Computer Music – Âm Nhạc Máy Tính

Nguyễn Quản Bá Hồng*

Ngày 14 tháng 8 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Computer Music – Âm Nhạc Máy Tính.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/computer_music/NQBH_computer_music.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/computer_music/NQBH_computer_music.tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

1 Basic Computer Music **2**
1.1 Arnaud Dessein, Arshia Cont, Guillaume Lemaitre. Real-Time Polyphonic Music Transcription with Nonnegative Matrix Factorization & Beta-Divergence. International Society for Music Information Retrieval. 2010 2
1.2 Renato Fabbri, Vilson Vieira de Silva Junior, Antônio Carlos Silvano Pessotti, Débora Cristina Corrêa, Osvaldo N. Oliveira Jr. Musical Elements in Discrete-Time Representation of Sound 9
1.3 [HWR22]. Michael S. Horn, Melanie West, Cameron Roberts. Introduction to Digital Music with Python Programming: Learning Music with Code 13
1.4 [KD06]. Anssi Klapuri, Manuel Davy. Signal Processing Methods for Music Transcription. 2006 72
1.5 Mengshan Li. Design & Implementation of Piano Audio Automatic Music Transcription Algorithm Based on CNN 77
1.6 [Mül15; Mül21]. Meinard Müller. Fundamentals of Music Processing: Using Python & Jupyter Notebooks. 2e 84
1.7 [Väl+06]. Vesa Välimäki, Jyri Pakarinen, Cumhur Erkut, Matti Karjalainen. Discrete-Time Modeling of Musical Instruments 123

2 Librosa **130**
2.1 Brian McFee, Colin Raffel, Dawen Liang, Daniel P. W. Ellis, Matt McVicar, Eric Battenberg, Oriol Nieto. librosa: Audio & Music Signal Analysis in Python 130

3 Wikipedia **135**
3.1 Wikipedia/computer music 135
3.1.1 History 135
3.1.2 Advances 135
3.1.3 Research 135
3.1.4 Machine improvisation 135
3.1.5 Live coding 135
3.2 Wikipedia/octave 135
3.2.1 Explanation & definition 136
3.2.2 Music theory 136
3.2.3 Notation 136
3.2.4 Equivalence 137
3.3 Wikipedia/transcription (music) 137
3.3.1 Adaptation 137
3.3.2 Transcription aids 138
3.3.3 Automatic music transcription (AMT) 138

4 Miscellaneous **139**

Tài liệu **139**

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

1 Basic Computer Music

1.1 Arnaud Dessein, Arshia Cont, Guillaume Lemaitre. Real-Time Polyphonic Music Transcription with Nonnegative Matrix Factorization & Beta-Divergence. International Society for Music Information Retrieval. 2010

[131 citations]

- Abstract. Investigate problem of real-time polyphonic music transcription by employing nonnegative matrix factorization techniques & β-divergence as a cost function. Consider real-world setups where music signal arrives incrementally to system & is transcribed as it unfolds in time. Proposed transcription system is addressed with a modified nonnegative matrix factorization scheme, called nonnegative decomposition, when incoming signal is projected onto a fixed basis of templates learned off-line prior to decomposition. Discuss use of nonnegative matrix factorization with β-divergence to achieve real-time decomposition. Proposed system is evaluated on specific task of piano music transcription & results show: it can outperform several state-of-art offline approaches.

  – Nghiên cứu vấn đề phiên âm nhạc đa âm thời gian thực bằng cách sử dụng kỹ thuật phân tích ma trận không âm & β-divergence làm hàm chi phí. Xem xét các thiết lập thực tế trong đó tín hiệu âm nhạc đến hệ thống từng bước & được phiên âm khi nó diễn ra theo thời gian. Hệ thống phiên âm đề xuất được xử lý bằng 1 sơ đồ phân tích ma trận không âm đã được sửa đổi, được gọi là phân tích không âm, khi tín hiệu đầu vào được chiếu lên 1 cơ sở cố định các mẫu đã học ngoại tuyến trước khi phân tích. Thảo luận về việc sử dụng phân tích ma trận không âm với β-divergence để đạt được phân tích thời gian thực. Hệ thống đề xuất được đánh giá trên 1 nhiệm vụ cụ thể là phiên âm nhạc piano & kết quả cho thấy: nó có thể vượt trội hơn 1 số phương pháp ngoại tuyến state-of-art.

- 1. Introduction. Task of music transcription consists in converting a raw music signal into a symbolic representation e.g. a score. Considering polyphonic signals, this task is closely related to problem of multiple-pitch estimation which has been largely investigated for music as well as speech, & for which a wide variety of methods have been proposed [8]. Nonnegative matrix factorization has already been used in this context, with offline approaches [1, 3, 20, 22–24] as well as on-line approaches [4, 6, 7, 17, 21].

  – Nhiệm vụ của phiên âm nhạc bao gồm việc chuyển đổi tín hiệu âm nhạc thô thành biểu diễn ký hiệu, ví dụ như bản nhạc. Xét về tín hiệu đa âm, nhiệm vụ này liên quan chặt chẽ đến bài toán ước lượng cao độ đa âm, vốn đã được nghiên cứu rộng rãi cho cả âm nhạc & lời nói, & rất nhiều phương pháp đã được đề xuất [8]. Phân tích ma trận không âm đã được sử dụng trong bối cảnh này, với các phương pháp ngoại tuyến [1, 3, 20, 22–24] cũng như các phương pháp trực tuyến [4, 6, 7, 17, 21].

  Generally speaking, nonnegative matrix factorization (NMF) is a technique for data analysis where observed data are supposed to be nonnegative [16]. Main philosophy of NMF is to build up these observations in a constructive additive manner, what is particularly interesting when negative values cannot be interpreted (e.g. pixel intensity, word occurrence, magnitude spectrum).

  – Nói chung, phân tích ma trận không âm (NMF) là 1 kỹ thuật phân tích dữ liệu trong đó dữ liệu quan sát được cho là không âm [16]. Triết lý chính của NMF là xây dựng các quan sát này theo cách cộng tính mang tính xây dựng, điều này đặc biệt thú vị khi không thể diễn giải các giá trị âm (ví dụ: cường độ điểm ảnh, sự xuất hiện của từ, phổ độ lớn).

  In this paper, employ NMF techniques to develop a real-time system for polyphonic music transcription. This system is thought as a front-end for musical interactions in live performances. Among applications, interested in computer-assisted improvisation for instruments e.g. piano. Do not discuss such applications in paper but rather concentrate on system for polyphonic music transcription & invite curious reader to visit companion website for complementary information & additional resources. Proposed system is addressed with an NMF scheme called nonnegative decomposition where signal is projected in real-time onto a basis of note templates learned offline prior to decomposition.

  – Trong bài báo này, chúng tôi sử dụng các kỹ thuật NMF để phát triển 1 hệ thống thời gian thực cho việc phiên âm nhạc đa âm. Hệ thống này được coi là nền tảng cho các tương tác âm nhạc trong các buổi biểu diễn trực tiếp. Trong số các ứng dụng, chúng tôi quan tâm đến việc ứng tác với sự hỗ trợ của máy tính cho các nhạc cụ, ví dụ như piano. Bài báo không thảo luận về các ứng dụng này mà tập trung vào hệ thống phiên âm nhạc đa âm & mời độc giả tò mò truy cập trang web đồng hành để biết thêm thông tin & các tài nguyên bổ sung. Hệ thống được đề xuất được xử lý bằng 1 lược đồ NMF gọi là phân tích không âm, trong đó tín hiệu được chiếu theo thời gian thực lên cơ sở các mẫu nốt nhạc đã học ngoại tuyến trước khi phân tích.

  In this context, price to pay for simplicity of standard NMF is overuse of templates to construct incoming signal, resulting in note insertions & substitutions e.g. octave & harmonic errors. In [6, 7], issue has been tackled with standard Euclidean cost by introduction of a sparsity constraint similar to [14]. Here investigate use of more complex costs by using β-divergence. This is in contrast to previous systems for real-time audio decomposition which have either considered Euclidean distance or Kullback–Leibler divergence. NMP with β-divergence has recently proved its relevancy for offline applications in speech analysis [18], music analysis [11] & music transcription [3, 23]. Adapt these approaches to a real-time setup & propose a tailored multiplicative update to compute decomposition. Also give intuition in understanding how β-divergence helps to improve transcription. Provided evaluation show: proposed system can outperform several offline algorithms at state-of-art.

  – Trong bối cảnh này, cái giá phải trả cho sự đơn giản của NMF chuẩn là việc lạm dụng các mẫu để xây dựng tín hiệu đầu vào, dẫn đến việc chèn & thay thế nốt nhạc, ví dụ như lỗi quãng tám & hài âm. Trong [6, 7], vấn đề đã được giải quyết bằng chi phí Euclidean chuẩn bằng cách đưa ra ràng buộc thưa thớt tương tự như [14]. Ở đây, hãy nghiên cứu việc sử dụng các chi phí phức tạp hơn bằng cách sử dụng β-divergence. Điều này trái ngược với các hệ thống trước đây để phân tích âm thanh thời

gian thực, vốn đã xem xét khoảng cách Euclidean hoặc phân kỳ Kullback–Leibler. NMP với β-divergence gần đây đã chứng minh được tính phù hợp của nó đối với các ứng dụng ngoại tuyến trong phân tích giọng nói [18], phân tích âm nhạc [11] & phiên âm nhạc [3, 23]. Hãy áp dụng các cách tiếp cận này vào thiết lập thời gian thực & đề xuất 1 bản cập nhật nhân được điều chỉnh để tính toán phân tích. Đồng thời cung cấp trực giác để hiểu cách β-divergence giúp cải thiện phiên âm. Đánh giá được cung cấp cho thấy: hệ thống được đề xuất có thể vượt trội hơn 1 số thuật toán ngoại tuyến ở mức hiện đại.

Paper is organized as follows. In Sect. 2, introduce a related background on NMF techniques. In Sect. 3, focus on NMF with β-divergence, provide a multiplicative update tailored to real-time decomposition, & discuss relevancy of β-divergence for decomposition of polyphonic music signals. In Sect. 4, depict general architecture of real-time system proposed for polyphonic music transcription, & detail 2 modules resp used for offline learning of note templates & for online decomposition of music signals. In Sect. 5, perform evaluations of system for specific task of piano music transcription.

– Bài báo được tổ chức như sau. Trong Phần 2, giới thiệu 1 số kiến thức nền tảng liên quan đến các kỹ thuật NMF. Trong Phần 3, tập trung vào NMF với β-divergence, cung cấp 1 bản cập nhật nhân được điều chỉnh theo phân tích thời gian thực, & thảo luận về sự liên quan của β-divergence đối với việc phân tích tín hiệu âm nhạc đa âm. Trong Phần 4, mô tả kiến trúc tổng quát của hệ thống thời gian thực được đề xuất cho việc phiên âm nhạc đa âm, & chi tiết 2 mô-đun được sử dụng cho việc học ngoại tuyến các mẫu nốt & cho việc phân tích trực tuyến các tín hiệu âm nhạc. Trong Phần 5, thực hiện đánh giá hệ thống cho nhiệm vụ cụ thể là phiên âm nhạc piano.

In sequel, uppercase bold letters denote matrices, lowercase bold letters denote column vectors, lowercase plain letters denote scalars. $\mathbb{R}_+, \mathbb{R}_{++}$ denote resp sets of nonnegative & positive scalars. Elementwise multiplication & division between 2 matrices A, B are denoted resp by $A \otimes B$, $\frac{A}{B}$. Elementwise power p of A is denoted by $A^{\cdot p}$.

– Tiếp theo, chữ in hoa đậm biểu thị ma trận, chữ in thường đậm biểu thị vectơ cột, chữ thường thường nhạt biểu thị số vô hướng. $\mathbb{R}_+, \mathbb{R}_{++}$ biểu thị các tập hợp tương ứng của các số vô hướng không âm & dương. Phép nhân & phép chia từng phần tử giữa 2 ma trận A, B được ký hiệu tương ứng là $A \otimes B$, $\frac{A}{B}$. Lũy thừa từng phần tử p của A được ký hiệu là $A^{\cdot p}$.

- 2. Related background. This sect introduces NMF model, standard NMF problem, & popular multiplicative updates algorithm used to solve it. Then present relevant literature in sound recognition with NMF.

  – Phần này giới thiệu mô hình NMF, bài toán NMF tiêu chuẩn & thuật toán cập nhật nhân phổ biến được sử dụng để giải bài toán này. Sau đó, trình bày các tài liệu liên quan về nhận dạng âm thanh bằng NMF.

  - 2.1. NMF model. NMF model is a low-rank approximation for unsupervised multivariate data analysis. Given an $n \times m$ nonnegative matrix V & a positive integer $r < \min\{m, n\}$, NMF tries to factorize V into an $n \times r$ nonnegative matrix W & an $r \times m$ nonnegative matrix H s.t. (1) $V \approx WH$. In this model, multivariate data are stacked into V, whose columns represent different observations, & whose rows represent different variables. Each column $\mathbf{v}_j$ of V can be expressed as $\mathbf{v}_j \approx W\mathbf{h}_j = \sum_i h_{ij}\mathbf{w}_i$, where $\mathbf{w}_i, \mathbf{h}_j$ are resp ith column of W & jth column of H. Columns of W then form a basis & each column of H is decomposition of corresponding column of V into this basis.

