2C-02

音楽情報処理ライブラリ Pytakt の設計

西村　憲†　　丸井　淳史‡

会津大学コンピュータ理工学部†　　東京藝術大学音楽学部音楽環境創造科‡

1. はじめに

近年、ニューラルネットワークなど機械学習を利用した音楽情報分野における研究が盛んである。機械学習の研究では Python を用いることが多いことから、Python 用の音楽情報処理ライブラリへの需要が高まっている。

そのようなライブラリとして、既に music21 [1], PrettyMIDI [2], mido [3] などが知られている。music21 は、MusicXML との相性が良く、五線譜ベースの音楽情報についての処理・解析に適している。しかし、人間の演奏を記録した MIDI データのように時刻が不規則な音楽情報を扱うには困難を伴い、また、MIDI コントロールチェンジ等を含めた処理・解析はできない。PrettyMIDI は、MIDI データを音符リストや時間フレームごとの発音状況のデータに変換する機能を持ち、時刻が秒で表現されているため、音響信号との同期をとりやすい。しかし、逆に拍単位の処理を行うには特別な工夫が必要となる。また、音楽生成の機能はごく基本的なものに留まっている。mido は、MIDI ファイルの読み書き及び MIDI 入出力についてのライブラリであり、単純で分かりやすい。しかし、音符データを常にノート・オンとノート・オフに分けて処理するため、例えば音符ごとの音価の取得などは簡単に実現できない。

このような現状を踏まえて、Pytakt と名付けられた新たな Python 用音楽情報処理ライブラリを開発している。これは、過去に著者の 1 人が開発した Takt [4] と呼ばれる音楽言語ツールを基にしており、イベント単位での音楽生成、解析を目的としている。本稿では Pytakt の概要について述べる。

2. 設計コンセプト

Pytakt においては、共通したデータ構造によってリアルタイム MIDI 処理とオフライン解析の双方を可能とし、大規模スコアを扱えるように性能面で配慮をしつつ、使用頻度の高いものについてはできるだけ簡潔に処理を表現できることを目標とする。

図 1 にライブラリ全体の機能をデータの流れとともに示す。中心となるのは Pytakt スコアと呼ばれる音楽情報の内部表現であり、これを通して、標準 MIDI ファイルや独自の JSON 形式ファイルの読み書き、ピアノロール表示、music21 を経由しての MusicXML への変換が可能である。また、後述の MML や手続きによってスコアを生成できる。以下、Pytakt の主な特徴について述べる。

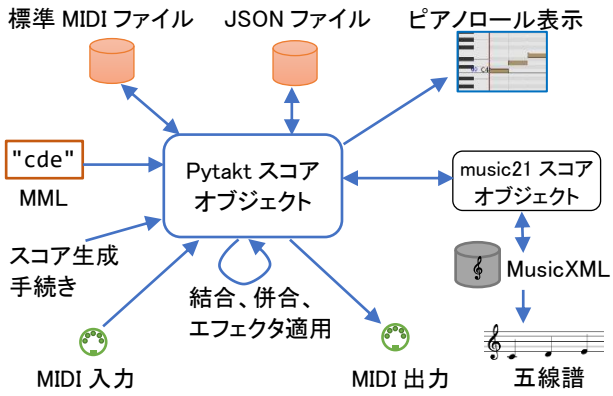

図 1．Pytakt におけるデータの流れ

ピッチと音程

ピッチは MIDI ノート番号を基本とするが、異名同音の区別を可能にするため、整数クラスを継承した Pitch クラスを設ける。このクラスのオブジェクトは、MIDI ノート番号を表す整数として振る舞うが、異名同音の情報を持っているため、例えば文字列に変換したときには、同じ整数値でも異なる結果を与えうる。

Pitch オブジェクトどうしの差は、Interval クラスのオブジェクトとなる。これは半音数を表す整数として振る舞うが、付随情報として五線譜上の符号付き距離（いわゆる度数に相当）を持ち、異名同音を正しく保った移調を可能にする。

時間の単位

時間の単位としてはティック(tick)を使用する。これは 4 分音符の 480 分の 1 の長さに相当し、整数に限定しない。可読性を考えて 4 分音符の長さを 100 や 1 ティックにすることも検討したが、5 連符までの音価を誤差なく表現できる利点を優先した。

デフォルトのテンポは 125 BPM とする。このとき 1 ティックは 1 ミリ秒に一致し、リアルタイム MIDI 入出力などテンポを使わない応用では、すべての時間をミリ秒だとみなして処理することが可能である。

音符の表現

各音符を表現する方法として、1 つのイベントで

The design of the Pytakt music information processing library
† NISHIMURA Satoshi, University of Aizu
‡ MARUI Atsushi, Tokyo University of the Arts

