.



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

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

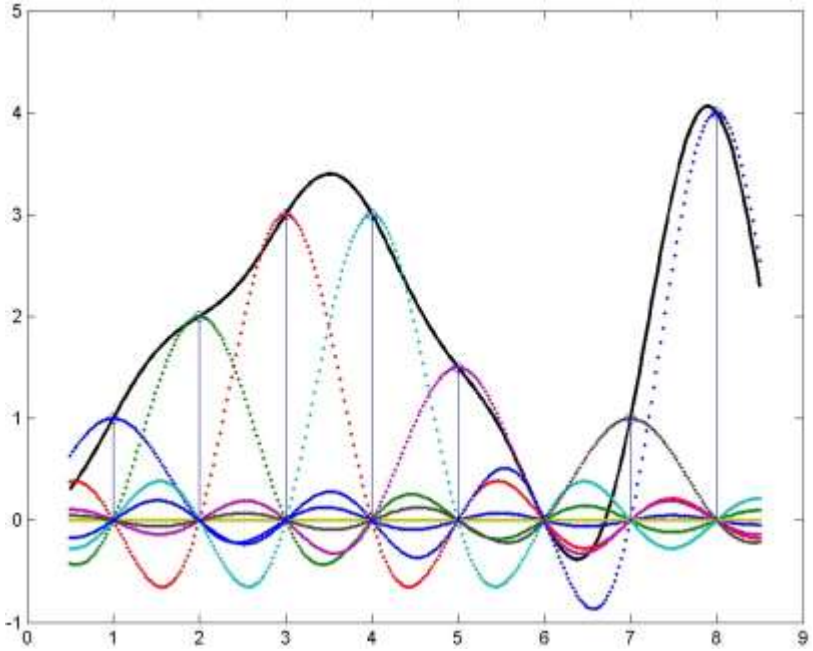
$$x[45{,}6] = x[45] + (x[46] - x[45]) \cdot 0{,}6$$

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 = indeks_u * (long)(a2 – a1)  // (n-n1)*(a2-a1)
a = (_sround(a<<1) >> 16) + a1       // + a1

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

Inne metody interpolacji

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



Sygnał z tablicy próbek

- Im więcej próbek w buforze, tym lepiej (mniejsze błędy interpolacji).
- Możemy zapisać zupełnie dowolny sygnał.
- Metoda działa dobrze jeżeli odczytujemy i zapętlamy okres sygnału.
- Jeżeli nie zapętlamy, zmiana częstotliwości powoduje skrócenie lub wydłużenie sygnału.
- Interpolacja sygnałów o złożonym widmie (np. prostokątny) może spowodować aliasing. Trzeba zwykle stosować wiele wersji sygnału o różnych częstotliwościach, zapisanych w tablicy.

Sygnał z tablicy próbek

Metody odczytu próbek dźwięku zapisanego w tablicy:

- krok odczytu = 1:
  - próbki czytane z częstotliwością próbkowania,
  - wysokość dźwięku – taka, jaką miał oryginalny dźwięk;
- krok odczytu < 1:
  - czytamy próbki wolniej – dźwięk się wydłuża,
  - dźwięk staje się niższy;
- krok odczytu > 1:
  - czytamy próbki szybciej – dźwięk się skraca,
  - dźwięk staje się wyższy.

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.
- Procesor sygnałowy 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 (multisampling).