JĘZYK PYTHON

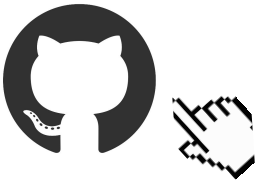
Projekt: Gitarowy stroik częstotliwościowy

Autor: Artur Guniewicz

Opiekun: dr hab. Andrzej Kapanowski

08.02.2021 r.

Github



Wstęp

Jest to projekt, w którym spróbuję wytłumaczyć działanie gitarowego stroika częstotliwościowego, a następnie zaprogramować jego działającą wersję w języku Python. Zaczniemy od naszego problemu, czyli rozstrojonej gitary, a następnie, poprzez teorię muzyki oraz matematykę i algorytmikę, zajmiemy się jego rozwiązaniem.

Podstawowe pojęcia

Zacznijmy jednak od bardzo podstawowych pojęć.

Gitara (z reguły) ma sześć strun opartych na dwóch elementach: siodełku i mostku. Zwieńczeniem szyjki gitary jest główka, na której znajdują się klucze do strojenia. Kiedy pociągnie się za strunę, otrzyma się dźwięk. Można go usłyszeć dlatego, że struna wibruje z określoną częstotliwością. Pitch to odbierana w danej chwili częstotliwość. Na potrzebę tej dokumentacji będę używał terminu "pitch" wymiennie z pojęciem "wysokość dźwięku". Ostatnim pojęciem będzie nuta; jest to wysokość dźwięku, której nadaliśmy nazwę.

Jak działa gitara?

Mając te definicje w głowie spójrzmy teraz na to jak gitara działa na poziomie muzycznym.

Struny zazwyczaj są nastrojone do standardowych dźwięków EADGBE, gdzie każda nuta odnosi się do jednej ze strun. Na przykład, najniższa struna jest nastrojona do nuty E_2. Znaczy to, że struna ta ma wysokości na poziomie 82.41 Hz i od teraz ton E_2 jest zdefiniowany. Jeżeli więc struna miałaby wysokość mniejszą niż zadana, na przykład 81 Hz znaczyłoby to, że nasza gitara jest rozstrojona i musielibyśmy użyć kluczy na główce, żeby przywrócić jej poprawne brzmienie. Wszystkie inne struny także mogą być przypisane do pewnej wysokości, a co za tym idzie do częstotliwości drgań.

Jak nastroić gitarę?

Jest to projekt informatyczny więc przejdźmy płynnie do tego jak nastroić gitarę używając wiedzy z zakresu muzyki, matematyki i programowania. Zatem aby nastroić gitarę będziemy potrzebować: gitary, komputera z podłączonym bądź wbudowanym mikrofonem oraz Pythona.

Istotne jest to, że zakładamy tak zwany system równomiernie temperowany, czyli strój muzyczny, w którym stosunek częstotliwości dwóch kolejnych dźwięków w dwunastotonowym systemie wynosi $\sqrt[12]{2}$, gdyż system ten zakłada podział oktawy na 12 równych części. Zakładamy także wysokość nuty A_4 na poziomie 440 Hz, co pozwala na pokrycie prawdopodobnie około 99% współczesnej muzyki.

Dzięki tym założeniom możemy wyprowadzić następujący wzór, który definiuje nuty i wysokości dźwięków w postępie półkrokowym:

$$f_i = f_0 \cdot 2^{\frac{i}{12}}$$

Na przykład, jeżeli mamy wysokość f_0 odpowiadającą nucie A_4, czyli 440 Hz i chcemy ją zwiększyć o pół kroku do nuty $A\#_4$, to musimy pomnożyć 440 Hz przez $2^{\frac{1}{12}}$ co w rezultacie da nam częstotliwość 466.16 Hz.