    – Mô hình NMF là 1 phép xấp xỉ hạng thấp cho phân tích dữ liệu đa biến không giám sát. Cho 1 ma trận không âm V $n \times m$ & 1 số nguyên dương $r < \min\{m, n\}$, NMF cố gắng phân tích V thành 1 ma trận không âm W $n \times r$ & 1 ma trận không âm H $r \times m$ s.t. (1) $V \approx WH$. Trong mô hình này, dữ liệu đa biến được xếp chồng thành V, trong đó các cột biểu diễn các quan sát khác nhau, & các hàng biểu diễn các biến khác nhau. Mỗi cột $\mathbf{v}_j$ của V có thể được biểu thị là $\mathbf{v}_j \approx W\mathbf{h}_j = \sum_i h_{ij}\mathbf{w}_i$, trong đó $\mathbf{w}_i, \mathbf{h}_j$ tương ứng với cột thứ i của W & cột thứ j của H. Các cột của W sau đó tạo thành 1 cơ sở & mỗi cột của H là sự phân tích của cột tương ứng của V thành cơ sở này.

  - 2.2. Standard problem & multiplicative updates. Standard NMF model of (1) provides an approximate factorization WH of V. Aim: find factorization which optimizes a given goodness-of-fit measure called cost function. In standard formulation, Euclidean distance is used, & NMF problem amounts to minimizing following cost function subject to nonnegativity of both W, H: (2)

$$\frac{1}{2}\|V - WH\|_F^2 = \frac{1}{2}\sum_j \|\mathbf{v}_j - W\mathbf{h}_j\|_2^2.$$

    For this particular cost function, factors W, H can be computed with popular multiplicative updates introduced in [16]. These updates are derived from a gradient descent scheme with judiciously chosen steps, as follows (3):

$$H \leftarrow H \otimes \frac{W^\top V}{W^\top WH},\ W \leftarrow W \otimes \frac{VH^\top}{WHH^\top}.$$

    Updates are applied in turn until convergence, & ensure both nonnegativity & decreasing of cost, but not necessarily local optimality of factors W, H.

    – Bài toán chuẩn & cập nhật phép nhân. Mô hình NMF chuẩn của (1) cung cấp 1 phép phân tích gần đúng WH của V. Mục tiêu: tìm phép phân tích tối ưu hóa 1 phép đo độ phù hợp tốt nhất cho trước được gọi là hàm chi phí. Trong công thức chuẩn, khoảng cách Euclid được sử dụng, & Bài toán NMF về cơ bản là tối thiểu hóa hàm chi phí sau với điều kiện cả W, H đều không âm: (2)

$$\frac{1}{2}\|V - WH\|_F^2 = \frac{1}{2}\sum_j \|\mathbf{v}_j - W\mathbf{h}_j\|_2^2.$$

Đối với hàm chi phí cụ thể này, các thừa số W, H có thể được tính bằng các phép cập nhật phép nhân phổ biến được giới thiệu trong [16]. Các bản cập nhật này được suy ra từ 1 sơ đồ giảm dần độ dốc với các bước được lựa chọn thận trọng, như sau (3):

$$H \leftarrow H \otimes \frac{W^\top V}{W^\top W H},\ W \leftarrow W \otimes \frac{V H^\top}{W H H^\top}.$$

Các bản cập nhật được áp dụng lần lượt cho đến khi hội tụ, & đảm bảo cả tính không âm & giảm chi phí, nhưng không nhất thiết là tính tối ưu cục bộ của các hệ số W, H.

A flourishing literature exists about extensions to standard NMF problem & their algorithms [5]. These extensions can be thought of in terms of modified cost functions (e.g., using divergences or adding penalty terms), of modified constraints (e.g. imposing sparsity), & of modified models (e.g. using tensors). E.g., cost function defined in (2) is often replaced with Kullback-Leibler divergence for which specific multiplicative updates have been derived [16].

– Có rất nhiều tài liệu về các phần mở rộng cho bài toán NMF tiêu chuẩn & các thuật toán của chúng [5]. Các phần mở rộng này có thể được xem xét dưới dạng các hàm chi phí đã sửa đổi (ví dụ: sử dụng phân kỳ hoặc thêm các điều khoản phạt), các ràng buộc đã sửa đổi (ví dụ: áp đặt độ thưa thớt), & các mô hình đã sửa đổi (ví dụ: sử dụng tenxơ). Ví dụ, hàm chi phí được định nghĩa trong (2) thường được thay thế bằng phân kỳ Kullback-Leibler mà các cập nhật nhân cụ thể đã được suy ra [16].

- 2.3. Applications in sound recognition. NMF algorithm have been applied to various problems in vision, sound analysis, biomedical data analysis & text classification among others [5]. In context fo sound analysis, matrix V is in general a time-frequency representation of sound to analyze. Rows & columns representation of sound to analyze. Rows & columns represent resp. different frequency bins & successive time-frames. Factorization $\mathbf{v}_j \approx \sum_i h_{ij}\mathbf{w}_i$ can then be interpreted as follows: each basis vector $\mathbf{w}_i$ contains a spectral template, & decomposition coefficients h_{ij} represent activations of ith template $\mathbf{w}_i$ at jth time-frame.

  – Ứng dụng trong nhận dạng âm thanh. Thuật toán NMF đã được áp dụng cho nhiều vấn đề khác nhau trong thị giác, phân tích âm thanh, phân tích dữ liệu y sinh & phân loại văn bản trong số những vấn đề khác [5]. Trong bối cảnh phân tích âm thanh, ma trận V nói chung là biểu diễn tần số thời gian của âm thanh để phân tích. Hàng & cột biểu diễn âm thanh để phân tích. Hàng & cột biểu diễn tương ứng các thùng tần số khác nhau & khung thời gian liên tiếp. Phân tích $\mathbf{v}_j \approx \sum_i h_{ij}\mathbf{w}_i$ sau đó có thể được diễn giải như sau: mỗi vectơ cơ sở $\mathbf{w}_i$ chứa 1 mẫu phổ, & hệ số phân tích h_{ij} biểu diễn các hoạt động của mẫu ith $\mathbf{w}_i$ tại khung thời gian jth.

  NMF has already been used in context of polyphonic music transcription (e.g. see [1, 22]). Several problem-dependent extensions have been developed to this end e.g. a source-filter model [24], an harmonic constraint [20], an harmonic model with temporal smoothness [3], or an harmonic model with spectral smoothness [23]. These approaches rely in general on offline nature of NMF, but some authors have used NMF in an on-line setup.

  – NMF đã được sử dụng trong bối cảnh phiên âm nhạc đa âm (ví dụ: xem [1, 22]). 1 số phần mở rộng phụ thuộc vào vấn đề đã được phát triển cho mục đích này, ví dụ: mô hình bộ lọc nguồn [24], ràng buộc hài hòa [20], mô hình hài hòa với độ mịn thời gian [3] hoặc mô hình hài hòa với độ mịn phổ [23]. Các phương pháp này nhìn chung dựa trên bản chất ngoại tuyến của NMF, nhưng 1 số tác giả đã sử dụng NMF trong thiết lập trực tuyến.

  A real-time system to identify presence & determine pitch of 1 or more voices is proposed in [21]. This system is also adapted for sight-reading evaluation of solo instrument in [4]. Concerning automatic transcription, a similar system is used in [17] for transcription of polyphonic music, & in [19] for drum transcription. A real-time system for polyphonic music transcription with sparsity considerations is proposed in [6]. Approach is further developed in [7] for real-time coupled multiple-pitch & multiple-instrument recognition. Yet, all these approaches are based on NMF with Euclidean distance or Kullback-Leibler divergence. Discuss use of more general β-divergence as a cost function & its relevancy for decomposition of music signals in Sect. 3.

  – 1 hệ thống thời gian thực để xác định sự hiện diện & xác định cao độ của 1 hoặc nhiều giọng hát được đề xuất trong [21]. Hệ thống này cũng được điều chỉnh để đánh giá đọc nhạc cụ độc tấu trong [4]. Về phiên âm tự động, 1 hệ thống tương tự được sử dụng trong [17] để phiên âm nhạc đa âm, & trong [19] để phiên âm trống. 1 hệ thống thời gian thực để phiên âm nhạc đa âm với các cân nhắc về độ thưa thớt được đề xuất trong [6]. Phương pháp tiếp cận được phát triển thêm trong [7] để nhận dạng nhiều cao độ & nhiều nhạc cụ được ghép nối theo thời gian thực. Tuy nhiên, tất cả các phương pháp tiếp cận này đều dựa trên NMF với khoảng cách Euclid hoặc độ phân kỳ Kullback-Leibler. Thảo luận về việc sử dụng β-độ phân kỳ tổng quát hơn như 1 hàm chi phí & sự liên quan của nó đối với việc phân tích tín hiệu âm nhạc trong Phần 3.

- 3. Nonnegative decomposition with β-divergence. Define β-divergence, give some of its properties, & review its use as a cost function for NMF. Finally formulate nonnegative decomposition problem with β-divergence & give multiplicative updates tailored to real-time for solving it.

  – Phân tích không âm với β-divergence. Định nghĩa β-divergence, đưa ra 1 số tính chất của nó, & xem xét việc sử dụng nó làm hàm chi phí cho NMF. Cuối cùng, xây dựng bài toán phân tích không âm với β-divergence & đưa ra các cập nhật nhân được điều chỉnh theo thời gian thực để giải bài toán.

  - 3.1. Definition & properties of beta-divergence. β-divergences form a parametric family of distortion functions [9]. For any $\beta \in \mathbb{R}$ & any points $x, y \in \mathbb{R}_{++}$, β-divergence from x to y is defined as follows: (4)

$$d_\beta(x|y) \coloneqq \frac{1}{\beta(\beta-1)}(x^\beta + (\beta-1)y^\beta - \beta x y^{\beta-1}).$$

As special cases when $\beta = 0, \beta = 1$, taking limits in above definition leads resp. to well-known Itakura-Saito & Kullback-Leibler divergences: (5)–(6)

– Định nghĩa & tính chất của phân kỳ beta. Phân kỳ β tạo thành 1 họ tham số của các hàm méo [9]. Với bất kỳ $\beta \in \mathbb{R}$ & bất kỳ điểm $x, y \in \mathbb{R}_{++}$, phân kỳ β từ x đến y được định nghĩa như sau: (4)

$$d_\beta(x|y) \coloneqq \frac{1}{\beta(\beta-1)}(x^\beta + (\beta-1)y^\beta - \beta x y^{\beta-1}).$$

Là trường hợp đặc biệt khi $\beta = 0, \beta = 1$, việc lấy giới hạn trong định nghĩa trên tương ứng dẫn đến đến sự phân kỳ Itakura-Saito & Kullback-Leibler nổi tiếng: (5)–(6)

$$d_{\beta=0}(x|y) = d_{\mathrm{IS}}(x|y) = \frac{x}{y} - \log\frac{x}{y} - 1,$$
$$d_{\beta=1}(x|y) = d_{\mathrm{KL}}(x|y) = x\log\frac{x}{y} + y - x.$$

For $\beta = 2$, β-divergence specializes to widely used half squared Euclidean distance (7)

$$d_{\beta=2}(x|y) = d_E(x|y) = \frac{1}{2}(x-y)^2.$$

Concerning their properties, all β-divergences are nonnegative & vanish iff $x = y$. However, they are not necessary distances in strict terms since they are not symmetric & do not satisfy triangle inequality in general. A property of β-divergences relevant to present work: for any scaling factor $\lambda \in \mathbb{R}_{++}$ have (8)

$$d_\beta(\lambda x|\lambda y) = \lambda^\beta d_\beta(x|y).$$

Discuss further interest of this scaling property for decomposition of polyphonic music signals in Sect. 3.3.

– Với $\beta = 2$, phân kỳ β chuyên biệt hóa khoảng cách Euclidean nửa bình phương được sử dụng rộng rãi (7)

$$d_{\beta=2}(x|y) = d_E(x|y) = \frac{1}{2}(x-y)^2.$$

Về các tính chất của chúng, tất cả các phân kỳ β đều không âm & triệt tiêu khi & chỉ khi $x = y$. Tuy nhiên, chúng không phải là khoảng cách cần thiết theo nghĩa chặt chẽ vì chúng không đối xứng & không thỏa mãn bất đẳng thức tam giác nói chung. 1 tính chất của phân kỳ β liên quan đến công trình hiện tại: với bất kỳ hệ số tỷ lệ $\lambda \in \mathbb{R}_{++}$ nào thì có (8)

$$d_\beta(\lambda x|\lambda y) = \lambda^\beta d_\beta(x|y).$$

Thảo luận thêm về tính chất tỷ lệ này trong việc phân tích tín hiệu âm nhạc đa âm trong Phần 3.3.

- 3.2. NMF & β-divergence. β-divergence was 1st used with NMF to interpolate between Euclidean distance & Kullback-Leibler divergence [15]. Starting with scalar divergence in (4), a matrix divergence can be constructed as a separable divergence, i.e., by summing element-wise divergences. NMF problem with β-divergence then amounts to minimizing following cost function subject to nonnegativity of both W, H: (9)

$$\mathcal{D}_\beta(V|WH) = \sum_{i,j} d_\beta(v_{ij}|[WH]_{ij}).$$

For $\beta = 2$, this cost function specializes to cost defined in (2) for standard NMF.