表現する方法と、ノート・オンとノート・オフという１対のイベントを用いる方法の２つが考えられる。音価を容易に取得できることから一般には前者が使いやすいが、イベントを先頭から時間順に走査してゆき各区間における発音状況を調べるような場合には後者の方が適している。また、リアルタイム処理では後者が必須となる。そこで Pytakt では、この両方の形式を許可（ただし、混在は禁止）し、その間の変換を行う手段を提供するものとする。代案としてリンクで接続されたノート・オンとノート・オフの対を検討したが、スコアの複製が複雑になるため採用しなかった。

スコアの構造

Pytakt スコアの構造は、次の３つのうちのいずれかとする。

- **EventList** ― イベントのリストにスコア全体の時間長(duration)の情報を加えたもの。時間長はイベントの最大時刻と必ずしも一致しない。
- **Tracks** ― 複数の EventList もしくは Tracks のコレクションであり、要素スコアは同時並行的に演奏される。Tracks の時間長は要素スコアのうちの最大値である。
- **EventStream** ― Python のジェネレータの仕組みを使ったスコアであり、イベントを時間順に yield するイテレータとして働く。これにより無限長のスコアの表現が可能となる。

構造間の相互変換は可能である。ただし、無限長の EventStream を EventList や Tracks に変換した場合は有限長で打ち切られる。

スコア間の演算

スコア間の演算として以下のものを提供する。これにより、小さなスコアを組み合わせてより大きなスコアを構築することが容易となる。

1. 逐次結合($s_1 + s_2$)。スコアs_1の時間長の分だけスコアs_2の演奏開始時刻を遅らせて１つに併合したスコアを生成する。
2. 併合($s_1 \,\&\, s_2$)。２つのスコアs_1, s_2を同時に演奏するスコアを生成する。
3. 繰り返し($s_1 * N$)。スコアs_1をN回演奏するスコアを生成する。

Music Macro Language (MML)

スコア生成を簡便に行う手段として、拡張された MML による記述法を提供する。MML は 1980 年代に主に BASIC 言語に埋め込まれた形で、日本を中心に広まった音楽記述法で、短い文字列で音楽を表現できる特徴がある。Pytakt では、オリジナルの MML を基本に、和音、多声記述、TeX ライクなパラメータ指定など Takt [4] における拡張記法を取り入れ、さらに Python のコードを埋め込めるようにして機能性を向上させる。また、文字の割り当てを変更するなど、カスタマイズ機能を提供する。

エフェクタ

スコア変換を行うためのオブジェクトをエフェクタと呼ぶ。ピッチ変換、時間変換、選択、パターン適用など可能である。

MIDI 入出力

イベントを介した MIDI 入出力が可能であり、他にはない特徴として、任意の時間順序でイベントを挿入可能な出力キューを持つ。

3. 使用例

下に現在の実装における Pytakt の使用例を示す。

```
>>> from pytakt import *  # モジュールインポート
>>> s = mml('cde')  # MML によるスコア生成
>>> s  # 内容の表示
EventList(duration=1440, events=[
    NoteEvent(t=0, n=C4, L=480, v=80, nv=None,
              tk=1, ch=1),
    NoteEvent(t=480, n=D4, L=480, v=80,
              nv=None, tk=1, ch=1),
    NoteEvent(t=960, n=E4, L=480, v=80,
              nv=None, tk=1, ch=1)])
>>> s.play()  # スコア演奏
>>> s.show()  # ピアノロール表示
>>> s.music21().show() # 五線譜表示(music21 経由)
>>> s2 = mml("^d{gab^c}/ ^dgg") & \
mml("o=3 [{g~a} b~~ ^d~~] b~~") #他の MML 使用例
>>> s3 = readsmf('menuet.mid') # SMF 読み込み
>>> s3 = s3.Modify('v*=0.8') #ベロシティを 0.8 倍
>>> s3.writesmf('menuet2.mid')  # SMF 書き出し
```

4. おわりに

本稿では Pytakt の設計コンセプトを中心に概要を述べた。開発したソースコードは公開する予定であり、現在そのための準備を進めている。

参考文献

[1] M. S. Cuthbert and C. Ariza, “music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data,” Proc. of ISMIR, pp. 637-642, 2010.

[2] C. Raffel and D. P. W. Ellis., “Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi,” Proc. of ISMIR, Late Breaking and Demo Papers, 2014.

[3] O. M. Bjørndalen, et al., “Mido - MIDI Objects for Python,” Accessed on 2023/12/26, https://github.com/mido/mido.

[4] S. Nishimura, “Takt: A read-eval-play-loop interpreter for a structural/procedural score language,” Proc. of ICMC, pp. 1736-1741, 2014.