Możemy również otrzymać odwrotność tego wzoru, która mówi nam ile półkroków jest pomiędzy badaną wysokością f_i i wysokością odniesienia f_0:

$$12 \cdot log_2(\frac{f_i}{f_0}) = i$$

To pozwala nam przypisać danej wysokości konkretną nutę. Jak można się domyślić ten wzór będzie miał dla nas specjalne znaczenie, ponieważ jeżeli możemy wyciągnąć wysokość dźwięku z nagrania, chcemy wiedzieć jaka jest najbliższa nuta i jak daleko się ona znajduje.

Znalezienie najbliższej nuty

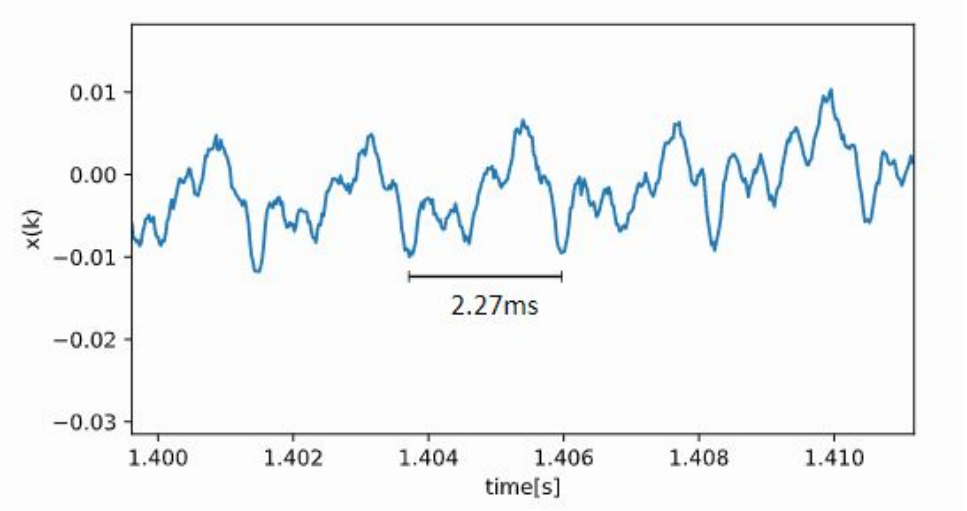
Wszystko to prowadzi nas do kodu zawartego w pliku *find_closest_note.py*. Mamy tutaj funkcję *find_closest_note(pitch)*. Jeżeli wprowadzimy do niej wysokość dźwięku w Hz, zwróci ona najbliższą nutę oraz odpowiadającą jej wysokość.

Najpierw obliczana jest liczba półkroków między nutą A_4, a rozważaną wysokością dźwięku. Następnie liczba ta jest używana do wybrania z tabeli nut tej odpowiedniej, razem z numerem oznaczającym właściwą oktawę oraz do wyznaczenia wysokości tej nuty. Ważne jest to, że nuty w oktawie zaczynają się od nuty C (stąd: $\frac{9 + abs(i)}{12}$).

Wykrycie wysokości dźwięku

Naszym następnym krokiem będzie nagranie gitary i zdefiniowanie wysokości tego nagrania. Skupmy się więc na wykryciu wysokości danego dźwięku. Dźwięk ten musimy jakoś nagrać i gdzieś zapisać. Służyć do tego będzie kod Pythona znajdujący się w pliku *recording.py*. Musimy zaimportować biblioteki *sounddevice* do nagrywania dźwięku, *scipy.io.wavfile* do zapisu dźwięku w formacie *.wav* oraz bibliotekę do obsługi czasu. Ustalamy częstotliwość nagrania na 44100 Hz i czas jego trwania na 2 sekundy. W odpowiednim momencie włączy się nagrywanie, następnie program odtworzy to co nagraliśmy i zapisze na dysku pod wskazaną nazwą.

Python dzięki funkcjonalności *pyplot* umożliwia rysowanie wykresów. Kod z pliku *signal_time_plot.py* używa wcześniej przez nas nagranego dźwięku i tworzy jego wykres.