– NMF & β-divergence. β-divergence lần đầu tiên được sử dụng với NMF để nội suy giữa khoảng cách Euclidean & phân kỳ Kullback-Leibler [15]. Bắt đầu với phân kỳ vô hướng trong (4), phân kỳ ma trận có thể được xây dựng như 1 phân kỳ tách rời, tức là bằng cách cộng các phân kỳ từng phần tử. Bài toán NMF với β-divergence sau đó sẽ là việc tối thiểu hóa hàm chi phí sau với điều kiện cả W, H đều không âm: (9)

$$\mathcal{D}_\beta(V|WH) = \sum_{i,j} d_\beta(v_{ij}|[WH]_{ij}).$$

Với $\beta = 2$, hàm chi phí này được chuyên biệt hóa thành chi phí được xác định trong (2) cho NMF chuẩn.

As for standard NMF, several algorithms including multiplicative updates have been derived for NMF with β-divergence & its extensions [5, 15]. β-divergence has also proved its relevancy as a cost function for audio offline applications in speech analysis [18], music analysis [11] & music transcription [3, 23].

– Đối với NMF tiêu chuẩn, 1 số thuật toán bao gồm các bản cập nhật nhân đã được phát triển cho NMF với β-divergence & các phần mở rộng của nó [5, 15]. β-divergence cũng đã chứng minh được tính liên quan của nó như 1 hàm chi phí cho các ứng dụng âm thanh ngoại tuyến trong phân tích giọng nói [18], phân tích âm nhạc [11] & phiên âm nhạc [3, 23].

◦ 3.3. Problem formulation & multiplicative update. Now formulate problem of nonnegative decomposition with β-divergence. Assume: W is a fixed dictionary of note templates onto which we seek to decompose incoming signal $\mathbf{v}$ as $\mathbf{v} \approx W\mathbf{h}$. Problem is therefore equivalent to minimizing following cost function subject to nonnegativity of $\mathbf{h}$: (10)

– Xây dựng bài toán & cập nhật nhân. Bây giờ, hãy xây dựng bài toán phân tích không âm với β-phân kỳ. Giả sử: W là 1 từ điển cố định các mẫu nốt nhạc mà chúng ta muốn phân tích tín hiệu đầu vào $\mathbf{v}$ thành $\mathbf{v} \approx W\mathbf{h}$. Do đó, bài toán tương đương với việc tối thiểu hóa hàm chi phí sau với điều kiện $\mathbf{h}$ không âm: (10)

$$\mathcal{D}_\beta(\mathbf{v}|W\mathbf{h}) = \sum_i d_\beta(v_i|[W\mathbf{h}]_i).$$

To solve this problem, update $\mathbf{h}$ iteratively by using a vector version of corresponding multiplicative update proposed in literature [5, 15]. As W is fixed, never apply its respective update. Algorithm thus amounts to repeating following update until convergence: (11)

$$\mathbf{h} \leftarrow \mathbf{h} \otimes \frac{W^\top((W\mathbf{h})^{\cdot\beta-2} \otimes \mathbf{v})}{W^\top(W\mathbf{h})^{\cdot\beta-1}}.$$

This scheme ensures nonnegativity of $\mathbf{h}$, but not necessarily local optimality. Unfortunately, no proof has been found yet to show: cost function is non-increasing under this update for a general parameter β, even if it has been observed in practice [11]. However, even if such theoretical issues need to be investigated further, simplicity of this scheme makes it suitable for real-time applications & gives good results in practice.

– Để giải quyết vấn đề này, hãy cập nhật $\mathbf{h}$ theo chu kỳ lặp bằng cách sử dụng phiên bản vectơ của bản cập nhật nhân tương ứng được đề xuất trong tài liệu [5, 15]. Vì W cố định, không bao giờ áp dụng bản cập nhật tương ứng của nó. Do đó, thuật toán tương đương với việc lặp lại bản cập nhật sau cho đến khi hội tụ: (11)

$$\mathbf{h} \leftarrow \mathbf{h} \otimes \frac{W^\top((W\mathbf{h})^{\cdot\beta-2} \otimes \mathbf{v})}{W^\top(W\mathbf{h})^{\cdot\beta-1}}.$$

Sơ đồ này đảm bảo tính không âm của $\mathbf{h}$, nhưng không nhất thiết đảm bảo tính tối ưu cục bộ. Thật không may, vẫn chưa có bằng chứng nào cho thấy: hàm chi phí không tăng theo bản cập nhật này đối với tham số chung β, ngay cả khi nó đã được quan sát thấy trong thực tế [11]. Tuy nhiên, ngay cả khi các vấn đề lý thuyết như vậy cần được nghiên cứu thêm, tính đơn giản của lược đồ này khiến nó phù hợp với các ứng dụng thời gian thực & mang lại kết quả tốt trong thực tế.

Concerning implementation, can take advantage of W being fixed to employ a multiplicative update tailored to real-time decomposition. Indeed, after some matrix manipulations, can rewrite updates as follows: (12)

$$\mathbf{h} \leftarrow \mathbf{h} \otimes \frac{(W \otimes (\mathbf{v}\mathbf{e}^\top))^\top(W\mathbf{h})^{\cdot\beta-2}}{W^\top(W\mathbf{h}^{\cdot\beta-1})},$$

where $\mathbf{e}$ is a vector full of 1s. This helps to reduce computational cost of update scheme as matrix $(W \times (\mathbf{v}\mathbf{e}^\top))^\top$ needs only to be computed once.

– Về mặt triển khai, có thể tận dụng W được cố định để sử dụng phép cập nhật nhân được điều chỉnh theo phân tích thời gian thực. Thật vậy, sau 1 số thao tác ma trận, có thể viết lại các phép cập nhật như sau: (12)

$$\mathbf{h} \leftarrow \mathbf{h} \otimes \frac{(W \otimes (\mathbf{v}\mathbf{e}^\top))^\top(W\mathbf{h})^{\cdot\beta-2}}{W^\top(W\mathbf{h}^{\cdot\beta-1})},$$

trong đó $\mathbf{e}$ là 1 vectơ chứa đầy 1. Điều này giúp giảm chi phí tính toán của lược đồ cập nhật vì ma trận $(W \times (\mathbf{v}\mathbf{e}^\top))^\top$ chỉ cần tính toán 1 lần.

Scaling property in (8) may give an insight in understanding relevancy of β-divergence in our context. For $\beta = 0$, Itakura-Saito divergence is only β-divergence to be scale-invariant as it was remarked in [11], i.e., corresponding NMF problem gives same relative weight to all coefficients, & thus penalizes equally a bad fit of factorization for small & large coefficients. Considering music signals, this amounts to giving same importance to high-energy & to low-energy frequency components. When $\beta > 0$, more emphasis is put on frequency components of higher energy, & emphasis augments with β. When $\beta < 0$, effect is converse. In our context of music decomposition, try to reconstruct an incoming music signal by addition of note templates. In order to avoid common octave & harmonic errors, a good reconstruction would have to find a compromise between focusing on fundamental frequency, 1st partials & higher partials. Parameter β can thus help to control this trade-off.

– Tính chất tỷ lệ trong (8) có thể cung cấp cái nhìn sâu sắc trong việc hiểu sự liên quan của β-divergence trong bối cảnh của chúng ta. Với $\beta = 0$, phân kỳ Itakura-Saito chỉ là phân kỳ β-bất biến theo tỷ lệ như đã được nhận xét trong [11], tức là, bài toán NMF tương ứng đưa ra cùng 1 trọng số tương đối cho tất cả các hệ số, & do đó phạt như nhau 1 sự phù hợp kém của phân tích nhân tử đối với các hệ số nhỏ & lớn. Xét các tín hiệu âm nhạc, điều này tương đương với việc đưa ra cùng 1 tầm quan trọng cho các thành phần tần số năng lượng cao & cho các thành phần tần số năng lượng thấp. Khi $\beta > 0$, trọng tâm hơn được đặt vào các thành phần tần số có năng lượng cao hơn, & sự nhấn mạnh tăng lên với β. Khi $\beta < 0$, hiệu ứng ngược lại. Trong bối cảnh phân tích âm nhạc của chúng ta, hãy thử tái tạo tín hiệu âm nhạc đến bằng cách thêm các mẫu nốt nhạc. Để tránh các lỗi phổ biến về quãng tám & hài hòa, 1 bản tái tạo tốt sẽ phải tìm ra sự thỏa hiệp giữa việc tập trung vào tần số cơ bản, các thành phần bậc 1 & các thành phần bậc cao hơn. Do đó, tham số β có thể giúp kiểm soát sự đánh đổi này.

- 4. General architecture of system. Present real-time system proposed for polyphonic music transcription. General architecture is shown schematically in Fig. 1: Schematic view of the general architecture. Right side of figure represents music signal arriving in real-time, & its decomposition onto notes whose descriptions are provided a priori to system as templates. These templates are learned offline, as shown on left side of figure, & constitute dictionary used during real-time decomposition. Describe 2 modules hereafter.

  – Kiến trúc tổng quát của hệ thống. Trình bày hệ thống thời gian thực được đề xuất cho việc phiên âm nhạc đa âm. Kiến trúc tổng quát được thể hiện sơ đồ trong Hình 1: Sơ đồ kiến trúc tổng quát. Phía bên phải của hình biểu diễn tín hiệu âm nhạc đến theo thời gian thực, & phân tích tín hiệu này thành các nốt nhạc có mô tả được cung cấp trước cho hệ thống dưới dạng mẫu. Các mẫu này được học ngoại tuyến, như được hiển thị ở phía bên trái của hình, & tạo thành từ điển được sử dụng trong quá trình phân tích thời gian thực. Mô tả 2 mô-đun sau đây.

  - 4.1. Note template learning. Learning module aims at building a dictionary W of note templates onto which polyphonic music signal is projected during real-time decomposition phase.

    – Học mẫu nốt nhạc. Mô-đun học tập nhằm mục đích xây dựng 1 từ điển W các mẫu nốt nhạc mà tín hiệu âm nhạc đa âm được chiếu lên trong giai đoạn phân tích thời gian thực.

    In present work, use a simple rank-1 NMF with standard cost function as a learning scheme. Suppose: user has access to isolated note samples of instruments to transcribe, from which system learns characteristic templates. Whole note sample k is 1st processed in a short-time sound representation supposed to be nonnegative & approximately additive (e.g., a short-time magnitude spectrum). Representations are stacked in a matrix $V^{(k)}$ where each column $\mathbf{v}_j^{(k)}$ is sound representation of jth time-frame. Then solve standard NMF with $V^{(k)}$ & a rank of factorization $r = 1$, using multiplicative updates in (3). This learning scheme simply gives a template $\mathbf{w}^{(k)}$ for each node sample (information in row vector $\mathbf{h}^{(k)}$ is discarded).

    – Trong công trình hiện tại, hãy sử dụng NMF bậc 1 đơn giản với hàm chi phí chuẩn làm sơ đồ học. Giả sử: người dùng có quyền truy cập vào các mẫu nốt nhạc riêng biệt của các nhạc cụ để phiên âm, từ đó hệ thống học các mẫu đặc trưng. Mẫu nốt nhạc toàn bộ k đầu tiên được xử lý trong biểu diễn âm thanh thời gian ngắn được cho là không âm & gần như cộng (ví dụ: phổ biên độ thời gian ngắn). Các biểu diễn được xếp chồng trong ma trận $V^{(k)}$ trong đó mỗi cột $\mathbf{v}_j^{(k)}$ là biểu diễn âm thanh của khung thời gian thứ j. Sau đó, giải NMF chuẩn với $V^{(k)}$ & bậc phân tích thừa số $r = 1$, sử dụng các bản cập nhật nhân trong (3). Sơ đồ học này chỉ đơn giản cung cấp 1 mẫu $\mathbf{w}^{(k)}$ cho mỗi mẫu nút (thông tin trong vectơ hàng $\mathbf{h}^{(k)}$ bị loại bỏ).

  - 4.2. Music signal decomposition. Having learned templates, stack them in columns to form dictionary W. Problem of real-time transcription then amounts to projecting incoming music signal $\mathbf{v}_j$ onto W, where $\mathbf{v}_j$ share same representation front-end as note templates. Problem is thus equivalent to a nonnegative decomposition $\mathbf{v}_j \approx W\mathbf{h}_j$ where W is kept fixed & only $\mathbf{h}_j$ is learned. Learned vectors $\mathbf{h}_j$ would then provide successive activations of different notes in music signal. Following discussion in Sect. 3, learn vectors $\mathbf{h}_j$ by employing β-divergence as a cost function & multiplicative update tailored to real-time decomposition in (11).

    – Phân tích tín hiệu âm nhạc. Sau khi đã học các mẫu, hãy xếp chúng theo cột để tạo thành từ điển W. Vấn đề phiên âm thời gian thực sau đó tương đương với việc chiếu tín hiệu âm nhạc đến $\mathbf{v}_j$ lên W, trong đó $\mathbf{v}_j$ chia sẻ cùng 1 biểu diễn đầu cuối như các mẫu nốt nhạc. Do đó, vấn đề tương đương với phân tích không âm $\mathbf{v}_j \approx W\mathbf{h}_j$ trong đó W được giữ cố định & chỉ học được $\mathbf{h}_j$. Các vectơ đã học $\mathbf{h}_j$ sau đó sẽ cung cấp các kích hoạt liên tiếp của các nốt nhạc khác nhau trong tín hiệu âm nhạc. Tiếp theo thảo luận trong Phần 3, hãy học các vectơ $\mathbf{h}_j$ bằng cách sử dụng β-phân kỳ làm hàm chi phí & cập nhật nhân được điều chỉnh theo phân tích thời gian thực trong (11).