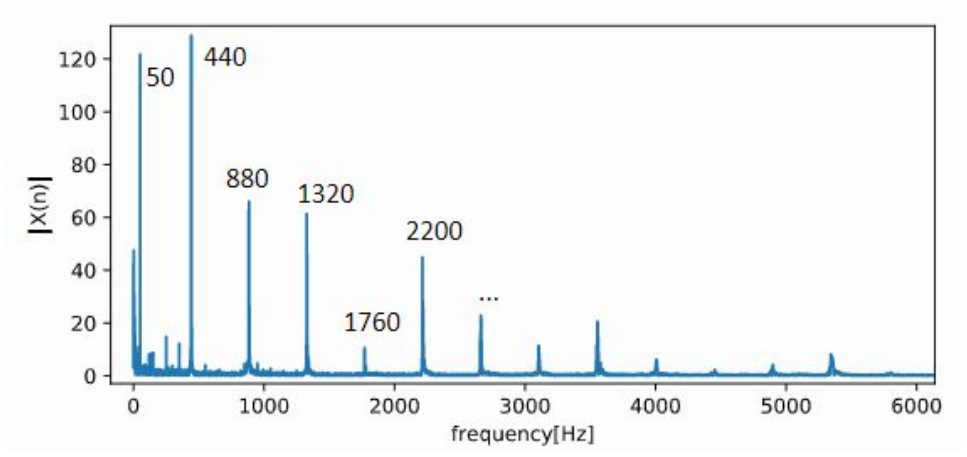
Stworzyliśmy więc wizualizację wysokości dźwięku i przestawiliśmy go jako wykres czasu i wartości. Jak widać sygnał ma okres długości około 2.27 ms co przekłada się na częstotliwość 440 Hz. Jak na razie wygląda to dobrze, ale sygnał ten daleki jest od bycia tak zwanym czystym tonem czyli dźwiękiem, którego przebieg ma postać sinusoidalną.

Dyskretna Transformacja Fouriera (DTF)

Będziemy potrzebować pojęcia Dyskretnej Transformacji Fouriera, w skrócie (DTF). Z matematycznego punktu widzenia pokazuje ona jak dyskretny sygnał może być rozłożony na zbiór funkcji cosinusowych oscylujących na różnych częstotliwościach.

Albo z muzycznego punktu widzenia: DTF pokazuje, które czyste tony znajdują się w nagraniu.

Fajną sprawą jest to, że DTF dostarcza nam tak zwane widmo wielkości. Dla dźwięku, którego użyliśmy wcześniej wygląda ono tak.



Na osi X widać częstotliwości czystych tonów, natomiast na osi Y zaznaczona jest ich intensywność.

Wykres ten otrzymałem wprowadzając nagranie do kodu z pliku *DFT_plot.py*. Używa on i jego i właściwości Dyskretnej Transformacji Fouriera, ale nie będę się już w to dalej zagłębiał. Skupmy się na tym co możemy z tego wykresu wyczytać.

Tak jak oczekiwaliśmy największą intensywnością charakteryzuje się czysty ton 440 Hz. Ale dalej mamy inne znaczące piki w wielokrotnościach 440 Hz. Na przykład 880 Hz, 1320 Hz i tak dalej. Osoby zaznajomione z muzyką mogą znać nazwy tych pików: są to harmonie lub overtony.

Powód powstawania overtonów jest dość prosty. Kiedy uderza się strunę gitary wprowadza się ją w wibrację o określonej częstotliwości. Zwłaszcza takie częstotliwości, które tworzą utrzymujące się fale, które mogą wibrować przez długi czas. Struna jednak nie może wibrować w punktach, w których przymocowana jest do gitary, czyli na mostku i na siodełku. W ten sposób wzbudzanych jest wiele overtonów, które są wielokrotnościami częstotliwości podstawowej. Ogólny zestaw harmonii i tego jak są ze sobą powiązane nazywamy tembrem. Tembr sprawia, że gitara brzmi jak gitara. Z jednej strony jest to całkiem fajne, z drugiej zaś sprawia że wykrycie odpowiedniej wysokości jest ciężkim zadaniem.

Najprostszym pomysłem na stworzenie stroika do gitary mogłoby być: stworzenie widma DTF, określenie częstotliwości najwyższego piku, koniec. Niestety to nie jest tak prosta sprawa, istnieje wiele przypadków, dla których w ten sposób uzyska się złe rezultaty.

Po pierwsze, podstawowa częstotliwość nie zawsze da najwyższy pik. Z tego powodu wykrycie wysokości dźwięku nie jest tylko wykryciem częstotliwości. Druga sprawa jest taka, że dźwięk gitary jest rozprowadzany w dużym paśmie częstotliwości. Algorytm wybierający tylko najwyższy pik byłby bardzo podatny na tak zwany szum szerokopasmowy. W podanym wcześniej przykładzie widać wysoki pik przy 50Hz, jest on spowodowany szumem sieci.

Założenia naszego stroika

Złożoność tego problemu doprowadziła do powstania wielu algorytmów wykrywania wysokości dźwięku. Aby wybrać ten odpowiedni musimy zastanowić się nad wymaganiami jakie stroik do gitary musi spełnić. Najważniejszymi wymaganiami na pewno są:

- **Dokładność**

  Zauważalna różnica dla złożonych tonów poniżej 1000 Hz to 1 Hz, więc naszym celem będzie osiągnięcie rozdzielczości częstotliwości 1 Hz w zakresie około 80-400 Hz

- **Możliwość pracy w czasie rzeczywistym**

  Używając stroika chcemy dostawać na bieżąco feedback, którą nutę właśnie gramy

- **Opóźnienie**

  Jeżeli wynik pokaże się nam po 5 sekundach po tym jak zagramy jakiś dźwięk strojenie gitary będzie bardzo ciężkie i uciążliwe. Zakładam, że opóźnienie rzędu 500 ms będzie sprawdzało się całkiem dobrze

Prosty stroik DTF

Pokażę teraz jak zaprogramować w Pythonie prosty stroik opierający się na wykrywaniu maksymalnej częstotliwości. Jak wcześniej zaznaczyłem metoda ta może nie zadziałać aż tak dobrze, przez to, że podstawowa częstotliwość nie zawsze musi mieć najwyższy pik, jednak metoda ta jest dość prosta do zrozumienia i stanowi niezłe wprowadzenie do dalszego rozwoju. Aby stworzyć dokładny stroik do gitary należałoby użyć złożonego algorytmu stosującego Widmo Produktów Harmonicznych. Jednak nie jest to tematem tego projektu.

Skupmy się zatem na prostym stroiku opartym na wykrywaniu pików DTF. Zazwyczaj algorytm DTF jest stosowany do całego czasu trwania sygnału. Jednak z założeń wynika, że nasz stroik ma być aplikacją działającą w czasie rzeczywistym, czyli nie ma miejsca na koncept „całego sygnału". Co więcej, jak będziemy grać różne nuty tylko kilka ostatnich sekund jest istotne do wykrycia wysokości dźwięku. Więc użyjemy tak zwanej Dyskretnej Krótkotrwałej Transformacji Fouriera, która jest właściwie tym samym co Dyskretna

Transformacja Fouriera, ale dotycząca najnowszych próbek. Można to sobie wyobrazić jako swego rodzaju okno, w którym nowe próbki wypychają najstarsze próbki.

Przed tym jak zaczniemy programować prosty stroik musimy się zastanowić nad rozwiązaniami projektowymi dotyczącymi algorytmu DTF. Bo czy w ogóle DTF może spełnić wymagania, które wcześniej sobie założyliśmy?