    As such, system reports only a frame-level activity of notes. Some post-processing is thus needed to extract more information about eventual presence of notes, & provide a symbolic representation of music signal for transcription. This post-processing potentially includes activation thresholding, onset detection, temporal modeling, etc. It is however not thoroughly discussed in this paper where use a simple threshold-based detection followed by a minimum duration pruning.

    – Do đó, hệ thống chỉ báo cáo hoạt động ở cấp độ khung hình của các nốt nhạc. Do đó, cần có 1 số xử lý hậu kỳ để trích xuất thêm thông tin về sự hiện diện cuối cùng của các nốt nhạc, & cung cấp 1 biểu diễn tượng trưng của tín hiệu âm nhạc để phiên mã. Quá trình hậu kỳ này có thể bao gồm ngưỡng kích hoạt, phát hiện khởi phát, mô hình hóa thời gian, v.v. Tuy nhiên, bài báo này không thảo luận kỹ lưỡng về vấn đề này khi sử dụng 1 phát hiện dựa trên ngưỡng đơn giản, tiếp theo là cắt tỉa thời lượng tối thiểu.

- 5. Evaluation & results. Evaluate system on polyphonic transcription of piano music. Provide a subjective valuation with musical excerpts synthesized from MIDI references. Also perform an objective evaluation with a real piano music database & standard evaluation metrics.

  – Đánh giá & kết quả. Đánh giá hệ thống về phiên âm đa âm của nhạc piano. Cung cấp đánh giá chủ quan với các trích đoạn nhạc được tổng hợp từ các tài liệu tham khảo MIDI. Đồng thời thực hiện đánh giá khách quan với cơ sở dữ liệu nhạc piano thực tế & các chỉ số đánh giá tiêu chuẩn.

  - 5.1. Subjective evaluation. As sample examples, transcribed 2 musical excerpts synthesized from MIDI references with real piano samples from Real World Computing (RWC) database [12].

    – Đánh giá chủ quan. Là ví dụ mẫu, đã sao chép 2 đoạn trích âm nhạc được tổng hợp từ các tài liệu tham khảo MIDI với các mẫu piano thực từ cơ sở dữ liệu Real World Computing (RWC) [12].

For nonnegative decomposition, β was set to 0.5 since this value was shown optimal for music transcription in [23] & provided good results in our tests. Threshold for detection was set to 2 & no minimum duration pruning was applied. For dictionary, 1 note template was learned & max-normalized for each of 88 notes of piano using corresponding samples taken from RWC. Used a simple short-time magnitude spectrum representation, with a frame size of 50 ms leading to 630 samples at a sampling rate of 12600 Hz, & computed with a 0-padded Fourier transform of 1024 bins. Frames were windowed with a Hamming function, & hopsize was set to 25 ms for template learning & refined to 10 ms for decomposition. Decomposition was computed in real-time simulation under MATLAB on a 2.40 GHz laptop with 4.00 Go of RAM, & was about 3 times faster than real-time.

– Đối với phân tích không âm, β được đặt thành 0,5 vì giá trị này được hiển thị là tối ưu cho phiên âm nhạc trong [23] & cung cấp kết quả tốt trong các thử nghiệm của chúng tôi. Ngưỡng phát hiện được đặt thành 2 & không áp dụng cắt tỉa thời lượng tối thiểu. Đối với từ điển, 1 mẫu nốt đã được học & chuẩn hóa tối đa cho mỗi 88 nốt của đàn piano bằng cách sử dụng các mẫu tương ứng lấy từ RWC. Sử dụng biểu diễn phổ biên độ thời gian ngắn đơn giản, với kích thước khung là 50 ms dẫn đến 630 mẫu ở tốc độ lấy mẫu là 12600 Hz, & được tính toán bằng biến đổi Fourier đệm 0 của 1024 thùng. Các khung được tạo cửa sổ bằng hàm Hamming, & hopsize được đặt thành 25 ms để học mẫu & được tinh chỉnh thành 10 ms để phân tích. Phân tích được tính toán trong mô phỏng thời gian thực dưới MATLAB trên máy tính xách tay 2,40 GHz với 4,00 Go RAM, & nhanh hơn thời gian thực khoảng 3 lần.

Results of decomposition are shown in Fig. 2: Transcription of 2 piano excerpts from Mamère l'Oye, Cinq pièces enfantines pour piano à quatre mains (1908-1910), Maurice Ravel (1875–1937), depict piano-roll representations of 2 piano excerpts. Ground-truth references are represented with rectangles & transcriptions with black dots. Overall, this shows: system is able to match reliably note templates to music signals. During note attacks, more templates are used due to transients but some post-processing e.g. minimum duration pruning would help to remove these errors. Also remark a tendency to shorten sustained notes which may be due to a different spectral content during note releases.

– Kết quả phân tích được thể hiện trong Hình 2: Bản chép lại 2 đoạn trích piano từ Mamère l'Oye, Cinq pièces enfantines pour piano à quatre mains (1908-1910), Maurice Ravel (1875–1937), mô tả các biểu diễn piano-roll của 2 đoạn trích piano. Các tham chiếu thực tế được biểu diễn bằng hình chữ nhật & bản chép lại bằng các chấm đen. Nhìn chung, điều này cho thấy: hệ thống có thể khớp các mẫu nốt nhạc với tín hiệu âm nhạc 1 cách đáng tin cậy. Trong các đợt tấn công nốt nhạc, nhiều mẫu hơn được sử dụng do các tín hiệu thoáng qua nhưng 1 số xử lý hậu kỳ, ví dụ như cắt tỉa thời lượng tối thiểu, sẽ giúp loại bỏ các lỗi này. Cũng lưu ý xu hướng rút ngắn các nốt nhạc kéo dài có thể là do nội dung phổ khác nhau trong quá trình nhả nốt.

- 5.2. Objective evaluation. For a more rigorous evaluation, considered standards of Music Information Retrieval Evaluation eXchange (MIREX) [2] & focused on 2 subtasks: (1) a frame-level estimation of present events in terms of musical pitch, & (2) a note-level tracking of present notes in terms of musical pitch, onset & offset times.

  – Đánh giá khách quan. Đối với 1 đánh giá nghiêm ngặt hơn, các tiêu chuẩn được xem xét của Trao đổi đánh giá truy xuất thông tin âm nhạc (MIREX) [2] & tập trung vào 2 nhiệm vụ phụ: (1) ước tính cấp khung hình của các sự kiện hiện tại về cao độ âm nhạc, & (2) theo dõi cấp nốt của các nốt hiện tại về cao độ âm nhạc, thời gian bắt đầu & bù.

  For evaluation dataset, chose MIDI-Aligned Piano Sounds (MAPS) database [10]. MAPS contains real recordings of piano pieces with ground-truth references. Selected 25 pieces & truncated each of them to 30s.

  – Đối với tập dữ liệu đánh giá, chúng tôi đã chọn cơ sở dữ liệu Âm thanh Piano Căn chỉnh MIDI (MAPS) [10]. MAPS chứa các bản ghi âm thực tế của các bản nhạc piano có tham chiếu thực tế. Chúng tôi đã chọn 25 bản nhạc & cắt bớt mỗi bản nhạc xuống còn 30 giây.

  Concerning parameters β was set to 0.5. Thresholds for detection were set empirically to 1 & 2 for frame & note levels resp. Minimum duration for pruning was set to 50 ms. Templates were learned from MAPs with same representation front-end as above. This algorithm is referenced by BND.

  – Về các tham số β được đặt thành 0,5. Ngưỡng phát hiện được đặt theo kinh nghiệm thành 1 & 2 cho các mức khung & ghi chú tương ứng. Thời gian cắt tỉa tối thiểu được đặt thành 50 ms. Các mẫu được học từ các MAP có cùng giao diện biểu diễn như trên. Thuật toán này được tham chiếu bởi BND.

  In addition, tested system with standard Euclidean decomposition algorithm referenced by END, & with sparse algorithm of [14] with projection onto cone of sparsity $s = 0.9$. For these 2 algorithms, detection thresholds were set to 2 & 4 for frame & note levels resp. To compare results, also performed evaluation for 2 offline systems at state-of-art: 1 based on NMF but an harmonic model & spectral smoothness [23], & another one based on a sinusoidal analysis with a candidate selection exploiting spectral features [25].

  – Ngoài ra, hệ thống được thử nghiệm với thuật toán phân tích Euclidean chuẩn được tham chiếu bởi END, & với thuật toán thưa thớt của [14] với phép chiếu lên hình nón có độ thưa $s = 0,9$. Đối với 2 thuật toán này, ngưỡng phát hiện được đặt thành 2 & 4 cho các mức khung & nốt tương ứng. Để so sánh kết quả, chúng tôi cũng đã thực hiện đánh giá cho 2 hệ thống ngoại tuyến ở mức hiện đại: 1 dựa trên NMF nhưng là mô hình điều hòa & độ mịn phổ [23], & 1 hệ thống khác dựa trên phân tích hình sin với lựa chọn ứng viên khai thác các đặc điểm phổ [25].

  Report evaluation results per algorithm in Table 1: Frame-level transcription results per algorithm. Table 2: Note-level transcription results per algorithm at frame & note levels resp. Standard evaluation metrics from MIREX are used as described in [2]: precision $\mathcal{P}$, recall $\mathcal{R}$, F-measure $\mathcal{F}$, accuracy $\mathcal{A}$, total error $\mathcal{E}_{\text{tot}}$, substitution error E_{subs}, missed error $\mathcal{E}_{\text{miss}}$, false alarm error $\mathcal{E}_{\text{fa}}$, mean overlap ratio $\mathcal{M}$. At note level, subscripts 1 & 2 represent resp onset-based & onset/offset-based results.

  – Báo cáo kết quả đánh giá theo từng thuật toán trong Bảng 1: Kết quả phiên mã cấp khung theo từng thuật toán. Bảng 2: Kết quả phiên mã cấp nốt theo từng thuật toán ở cấp khung & cấp nốt tương ứng. Các số liệu đánh giá tiêu chuẩn từ MIREX

được sử dụng như mô tả trong [2]: độ chính xác $\mathcal{P}$, độ thu hồi $\mathcal{R}$, độ đo F $\mathcal{F}$, độ chính xác $\mathcal{A}$, tổng lỗi $\mathcal{E}_{\text{tot}}$, lỗi thay thế E_{subs}, lỗi bỏ lỡ $\mathcal{E}_{\text{miss}}$, lỗi báo động giả $\mathcal{E}_{\text{fa}}$, tỷ lệ chồng lấp trung bình $\mathcal{M}$. Ở cấp nốt, chỉ số dưới 1 & 2 tương ứng biểu thị kết quả dựa trên & dựa trên độ lệch bắt đầu.

Overall, results show: proposed real-time system performs comparably to state-of-art offline algorithms of [23, 25]. Using β-divergence, system BND even outperforms other algorithms. Sparse algorithm of [14] reduces insertions & substitutions, but augments number of missed notes so that it actually does not perform better than standard scheme END. Standard Euclidean cost also shows its limits for transcription where more complex costs with β-divergence give better results. Mean overlap ratio scores corroborate observation that sustained notes tend to be shortened.

– Nhìn chung, kết quả cho thấy: hệ thống thời gian thực được đề xuất hoạt động tương đương với các thuật toán ngoại tuyến tiên tiến của [23, 25]. Sử dụng β-divergence, hệ thống BND thậm chí còn vượt trội hơn các thuật toán khác. Thuật toán thưa thớt của [14] làm giảm số lần chèn & thay thế, nhưng lại tăng số lượng nốt bị bỏ sót nên thực tế nó không hoạt động tốt hơn lược đồ END tiêu chuẩn. Chi phí Euclidean tiêu chuẩn cũng cho thấy giới hạn của nó đối với việc phiên âm, trong khi chi phí phức tạp hơn với β-divergence cho kết quả tốt hơn. Điểm số tỷ lệ chồng chéo trung bình xác nhận quan sát rằng các nốt kéo dài có xu hướng bị rút ngắn.

- 6. Conclusion. Address problem of real-time polyphonic music transcription by employing NMF techniques. Discussed use of β-divergence as a cost function for nonnegative decomposition tailored to real-time transcription. Obtained results show: proposed system can outperform state-of-art offline approaches, & are encouraging for further development.

  – Giải quyết vấn đề phiên âm nhạc đa âm thời gian thực bằng cách sử dụng kỹ thuật NMF. Thảo luận về việc sử dụng β-divergence làm hàm chi phí cho phân tích phi âm được điều chỉnh theo phiên âm thời gian thực. Kết quả thu được cho thấy: hệ thống đề xuất có thể vượt trội hơn các phương pháp ngoại tuyến hiện đại, & rất đáng khích lệ để phát triển thêm.

  A problem in our approach: templates are inherently considered as stationary. 1 way to tackle this: consider representations that capture variability over a short time-span as in [7]. Could also combine NMF with a state representation & use templates for each state.