Zacznijmy od zakresu częstotliwości. DTF pozwala na analizę częstotliwości w przedziale $f < \frac{f_s}{2}$, gdzie f_s jest częstotliwością próbkowania. Typowe urządzenia nagrywające używają częstotliwości próbkowania około 40 kHz, co daje nam zakres częstotliwości $f < 20\,kHz$. To więcej niż wystarczająco, żeby złapać wszystkie overtony.

Należy zauważyć, że zakres częstotliwości jest nieodłączną właściwością algorytmu DTF, ale istnieje również ścisły związek z twierdzeniem o próbkowaniu, które mówi, że nie można wydobyć wszystkich informacji z sygnału, jeśli najwyższe występujące częstotliwości są większe niż $\frac{f_s}{2}$. Oznacza to, że DTF działa już na teoretycznej granicy.

Następnie trzeba się skupić na rozdzielczości częstotliwości DTF która przedstawia się tak:

$$\frac{f_s}{N} \approx \frac{1}{t_{window}}[Hz]$$

Gdzie N to rozmiar okna w próbkach, a t_{window} to rozmiar okna w sekundach. Rozdzielczość w Hz to w przybliżeniu odwrotność rozmiaru okna w sekundach. Więc jeśli mamy okno o rozmiarze 500 ms, to nasza rozdzielczość częstotliwości to 2 Hz. I tutaj zaczyna się robić podchwytliwie, ponieważ im większe okno tym lepsza rozdzielczość częstotliwości, ale tym większe opóźnienie. Jeśli weźmiemy pod uwagę rozdzielczość częstotliwości, do pewnego stopnia ważniejszą niż opóźnienie, rozmiar okna 1 s wydaje się dobrym wyborem. Przy takim ustawieniu uzyskujemy rozdzielczość częstotliwości 1 Hz.

Kod Python

Przeanalizujmy kod z pliku *tuner.py*.

- Linia 1-4: To są podstawowe importy takie jak numpy do działań matematycznych i sounddevice do przechwytywania wejścia mikrofonu
- Linia 6-11: Są to zmienne globalne, których użyjemy później w kodzie. Ustawiamy częstotliwość próbkowania na 44,1 kHZ, taki też rozmiar okna i dalej implementujemy rozważania, o których wspominałem wcześniej
- Linia 13-25: Jest to wcześniej wspomniana funkcja do znajdywania najbliższej nuty dla zadanej wysokości
- Linia 28-47: To jest serce naszego kodu, więc przyjrzyjmy mu się bliżej
- Linia 33-34: Tutaj przychodzące próbki są dołączane do tablicy, a stare próbki są z niej usuwane
- Linia 35: Widmo wielkości jest uzyskiwane za pomocą Szybkiej Transformacji Fouriera
- Linia 37-38: Tutaj szum sieciowy jest niwelowany poprzez ustawienie wszystkich częstotliwości poniżej 62Hz na 0
- Linia 40-42: Najpierw najwyższy pik częstotliwości jest określany, a potem, jako następny krok, najwyższe częstotliwości są użyte do otrzymania najbliższej wysokości oraz nuty
- Linia 44-47: Ten fragment wyświetla wynik na ekran
- Linia 50-57: Tutaj strumień wejściowy jest inicjalizowany i uruchamiana jest nieskończona pętla. Po pobraniu wystarczającej ilości danych funkcja callback, czyli funkcja zwrotna jest wywoływana

Podsumowanie

Jeżeli spróbuje się nastroić swoją gitarę używając tego kodu może nie brzmieć za dobrze. Tak jak można było przewidzieć największym problemem są błędy harmoniczne, ponieważ overtony są często bardziej intensywne niż częstotliwość podstawowa. Niemniej jednak stroik działa i da się za jego pomocą rozwiązać problem rozstrojonej gitary w stopniu satysfakcjonującym.