  – 1 vấn đề trong cách tiếp cận của chúng tôi: các mẫu vốn được coi là tĩnh. 1 cách để giải quyết vấn đề này: xem xét các biểu diễn nắm bắt tính biến thiên trong 1 khoảng thời gian ngắn như trong [7]. Cũng có thể kết hợp NMF với biểu diễn trạng thái & sử dụng các mẫu cho từng trạng thái.

  Template learning method can be further improved by using extended NMF problems & algorithms to learn 1 or more templates for each note. Such issues have not been developed but interesting perspectives include learning sparse or harmonic templates. Using β-divergence during template learning in our experience did not improve results. Further considerations are needed on this line.

  – Phương pháp học mẫu có thể được cải thiện hơn nữa bằng cách sử dụng các bài toán NMF mở rộng & thuật toán để học 1 hoặc nhiều mẫu cho mỗi nốt. Những vấn đề như vậy chưa được phát triển, nhưng những khía cạnh thú vị bao gồm việc học các mẫu thưa thớt hoặc hài hòa. Theo kinh nghiệm của chúng tôi, việc sử dụng β-divergence trong quá trình học mẫu không cải thiện kết quả. Cần xem xét thêm về vấn đề này.

  In a live performance setup e.g. ours, templates can be directly learned from corresponding instrument. Yet in other setups, issue of generalization must be carefully considered & will be discussed in future work. Think of considering adaptive templates by adapting an approach proposed in [13] to real-time decomposition.

  – Trong thiết lập biểu diễn trực tiếp, ví dụ như của chúng tôi, các mẫu có thể được học trực tiếp từ nhạc cụ tương ứng. Tuy nhiên, trong các thiết lập khác, vấn đề khái quát hóa cần được cân nhắc kỹ lưỡng & sẽ được thảo luận trong các nghiên cứu tiếp theo. Hãy cân nhắc việc xem xét các mẫu thích ứng bằng cách áp dụng phương pháp được đề xuất trong [13] cho phân tích thời gian thực.

  Would like also to improve robustness against noise, by keeping information from activations during template learning, or by using noise templates as in [7]. In addition, want to develop more elaborate sparsity controls than in [6, 7, 14]. In our approach, sparsity is controlled implicitly during decomposition. Yet in some applications, specially for complex problems e.g. auditory scene analysis, controlling explicitly sparsity becomes crucial. Proposed system is currently under development for Max/MSP real-time computer music environment & will be soon available for free download on companion website.

  – Cũng muốn cải thiện khả năng chống nhiễu, bằng cách giữ thông tin khỏi các kích hoạt trong quá trình học mẫu, hoặc bằng cách sử dụng các mẫu nhiễu như trong [7]. Ngoài ra, muốn phát triển các điều khiển độ thưa thớt tinh vi hơn so với trong [6, 7, 14]. Theo cách tiếp cận của chúng tôi, độ thưa thớt được điều khiển ngầm trong quá trình phân tích. Tuy nhiên, trong 1 số ứng dụng, đặc biệt đối với các vấn đề phức tạp, ví dụ như phân tích cảnh thính giác, việc điều khiển độ thưa thớt 1 cách rõ ràng trở nên rất quan trọng. Hệ thống đề xuất hiện đang được phát triển cho môi trường âm nhạc máy tính thời gian thực Max/MSP & sẽ sớm có sẵn để tải xuống miễn phí trên trang web đồng hành.

1.2 Renato Fabbri, Vilson Vieira de Silva Junior, Antônio Carlos Silvano Pessotti, Débora Cristina Corrêa, Osvaldo N. Oliveira Jr. **Musical Elements in Discrete-Time Representation of Sound**

[2 citations]

- Abstract. Representation of basic elements of music in terms of discrete audio signals is often used in software for musical creation & design. Nevertheless, there is no unified approach that relates these elements to discrete samples of digitized sound. In this article, each musical element is related by equations & algorithms to discrete-time samples of sounds, & each of these relations are implemented in scripts within a software toolbox, referred to as MASS (Music & Audio in Sample Sequences). Fundamental element, musical note with duration, volume, pitch, & timbre, is related quantitatively to characteristics of digital signal. Internal variations of a note, e.g. tremolos, vibratos, & spectral fluctuations, are also considered, which enables synthesis of notes inspired by real instruments & new sonorities. With this representation of notes, resources are provided for generation of higher scale musical structures, e.g. rhythmic meter, pitch intervals & cycles. This framework enables precise & trustful scientific experiments, data sonification & is useful for education & art. Efficacy of MASS is confirmed by synthesis of small musical pieces using basic notes, elaborated notes & notes in music, which reflects organization of toolbox & thus of this article. Possible to synthesize whole albums through collage of scripts & settings specified by user. With open source paradigm, toolbox can promptly scrutinized, expanded in co-authorship processes & used with freedom by musicians, engineers, & other interested parties. In fact, MASS has already been employed for diverse purposes which include music production, artistic presentations, psychoacoustic experiments & computer language diffusion where appeal of audiovisual artifacts is exploited for education.

- CCS Concepts: Applied computing → Sound & music computing. Computing methodologies → Modeling methodologies. General & reference → Surveys & overviews, Reference works.

- Additional Key Words & Phrases: music, acoustics, psychophysics, digital audio, signal processing.

- 1. Introduction. Music is usually defined as art whose medium is sound. Definition might also state: medium includes silences & temporal organization of structures, or music is also a cultural activity or product. In physics & in this document, sounds are longitudinal waves of mechanical pressure. Human auditory system perceives sounds in frequency bandwidth between 20Hz & 20kHz, with actual boundaries depending on person, climate conditions & sonic characteristics themselves. Since speed of sound ≈ 343.2 m/s, such frequency limits corresponds to wavelengths of $\frac{343.2}{20} \approx 17.6$ m & $\frac{343.2}{20000} \approx 17.16$ mm. Hearing involves stimuli in bones, stomach, ears, transfer functions of head & torso (thân mình), & processing by nervous system. Ear is a dedicated organ or appreciation of these waves, which decomposes them into their sinusoidal spectra & delivers to nervous system. Sinusoidal components are crucial to musical phenomena, as one can recognize in constitution of sounds of musical interest (e.g. harmonic sounds & noises, discussed in Sects. 2–3), & higher level musical structures (e.g. tunings, scales, & chords, Sect. 4) [55]

  Representation of sound can take many forms, from musical scores & texts in a phonetic language to electric analog signals & binary data. It includes sets of features e.g. wavelet or sinusoidal components. Although terms 'audio' & 'sound' are often used without distinction & 'audio' has many definitions which depend on context & author, audio most often means a representation of amplitude through time. In this sense, audio expresses sonic waves yield by synthesis or input by microphones, although these sources are not always neatly distinguishable e.g. as captured sounds are processed to generate new sonorities (âm thanh). Digital audio protocols often imply in quality loss (to achieve smaller files, ease storage & transfer) & are called *lossy* [47]. This is case e.g. of MP3 & Ogg Vorbis. Non-lossy representations of digital audio, called *lossless* protocols or formats, on other hand, assures perfect reconstruction of analog wave within any convenient precision. Standard paradigm of lossless audio consists of representing sound with samples equally spaced by a duration δ_s, & specifying amplitude of each sample by a fixed number of bits. This is linear Pulse Code Modulation (LPCM) representation of sound, herein referred to as PCM. A PCM audio format has 2 essential attributes: a sampling frequency $f_s = \frac{1}{\delta_s}$ (also called e.g. sampling rate or sample rate), which is number of samples used for representing a second of sound; & a bit depth, which is number of bits used for specifying amplitude of each sample. Fig. 1. Example of PCM audio: a sound wave is represented by 25 samples equally spaced in time where each sample has an amplitude specified with 4 bits. shows 25 samples of a PCM audio with a bit depth of 4, which yields $2^4 = 16$ possible values for amplitude of each sample & a total of $4 \cdot 25 = 100$ bits for representing whole sound.

  Fixed sampling frequency & bit depth yield quantization error or quantization noise. This noise diminishes as bit depth increases while greater sampling frequency allows higher frequencies to be represented. Nyquist theorem asserts: sampling frequency is twice maximum frequency: represented signal can contain [49]. Thus, for general musical purposes, suitable to use a sample rate of at least twice highest frequency heard by humans, i.e., $f_s \geq 2 \cdot 20$ kHz $= 40$ kHz. This is basic reason for adoption of sampling frequencies e.g. 44.1 kHz & 48 kHz, which are standards in Compact Disks (CD) & broadcast systems (radio & television), resp.

  Within this framework for representing sounds, musical notes can be characterized. Note often stands as 'fundamental unit' of musical structures (e.g. atoms in matter or cells in macroscopic organisms) &, in practice, it can unfold into sounds that uphold other approaches to music. This is of capital importance because science & scholastic artists widened traditional comprehension of music in 20th century to encompass discourse without explicit rhythm, melody or harmony. This is evident, e.g., in concrete, electronic, electroacoustic, & spectral musical styles. In 1990s, it became evident: popular (commercial) music had also incorporated sound amalgrams & abstract discursive arcs. [There are well known incidences of such characteristics in ethnic music, e.g. in Pygmy music, but western theory assimilated them only in last century [74].] Notes are also convenient for another reason: average listener – & a considerable part of specialists – presupposes rhythmic & pitch organization (made explicit in Sect. 4) as fundamental musical properties, & these are developed in traditional musical theory in terms of notes. Thereafter, in this article describe musical notes in PCM audio through equations & then indicate mechanisms for deriving higher level musical structures. Understand: this is not unique approach to mathematically express music in digital audio, but musical theory & practice suggest: this is a proper framework for understanding & making computer music, as should become

patent in reminder of this text & is verifiable by usage of MASS toolbox. Hopefully, interested reader or programmer will be able to use this framework to synthesize music beyond traditional conceptualizations when intended.

This document provides a fundamental description of musical structures in discrete-time audio. Results include mathematical relations, usually in terms of musical characteristics & PCM samples, concise musical theory considerations, & their implementation as software routines both as very raw & straightforward algorithms & in context of rendering musical pieces. Despite general interests involved, there are only a few books & computer implementations that tackle subject directly. These mainly focus on computer implementations & way to mimic traditional instruments, with scattered mathematical formalisms for basic notations. Articles on topic appear to be lacking, to best of our knowledge, although advanced & specialized developments are often reported. A compilation of such works & their contributions is in Appendix G of [21]. Although current music software uses analytical descriptions presented here, there is no concise mathematical description of them, & far from trivial to achieve equations by analyzing available software implementations.

Accordingly, objectives of this paper:

1. Present a concise set of mathematical & algorithmic relations between basic musical elements & sequences of PCM audio samples.
2. Introduce a framework for sound & musical synthesis with control at sample level which entails potential uses in psychoacoustic experiments, data sonification & synthesis with extreme precision (recap in Sect. 5).
3. Provide a powerful theoretical framework which can be used to synthesize musical pieces & albums.
4. Provide approachability to developed framework [All analytic relations presented in this article are implemented as small scripts in public domain. They constitute MASS toolbox, available in an open source Git repository [9]. These routines are written in Python & make use of Numpy, which performs numerical routines efficiently (e.g. through LAPACK), but language & packages are by no means mandatory. Part of scripts has been ported to JavaScript (which favors their use in Web browsers e.g. Firefox & Chromium) & native Python [48, 56, 70]. These are all open technologies, published using licenses that grant permission for copying, distributing, modifying & usage in research, development, art & education. Hence, work presented here aims at being compliant with recommended practices for availability & validation & should ease co-authorship processes [43, 52].]
5. Provide a didactic (mang tính giáo huấn) presentation of content, which is highly multidisciplinary, involving signal processing, music, psychoacoustics & programming.

Reminder of this article is organized as follows: Sect. 2 characterizes basic musical note; Sect. 3 develops internal dynamics of musical notes; Sect. 4 tackles organization of musical notes into higher level musical structures [14, 41, 42, 54, 62, 72, 74, 76]. As these descriptions require knowledge on topics e.g. psychoacoustics, cultural traditions, & mathematical formalisms, text points to external complements as needed & presents methods, results, & discussions altogether. Sect. 5 is dedicated to final considerations & further work.

- 1.1. Additional material. 1 Supporting Information document [27] holds commented listings of all equations, figures, tables, & sects in this document & scripts in MASS toolbox. Another Supporting Information document [28] is a PDF version of code that implements equations & concepts in each sect [Toolbox contains a collection of Python scripts which
  - implements each of equations
  - render music & illustrate concepts
  - render each of figures used in this article.

  Documentation of toolbox consists of this article, Supporting Information documents & scripts themselves.]. Git repository [26] holds all PDF documents & Python scripts. Rendered musical pieces are referenced when convenient & linked directly through URLs, & constitute another component of framework. They are not very traditional, which facilitates understanding of specific techniques & extrapolation of note concept. There are MASS-based software packages [23, 25] & further musical pieces that are linked in Git repository.
- 1.2. Synonymy, polysemy & theoretical frames (disclaimer). Given: main topic of this article (expression of musical elements in PCM audio) is multidisciplinary & involves art, reader should be aware: much of vocabulary admits different choices of terms & defs. More specifically, often case where many words can express same concept & where 1 word can carry different meanings. This is a very deep issue which might receive a dedicated manuscript. Reader might need to read rest of this document to understand this small selection of synonymy & polysemy (đa nghĩa) in literature, but important to illustrate point before more dense sects:
  - a "note" can mean a pitch or an abstract construct with pitch & duration or a sound emitted from a musical instrument or a specific note in a score or a music.
  - Sampling rate is also called *sampling frequency* or *sample rate*.
  - A harmonic in a sound is most often a sinusoidal component which is in harmonic series of fundamental frequency. Many times, however, terms harmonic & component are not distinguished. A harmonic can also be a note performed in an instrument by preventing certain overtones (components).
  - Harmony can refer to chords or to note sets related to chords or even to "harmony" in a more general sense, as a kind of balance & consistency.

∗ A "tremolo" can mean different things: e.g. in a piano score, a tremolo is a fast alternation of 2 notes (pitches) while in computer music theory it is (most often) an oscillation of loudness.

Strived to avoid nomenclature clashes & use of more terms than needed. Also, there are many theoretical standpoints for understanding musical phenomena, which is an evidence: most often there is not a single way to express or characterize musical structures. Therefore, in this article, adjectives e.g. "often", "commonly", & "frequently" are abundant & they would probably be even more numerous if wanted to be pedantically precise. Some of these issues are exposed when content is convenient, e.g. in 1st considerations of timbre.

– Cố gắng tránh xung đột danh pháp & sử dụng nhiều thuật ngữ hơn mức cần thiết. Ngoài ra, có nhiều quan điểm lý thuyết để hiểu các hiện tượng âm nhạc, đây là bằng chứng: thường không có 1 cách duy nhất để diễn đạt hoặc mô tả các cấu trúc âm nhạc. Do đó, trong bài viết này, các tính từ như "thường xuyên", "thường xuyên", & "thường xuyên" rất nhiều & chúng có thể còn nhiều hơn nữa nếu muốn chính xác về mặt học thuật. 1 số vấn đề này được nêu ra khi nội dung thuận tiện, ví dụ như trong những cân nhắc đầu tiên về âm sắc.

- 2. Characterization of musical note in discrete-time audio. In diverse artistic & theoretical contexts, music is conceived as constituted by fundamental units referred to as notes, "atoms" that constitute music itself [44, 72, 74]. In a cognitive perspective, notes are understood as discernible elements that facilitate & enrich transmission of information through music [41, 55]. Canonically, basic characteristics of a musical note are duration, loudness, pitch, & timbre (âm sắc) [41]. All relations described in this sect are implemented in file `src/sections/eqs2.1.py`. Musical pieces *5 sonic portraits & reduced-fi* are also available online to corroborate & illustrate concepts.

  - 2.1. Duration. Sample frequency f_s is defined as number of samples in each sec of discrete-time signal. Let $T = \{t_i\}$ be an ordered set of real samples separated by $\delta_s = \frac{1}{f_s}$ secs ($f_s = 44.1$ kHz $\Rightarrow \delta_s = \frac{1}{44100} \approx 0.023$ ms). A musical note of duration Δ secs can be expressed as a sequence T^Δ with $\Lambda = \lfloor \Delta \cdot f_s \rfloor$ samples. I.e., integer part of multiplication is considered, & an error of $\leq \delta_s$ missing secs is admitted, which is usually fine for musical purposes. Thus

$$T^\Delta = \{t_i\}_{i=0}^{\lfloor \Delta f_s \rfloor - 1} = \{t_i\}_0^{\Lambda - 1}.$$

  - 2.2. Loudness. Loudness [Loudness & "volume" are often used indistinctly. In technical contexts, loudness is used for subjective perception of sound intensity while volume might be used for some measurement of loudness or to a change in intensity of signal by equipment. Accordingly, one can perceive a sound as loud or soft & change volume by turning a knob. Will use term loudness & avoid more ambiguous term volume.] is a perception of sonic intensity that depends on reverberation, spectrum, & other characteristics described in Sect. 3 [11]. One can achieve loudness variations through power of wave [11]:

$$\mathrm{pow}(T) = \frac{\sum_{i=0}^{\Lambda-1} t_i^2}{\Lambda}.$$

    Final loudness is dependent on amplification of signal by speakers. Thus, what matters: relative power of a note in relation to the others around it, or power of a musical sect in relation to the rest. Differences in loudness are result of complex psychophysical phenomena but can often be reasoned about in terms of decibels, calculated directly from amplitudes through energy or power:

$$V_{\mathrm{dB}} = 10 \log_{10} \frac{\mathrm{pow}(T')}{\mathrm{pow}(T)}.$$

  - 2.3. Pitch.
  - 2.4. Timbre.
  - 2.5. Spectra of sampled sounds.
  - 2.6. Basic note.
  - 2.7. Spatialization: localization & reverberation.
  - 2.8. Musical usages.

- 3. Variation in Basic Note.
  - 3.1. Lookup table.
  - 3.2. Incremental variations of frequency & intensity.
  - 3.3. Application of digital filters.
  - 3.4. Noise.
  - 3.5. Tremolo & vibrato, AM & FM.
  - 3.6. Musical usages.

- 4. Organization of notes in music.
  - 4.1. Tuning, intervals, scales, & chords.
  - 4.2. Atonal & tonal harmonies, harmonic expansion & modulation.

- 4.3. Counterpoint.
- 4.4. Rhythm.
- 4.5. Repetition & variation: motifs & larger units.
- 4.6. Directional structures.
- 4.7. Cyclic structures.
- 4.8. Serialism & post-serial techniques.
- 4.9. Musical idiom?
- 4.10. Musical usages.

- 5. Conclusions & Further Developments.

1.3 [HWR22]. Michael S. Horn, Melanie West, Cameron Roberts. Introduction to Digital Music with Python Programming: Learning Music with Code

[4 Amazon ratings]

- Amazon reviews. *Introduction to Digital Music with Python Programming* provides a foundation in music & code for beginner. It shows how coding empowers new forms of creative expression while simplifying & automating many of tedious aspects of production & composition.

  With help of online, interactive examples, this book covers fundamentals of rhythm, chord structure, & melodic composition alongside basics of digital production. Each new concept is anchored in a real-world musical example that will have you making beats in a matter of minutes.

  Music is also a great way to learn core programming concepts e.g. loops, variables, lists, & functions, *Introduction to Digital Music with Python Programming* is designed for beginners of all backgrounds, including high school students, undergraduates, & aspiring professionals, & requires no prev experience with music or code.

  A beginner's approach to digital music production focuses on key concepts, ensuring ease & progress in learning.

  Streamline your programming education by incorporating music, making complex core concepts easier to grasp & apply.

  Amplify your music creativity by generating unique beats with code in minutes, without needing advanced technical skills.

  A great book for learning Python programming & exploring digital music.

  This broad manual combines music theory & programming basics, providing interactive examples & real-world applications to help you compose & produce music from scratch.

  Perfect for aspiring musicians & programmers exploring music-code fusion.

- About Author. Michael S. Horn is Associate Prof of CS & Learning Sciences at Northwestern University in Evanston, Illinois, where he directs Tangible Interaction Design & Learning (TIDAL) Lab.

  Melanie West is a PhD student in Learning Sciences at Northwestern University & co-founder of Tiz Media Foundation, a nonprofit dedicated to empowering underrepresented youth through science, technology, engineering, & mathematics (STEM) programs.

  Cameron Roberts is a software developer & musician living in Chicago. He holds degrees from Northwestern University in Music Performance & CS.

- Foreword. When I was a kid growing up in Texas, I "learned" how to play viola. I put *learned* in quotes because it was really just a process of rote memorization – hours & hours of playing same songs over & over again. I learned how to read sheet music, bu only to extent that I knew note names & could translate them into grossest of physical movements. I never learned to read music as literature, to understand its deeper meaning, structure, or historical context. I never understood anything about music theory beyond being annoyed that I had to pay attention to accidentals in different keys. I never composed *anything*, not even informally scratching out a tune. I never developed habits of deep listening, of taking songs apart in my head & puzzling over how they were put together in 1st place. I never played just for fun. &, despite best intentions of parents & teachers, I never fell in love with music.

  Learning how to code was complete opposite experience for me. I was largely self-taught. Courses I took in school were electives (môn tự chọn) that I chose for myself. Teachers gave me important scaffolding at just right times, but it never felt forced. I spent hours working on games or other projects (probably when I should have been practicing viola). I drew artwork, planned out algorithms, & even synthesized my own rudimentary sound effects (hiệu ứng âm thanh thô sơ). I had no idea what I was doing, but that was liberating. No one was around to point out my mistakes or to show me ow to do things "right" way (at least, not until college). I learned how to figure things out for myself, & skills I picked up from those experiences are still relevant today. I fell in love with coding. [I was also fortunate to have grown up in a time & place where these activities were seen as socially acceptable for a person of my background & identity.]

  But I know many people whose stories are flipped 180 degrees. For them, music was so personally, socially, & culturally motivating that they couldn't get enough. They'd practice for hours & hours, not just for fun but for something much deeper.

For some it was an instrument like guitar that got them started. For others it was an app like GarageBand that gave them a playful entry point into musical ideas. To extent that they had coding experiences, those experiences ranged from uninspiring to off-putting (từ không hấp dẫn đến khó chịu). It's not that they necessarily hated coding, but it was something they saw as not being for them.

In foreword of his book, *Mindstorms: Children, Computers, & Powerful Ideas*, Seymour Papert wrote: "fell in love with gears" as a way of helping us imagine a future in which children (like me) would fall in love with computer programming, not for its own sake, but for creative worlds & powerful ideas that programming could open up. Part of what he was saying was: love & learning go hand in hand, & that computers could be an entry point into many creative & artistic domains e.g. mathematics & music. Coding can revitalize subjects that have become painfully rote in schools.

Process of developing TunePad over past several years has been a fascinating rediscovery of musical ideas for me. Code has given me a different kind of language for thinking about things like rhythm, chords, & harmony. I can experiment with composition unencumbered by my maladroit hands. Music has become something creative & alive in a way that it never was for me before. Music theory is no longer a thicket of confusing terminology & instead has become a fascinating world of mathematical beauty that structures creative process.

– Quá trình phát triển TunePad trong nhiều năm qua là 1 sự khám phá lại đầy hấp dẫn đối với tôi về các ý tưởng âm nhạc. Mã đã cho tôi 1 loại ngôn ngữ khác để suy nghĩ về những thứ như nhịp điệu, hợp âm, & sự hòa âm. Tôi có thể thử nghiệm sáng tác mà không bị cản trở bởi đôi tay vụng về của mình. Âm nhạc đã trở thành 1 thứ gì đó sáng tạo & sống động theo cách mà trước đây tôi chưa từng có. Lý thuyết âm nhạc không còn là 1 mớ thuật ngữ khó hiểu & thay vào đó đã trở thành 1 thế giới hấp dẫn của vẻ đẹp toán học cấu trúc nên quá trình sáng tạo.

Melanie, Cameron, & I hope: this book gives a similarly joyful learning experience with music & code. Hope: feel empowered to explore algorithmic & mathematical beauty of music. Hope: discover, as we have: music & code reinforce one another in surprising & powerful ways that open new creative opportunities for you. Hope, regardless of your starting point – as a coder, as a musician, as neither, as both – will discover something new about yourself & what you can become.

- 1. Why music & coding? This book is designed for people who *love* music & are interested in intersection of music & coding. Maybe you're an aspiring musician or music producer who wants to know more about coding & what it can do. Or maybe already know a little about coding, & want to expand your creative musical horizon. Or maybe a total beginner in both. Regardless of your starting point, this book is designed for you to learn about music & coding as mutually reinforcing skills. Code gives us an elegant language to think about musical ideas, & music gives us a context within which code makes sense & is immediately useful. Together they form a powerful new way to create music that will be interconnected with digital production tools of future.

  More & more code will be used to produce music, to compose music, & even to perform music for live audiences. Digital production tools e.g. Logic, Reason, Pro Tools, FL Studio, & Ableton Live are complex software applications created with *millions* of lines of code written by huge teams of software engineers. With all of these tools can write code to create custom plugins & effects. Beyond production tools, live coding is an emerging from of musical performance art in which Information Age DJs write computer code to generate music in real time for live audiences.

  In other ways, still on cusp of a radical transformation in way use code to create music. History of innovation in music has always been entwined with innovations in technology. Whether talking about Franz Liszt in 19th century, who pioneered persona of modern music virtuoso based on technological breakthroughs of piano [Fans were so infatuated with Liszt's piano "rockstar" status that they fought over his silk handkerchiefs & velvet gloves at his performances.], or DJ Kool Herc in 20th century, who pioneered hip-hop with 2 turntables & a crate full of funk records in Bronx, technologies have created new opportunities for musical expression that have challenged status quo & given birth to new genres. Don't have Franz Liszt or DJ Kool Herc of coding yet, but it's only a matter of time before coding virtuosos of tomorrow expand boundaries of what's possible in musical composition, production, & performance.

  - 1.1. What is Python? In this book learn how to create your own digital music using a computer programming language called *Python*. If not familiar with programming languages, Python is a general-purpose language 1st released in 1990s that is now 1 of most widely used languages in world. Python is designed to be easy to read & write, which makes it a popular choice for beginners. Also fully featured & powerful, making it a good choice for professionals working in fields as diverse as DS, web development, arts, & video game development. Because Python has been around for decades, it runs on every major computer OS. Examples in this book even use a version of Python that runs directly inside of your web browser without need for any special software installation.

    Unlike many other common beginner programming languages, Python is "text-based", i.e., type code into an editor instead of dragging code blocks on computer screen. This makes Python a little harder to learn than other beginner languages, but it also greatly expands what you can do. By time yet through this book should feel comfortable writing short Python programs & have conceptual tools need to explore more on your own.

  - 1.2. What this book is *not*. Before get into a concrete example of what you can do with a little bit of code, just a quick note about what this book is *not*. This book is not a comprehensive guide to Python programming. There are many excellent books & tutorials designed for beginners, several of which are free. [Recommend https://www.w3schools.com/python/.]

    This book is also not a comprehensive guide to music theory or Western music notation. Get into core ideas behind rhythm, harmony, melody, & composition, but there are, again, many other resources available for beginners who want to go deeper. What offering is a different approach that combines learning music with learning code in equal measure.

- 1.3. What this book *is*. What will do is give an intuitive understanding of fundamental concepts behind both music & coding. Code & music are highly technical skills, full of arcane symbols & terminology, seem almost designed to intimidate beginners. In this book put core concepts to use immediately to start making music. Get to play with ideas at your own pace & get instant feedback as bring ideas to life. Skip most of technical jargon & minutiae for now – can come later. Instead, focus on developing your confidence & understanding. Importantly, skills, tools, & ways of thinking introduce in this book will be broadly applicable in many other areas as well. Working in Python code, but core structures of variables, functions, loops, conditional logical, & classes are same across many programming languages including JavaScript, Java, C, C++, & C#. After learn 1 programming language, each additional language is that much easier to pick up.
- 1.4. TunePad & EarSketch. This book uses 2 free online platforms that combine music & Python coding. 1st, called TunePad https://tunepad.com, was developed by a team of researchers at Northwestern University in Chicago. TunePad lets create short musical loops that you can layer together using a simple digital audio workstation (DAW) interface. 2nd platform, called EarSketch https://earsketch.gatech.edu, was created by researchers at Georgia Tech in Atlanta. EarSketch uses Python code to arrange samples & loops into full-length compositions. Both platforms are browser-based apps, so all need to get started is a computer (tablets or Chromebooks are fine), an internet connection, & a web browser like Chrome or Firefox. External speakers or headphones are also nice but not required. Both platforms have been around for years & have been used by many thousands of students from middle school all way up to college & beyond. TunePad & EarSketch are designed primarily as learning platforms, but there are easy ways to export your work to professional production software if want to go further.
- 1.5. A quick example. A quick example of what coding in Python looks like. This program runs in TunePad to create a simple beat pattern, variants of which have been used in literally thousands of songs e.g. *Blinding Lights* by The Weeknd & *Roses* by SAINt JHN.

```
playNote(1) # play a kick drum sound
playNote(2) # play a snare drum sound
playNote(1)
playNote(2)
rewind(4)   # rewind 4 beats
for i in range(4):
    rest(0.5)
    playNote(4, beats = 0.5) # play hat for a half beat
```

  These 8 lines of Python code tell TunePad to play a pattern of kick drums, snare drums, & high-hats. Most of lines are *playNote* instructions, & those instructions tell TunePad to play musical sounds indicated by numbers inside of parentheses. This example also includes sth called a *loop*. Loop is an easy way to repeat a set of actions over & over again. In this case, loop tells Python to repeat lines 7 & 8 4 times in a row. Screenshot Fig. 1.1: A TunePad program to play a simple rock beat. shows what this looks like in TunePad. Can try out example for yourself with this link: https://tunepad.com/examples/roses.
- 1.6. 5 reasons to learn code. Now seen a brief example of what can do with a few lines of Python code, here are top 5 reasons to get started with programming & music if still in doubts.
  - 1.6.1. Reason 1: Like it or note, music is already defined by code. Looking across modern musical landscape, clear: music is already defined by code. 1 of biggest common factors of almost all modern music from any popular genre: *everything* is edited, if not created entirely, with sophisticated computer software. Hard to overstate how profoundly such software has shaped sound of music in 21st century. Relatively inexpensive DAW applications & myriad ubiquitous plugins that work across platforms have had a disruptive & democratizing effect across music industry. Think about effects plugins like autotune, reverb, or ability to change pitch of a sample without changing tempo. These effects are all generated with sophisticated software. Production studios size of small offices containing hundreds of thousands of dollars' worth of equipment now fit on screen of a laptop computer available to any aspiring producer with passion, a WiFi connection, & a small budget. Reasons behind shift to digital production tools are obvious. Computers have gotten to a point where they are cheap enough, fast enough, & capacious enough to do real-time audio editing. Can convert sound waves into editable digital information with microsecond precision & then hear effects of our changes in real time. These DAWs didn't just appear out of nowhere. They were constructed by huge teams of software engineers writing code – millions of lines of it. E.g., TunePad was created with $>$ 1.5 million lines of code written in over a dozen computer languages e.g. Python, HTML, JavaScript, CSS, & Dart. Regardless of how feel about digital nature of modern music, not going away. Learning to code will help understand a little more about how all of this works under hood. More to point, it's increasingly common for producers to write their own code to manipulate sound. E.g., in Logic, can write JavaScript code to process incoming MIDI (Musical Instrument Digital Interface) data to do things like create custom arpeggiators. Learning to code can give you more control & help expand your creative potential Fig. 1.2: Typical DAW software.
  - 1.6.2. Reason 2: Code is a powerful way to make music. Don't always think about it this way, but music is *algorithmic* in nature – it's full of mathematical relationships, logical structure, & recursive patterns. Beauty of Baroque fugue is in part a reflection of beauty of mathematical & computational ideas behind music. Call Bach a genius not just because his music is so compelling, but also because he was able to hold complex algorithms in his mind & then transcribe them to paper using representation system called Western music notation. I.e., music notation is a language for recording output of composition process, but not a language for capturing algorithmic nature of composition process itself.

Code, on other hand, is a language specifically designed to capture mathematical relationships, logical structure, & recursive patterns. E.g., take stuttered hi-hat patterns that are 1 of defining characteristics of trap music. Here are a few lines of Python code that generate randomized hi-hat stutters that can bring an otherwise conventional beat to life with sparking energy.

```
for _ in range(16):
    if randint(6) > 1: # roll die for a random number
        playNote(4, beats=0.5) # play an 8th note
    else:
        playNote(4, beats=0.25) # or play 16th notes
        playNote(4, beats=0.25)
```

Or, as another example, here's a 2-line Python program that plays a snare drum riser effect common in house, EDM, or pop music. Often hear this technique right before beat drops. This code uses a decay function so that each successive note is a little shorter resulting in a gradual acceleration effect.

```
for i in range(50): # play 50 snares
    playNote(2, beats = pow(2, -0.09 * i))
```

What's cool about these effects: they're *parametrized*. Because code describes algorithms to generate music, & not music itself, i.e., can create infinite variation by adjusting numbers involved. E.g., in trap hi-hat code, can easily play around with how frequently stuttered hats are inserted into pattern by increasing or decreasing 1 number. Can think of code as sth like a power drill; can swap out different bits to make holes of different sizes. Drill bits are like parameters that change what tool does in each specific instance. In same way, algorithms are vastly more general-purpose tools that can accomplish myriad tasks by changing input parameters.

Creating a snare drum riser with code is obviously a very different kind of thing than picking up 2 drumsticks & banging out a pattern on a real drum. &, to be clear, not advocating for code to replace learning how to perform with live musical instruments. But, code can be another tool in your musical repertoire for generating repetitive patterns, exploring mathematical ideas, or playing sequences that are too fast or intricate to play by hand.

– Tạo 1 bộ phận nâng cao trống snare bằng mã rõ ràng là 1 việc rất khác so với việc nhặt 2 dùi trống & đánh 1 mẫu trên 1 chiếc trống thật. &, nói rõ hơn, không ủng hộ việc sử dụng mã để thay thế việc học cách biểu diễn với các nhạc cụ sống. Nhưng mã có thể là 1 công cụ khác trong tiết mục âm nhạc của bạn để tạo ra các mẫu lặp lại, khám phá các ý tưởng toán học hoặc chơi các chuỗi quá nhanh hoặc phức tạp để chơi bằng tay.

* 1.6.3. Reason 3: Code lets you build your own musical toolkit. Becoming a professional in any field is about developing expertise with tools – acquiring equipment & knowing how to use it. Clearly, this is true in music industry, but also true in software. Professional software engineers acquire specialized equipment & software packages. Develop expertise in a range of programming languages & technical frameworks. But, they also build their own specialized tools that they use across projects. In this book, show how to build up your own library of Python functions. Can think of functions as specialized tools that you create to perform different musical tasks. In addition to examples described above, might write a function to generate a chord progression or play an arpeggio, & can use functions again & again across many musical projects.
* 1.6.4. Reason 4: Code is useful for a thousand & 1 other things. Python is 1 of most powerful, multi-purpose languages in world. Used to create web servers & social media platforms as much as video games, animation, & music. Used for research & DS, politics & journalism. Knowing a little Python gives access to powerful ML & AI (AI/ML) techniques that are poised to transform most aspects of human work, including in creative domains e.g. music. Python is both a scripting language & a software engineering platform – equal parts duct tape & table saw – & it's capable of everything from quick fixes to durable software applications. Learning a little Python won't make you a software engineer, just like learning a few guitar chords won't make you a performance musician. But it's a start down a path. An open door that was previously closed, & a new way of using your mind & a new way of thinking about music.
* 1.6.5. Reason 5: Coding makes us more human. When think about learning to code, tend to think about economic payoff. Hear arguments that learning to code is a resume builder & a path to a high-paying job. Not that this perspective is wrong, but it might be wrong reason for you to learn how to code.

  Just like people who are good at music *love* music, people who are good at coding tend to *love* coding. Craft of building software can be tedious & frustrating, but it can also be rewarding. A way to express oneself creatively & to engage in craftwork. People don't learn to knit, cook, or play an instrument for lucrative (có lợi nhuận) career paths that these pursuits open up – although by all means those pursuits can lead to remarkable careers. People learn these things because they have a *passion* for them. Because they are personally fulfilling. These passions connect us to centuries of tradition; they connect us to communities of teachers, learners, & practitioners; &, in end, they make us more *human*. So when things get a little frustrating – & things always get a little frustrating when learning any worthwhile skill – remember: just like poetry, literature, or music, code is an arts as much as it is a science. &, just like woodworking, knitting, or cooking, code is a craft as much as it is an engineering discipline. Be patient & give yourself a chance to fall in love with coding.

◦ 1.7. Future of music & code. Before get on with book, wanted to leave you with a brief thought about future of technology, music, & code. For as long as there have been people on this planet there has been music. &, as long as there has been music, people have created technology to expand & enhance their creative potential. A drum is a kind of technology – a piece of animal hide stretched across a hollow log & tied in place. It's a polylithic accomplishment, an assembly of parts

that requires skill & craft to make. One must know how to prepare animal hide, to make rope from plant fiber, & to craft & sharpen tools. More than that, one must know how to perform with drum, to connect with an audience, to enchant them to move their bodies through an emotional & rhythmic connection to beat. Technology brings together materials & tools with knowledge. People must have knowledge both to craft an artifact & to wield it. &, over time – over generations – that knowledge is refined as it gets passed down from teacher to student. It becomes stylized & diversified. Tools, artifacts, knowledge, & practice all become sth greater. Sth we call culture.

Again & again world of music has been disrupted, democratized, & redefined by new technologies. Hip-hop was a rebellion against musical status quo fueled by low-cost technologies like recordable cassette tapes, turntables, & 808 drum machines. Early innovators shattered norms of artistic expression, redefining music, poetry, visual art, & dance in process. Inexpensive access to technology coupled with a need for new forms of authentic self-expression was a match to dry tinder of racial & economic oppression.

Hard to overstate how quickly world is still changing as a result of technological advancements. Digital artifacts & infrastructures are so ubiquitous that they have reconfigured social, economic, legal, & political systems; revolutionized scientific research; upended arts & culture; & even wormed their way into most intimate aspects of our personal & romantic lives. Already talked about transformative impact that digital tools have had on world of music in 21st century, but exhilarating (& scary) part: we're on precipice of another wave of transformation in which human creativity will be redefined by AI & ML. Imagine AI accompanists that can improvise harmonies or melodies in real time with human musicians. Or DL algorithms that can listen to millions of songs & innovate music in same genre. Or silicon poets that grasp human language well enough to compose intricate rap lyrics. Or machines with trillions of transistor synapses so complex that they begin to "dream" – inverted ML algorithms that ooze imagery unhinged enough to disturb absinthe slumber of surrealist painters. Now, imagine: this is not speculative science fiction, but reality of our world today. These things are here now & already challenging what we mean by human creativity. What are implications of a society of digital creative cyborgs?

But here's trick: we've always been cyborgs. Western music notation is as much a technology as Python code. Becoming literate in any sufficiently advanced representation system profoundly shapes how think about & perceive world around us. Classical music notation, theory, & practice shaped mind of Beethoven as much as he shaped music with it – so much so that he was still able to compose many of his most famous works while almost totally deaf. BEETHOVEN was a creative cyborg enhanced by technology of Western music notation & theory. Difference: now we've externalized many of cognitive processes into machines that think alongside us. &, increasingly, these tools are available to everyone. How that changes what it means to be a creative human being is anyone's guess.

- 1.8. Book overview. Excited to have you with us on this journey through music & code. A short guide for where go from here. Chaps. 2–3 cover foundations of rhythm, pitch, & harmony. These chaps are designed to move quickly & get you coding in Python early on. Cover Python variables, loops, which both connect directly to musical concepts. Chaps. 4–6 cover foundations of chords, scales, & keys using Python lists, functions, & data structures. Chaps. 7, 8, 10 shift from music composition to music production covering topics e.g. frequency domain, modular synthesis, & other production effects. In Chap. 9, switch to EarSketch platform to talk about how various musical elements are combined to compose full-length songs. Finally, Chap. 11 provides a short overview of history of music & code along with a glimpse of what future might hold. Between each chap, provide a series of short *interludes* that are like step-by-step tutorials to introduce new music & coding concepts.

  A few notes about how to read this book. Any time include Python code, it will be shown in a programming font. Sometimes write code in a table with line numbers so that can refer to specific lines. When introduce new terms, bold word. If get confused by any of programming or music terminology, check out appendices, which contain quick overviews of all of important concepts. Often invite to follow along with online examples. Best way to learn is by doing it yourself, so strongly encourage to try coding in Python online as go through chaps.

- Interlude 1: Basic Pop Beat. In this interlude, get familiar with TunePad interface by creating a basic rock beat in style of songs like *Roses* by SAINt JHN. Can follow along online by visiting https://tunepad.com/interlude/pop-beat

1. **Step 1: Deep listening.** Good to get in habit of deep listening. Deep listening is practice of trying every possible way of listening to sounds. Start by loading a favorite song in a streaming service & listening – really listening – to it. Take song apart element by element. What sounds do you hear? How are they layered together? When do different parts come into track & how do they change over time? Think about how producer balances sounds across frequency spectrum or opens up space for transitions in lyrics. Try focusing on just drums. Can start to recognize individual percussion sounds & their rhythmic patterns?
2. **Step 2: Create a new TunePad project.** Visit https://tunepad.com on a laptop or Chromebook & set up an account. [Recommend using free Google Chrome browser for best overall experience.] After signing in, click on `New Project` button to create an empty project workspace. Your project will look sth like this Fig. 1.3: TunePad project workspace.
3. **Step 3: Kick drums.** In your project window, click on `ADD CELL` button & then select `Drums` Fig. 1.4: Selecting instruments in TunePad. In TunePad can think of a "cell" as an instrument that you can program to play music. Name new instrument "Kicks" & then add this Python code.

```
# play four kick drums
playNote(1)
playNote(1)
```

```
playNote(1)
playNote(1)
```

When done, your project should look sth like Fig. 1.5: Parts of a TunePad cell.

Go ahead & press Play button at top left to hear how this sounds.

Syntax errors. Occasionally your code won't work right & get a red error message box that looks sth like Fig. 1.6: Python syntax error in TunePad. This kind of error message is called a "syntax" error. In this case, code was written as `playnote` as a lowecase "n" instead of an uppercase "N". Can fix this error by changing code to read `playNote` on line 2.

4. **Step 4: Snare drums.** In your project window, click on `ADD CELL` button again & select `Drums`. Now should have 2 drum cells one appearing above the other in your project. Name 2nd instrument "Snare Drums" & then add this Python code.

```
# play 2 snare drums on the up beats only
rest(1) # skip a beat
playNote(2) # play a snare drum sound
rest(1)
playNote(2)
```

Might start to notice text that comes after hashtag symbol # is a special part of your program. This text is called a *comment*, & it's for human coders to help organize & document their code. Anything that comes after hashtag on a line is ignored by Python. Try playing this snare drum cell to hear how it sounds. Can also play kick drum cell at same time to see how they sound together.

5. **Step 5: Hi-hats.** Click on `ADD CELL` button again to add a 3rd drum cell. Change title of this cell to be "Hats" & add code:

```
# play four hats between the kicks and snares
rest(0.5) # rest for half a beat
playNote(4, beats=0.5) # play a hat for half a beat
rest(0.5)
playNote(4, beats=0.5)
rest(0.5)
playNote(4, beats=0.5)
rest(0.5)
playNote(4, beats=0.5)
```

When play all 3 of drum cells together, should hear a basic rock beat pattern: `kick - hat - snare - hat - kick - hat - snare - hat`.

6. **Step 6: Fix your kicks.** Might notice kick drums feel a little heavy in this mix. Can make some space in pattern by resting on up beats (beats 2 & 4) when snare drums are playing. Scroll back up to your `Kick drum cell` & change code to look like this:

```
# play kicks on the down beats only
playNote(1)
rest(1)
playNote(1)
rest(1)
playNote(1)
rest(1)
playNote(1)
rest(0.5) # rest a half beat
playNote(1, beats = 0.5) # half beat pickup kick
```

7. **Step 7: Adding a bass line.** Add a new cell to your project, but this time select `Bass` instead of `Drums`. Once cell is loaded up, change voice to `Plucked Bass` Fig. 1.7: Selecting an instrument's voice in TunePad.

Entering this code to create a simplified bass line in style of *Roses* by SAINt JHN. When done, try playing everything together to get full sound.

```
playNote(5, beats=0.5) # start on low F
playNote(17, beats=0.5) # up an octave
rest(1)

playNote(10, beats=0.5) # A sharp
playNote(22, beats=0.5) # up an octave
rest(1)
```

```
playNote(8, beats=0.5) # G sharp
playNote(20, beats=0.5) # up an octave
rest(0.5)

playNote(8, beats=0.5) # G sharp - G - G
playNote(12, beats=0.5)
playNote(24, beats=0.5)

playNote(10, beats=0.75) # C sharp
playNote(22, beats=0.25) # D sharp
```

- 2. Rhythm & tempo. This chap dives into fundamentals of *rhythm* in music. Start with beat – what it is, how it's measured, & how can visualize beat to compose, edit, & play music. From there provide examples of some common rhythmic motifs from different genres of music & how to code them with Python. Main programming concepts for this chap include loops, variables, calling function, & passing parameter values. This chap covers a lot of ground, but it will give you a solid start on making music with code.
  - 2.1. Beat & tempo. *Beat* is foundation of rhythm in music. Term *beat* has a number of different meanings in music, [Term beat can also refer to main groove in a dance track ("drop the beat") or instrumental music that accompanies vocals in a hip-hop track ("she produced a beat for a new artist") in addition to other meanings.] but this chap uses it to mean a unit of time, or how long an individual note is played – e.g., "rest for 2 beats" or "play a note for half a beat". Based on beat, musical notes are combined in repeated patterns that move through time to make rhythmic sense to our ears.

    Tempo refers to speed at which rhythm moves, or how quickly 1 beat follows another in a piece of music. As a listener, can feel tempo by tapping your foot to rhythmic pulse. Standard way to measure tempo is in beats per minute (BPM or bpm), meaning total number of beats played in 1 minute's time. This is almost always a whole number like 60, 120, or 78. At a tempo of 60 bpm, your foot taps 60 times each minute (or 1 beat per sec). At 120 bpm, get 2 beats every sec; &, at 90 bpm, get 1.5 beats every sec. Later in this chap when start working with TunePad, can set tempo by clicking on bpm indicator in top bar of a project, see Fig. 2.1: TunePad project information bar. Can click on tempo, time signature, or key to change settings for your project.

    Different genres of music have their own typical tempo ranges (although every song & every artist is different). E.g., hip-hop usually falls in 60–110 bpm range, while rock is faster in 100–140 bpm range. House/techno/trance is faster still, with tempos between 120–140 bpm. [Table: Genre: Tempo Range (BPM)].

    It takes practice for musicians to perform at a steady tempo, & they sometimes use a device called a *metronome* to help keep their playing constant with pulse of music. Can create a simple metronome in TunePad using 4 lines of code in a drum cell. This works best if switch instrument to `Drums` → `Percussion Sounds`.

    ```
    playNote(3, velocity = 100) # louder 1st note
    playNote(3, velocity = 60)
    playNote(3, velocity = 60)
    playNote(3, velocity = 60)
    ```

    Can adjust tempo of your metronome with bpm indicator Fig. 2.1: TunePad project information bar. Can click on tempo, time signature, or key to change settings for your project. As this example illustrates, computers excel at keeping a perfectly steady tempo. This is great if want precision, but there's also a risk that resulting music will sound too rigid & machine-like. When real people play music they often speed up or slow down, either for dramatic effect or just as a result of being a human. Depending on genre, performers might add slight variations in rhythm called swing or shuffle, that's a kind of back & forth rocking of beat that you can feel almost more than you can hear. Show how to add a more human feel to computer generated music later in book.
  - 2.2. Rhythmic notations. Over centuries, musicians & composers have developed many different written systems to record & share music. With invention of digital production software, a number of other interactive representations for mixing & editing have become common as well. Here are 4 common visual representations of same rhythmic pattern. Pattern has a total duration of 4 beats & can be counted as "1 & 2, 3 & 4". 1st 2 notes are $\frac{1}{2}$ beats long followed by a note that is 1 beat long. Then pattern repeats.
    - 2.2.1. Representation 1: Standard Western music notation. 1st representation shows standard music notation (or Western notation), a system of recording notes that has been developed over many hundreds of years. 2 thick vertical lines on left side of illustration indicate: this is rhythmic notation, i.e., there is no information about musical pitch, only rhythmic timing. Dots on long horizontal lines are notes whose shapes indicate duration of each to be played. Sometimes different percussion instruments will have their notes drawn on different lines. Describe what various note symbols mean in more detail in Fig. 2.2: Standard notation example.
    - 2.2.2. Representation 2: Audio waveforms. 2nd representation shows a visualization of actual audio waveform that gets sent to speakers when play music. Waveform shows amplitude (or volume) of audio signal over time. Next chap talks more

about audio waveforms, but for now can think of a waveform as a graph that shows literal intensity of vibration of your speakers over time. When compose a beat in TunePad, can switch to waveform view by clicking on small dropdown arrow at top-left side of timeline Fig. 2.3: Waveform representation of Fig. 2.2.

* 2.2.3. Representation 3: Piano (MIDI) roll. 3rd representation shows a piano roll (or MIDI (Musical Instrument Digital Interface) roll). This uses solid lines to show individual notes. Length of lines represents length of individual notes, & vertical position of lines represents percussion sound being played (kick drums & snare drums in this case). This representation is increasingly common in music production software. Many tools even allow for drag & drop interaction with individual notes to compose & edit music Fig. 2.4: Piano or MIDI roll representation of Fig. 2.2.
* 2.2.4. Representation 4: Python code. A final representation for now shows Python code in TunePad. In this representation, duration of each note is set using `beats` parameter of `playNote` function calls.

```
playNote(2, beats = 0.5)
playNote(2, beats = 0.5)
playNote(6, beats = 1)

playNote(2, beats = 0.5)
playNote(2, beats = 0.5)
playNote(6, beats = 1)
```

Each of these representation has advantages & disadvantages; they are good for conveying some kinds of information & less good at conveying others. E.g., standard rhythm notation has been refined over centuries & is accessible to an enormous, worldwide community of musicians. On other hand, it can be confusing for people who haven't learn how to read sheet music. Timing of individual notes is communicated using tails & flags attached to notes, but there's no consistent mapping between horizontal space & timing.

Audio waveform is good at showing what sound *actually* looks like – how long each note rings out ("release") & how sharp its onset is ("attack"). Helpful for music production, mixing, & mastering. On other hand, waveforms don't really tell you much about pitch of a note or its intended timing as recorded by composer.

Python code is easier for computers to read than humans – it's definitely not sth you would hand to a musician to sight read. On other hand, it has advantage that it can be incorporated into computer *algorithms* & manipulated & transformed in endless ways.

There are many, many other notation systems designed to transcribe a musical performance – what hear at a live performance – onto a sheet of paper or a computer screen. Each of these representations was invented for a specific purpose &/or genre of music. Might pick a representation based on context & whether you're in role of a musician (& what kind of instrument you play), a singer, a composer, a sound engineer, or a producer. Music notation systems are as rich & varied as cultures & musical traditions that invented them. 1 nice thing about working with software: easy to switch between multiple representations of music depending on task trying to accomplish.

◦ 2.3. Standard rhythmic notation. This sect will review a standard musical notation system that has roots in European musical traditions. This system is versatile & has been refined & adapted over a long period of time across many countries & continents to work with an increasingly diverse range of instruments & musical genres. Start with percussive rhythmic note values in this chap, & move on to working with pitched instruments in Chap. 3.

Fig. 2.5: Common note symbols starting with a whole note (4 beats) on top down to 16th notes on bottom. Notes on each new row are half length of row above. shows most common symbols used in rhythmic music notation. Notes are represented with oval-shaped dots that are either open or closed. All notes except for whole note have tails attached to them that can point either up or down. It doesn't matter which direction (up or down) tail points. Notes that are faster than a quarter note also have horizontal flags or beams connected to tails. Each additional flag or beam indicates note is twice as fast.

Symbol: Name: Beats: TunePad code:

1. Whole Note: Larger open circle with no tail & no flag: 4: `playNote(1, beats = 4)`
2. Half Note: Open circle with a tail & no flag: 2: `playNote(1, beats = 2)`
3. Quarter Note: Solid circle with a tail & no flag: 1: `playNote(1, beats = 1)`
4. 8th Note: Solid circle with a tail & 1 flag or bar: 0.5 or $\frac{1}{2}$: `playNote(1, beats = 0.5)`
5. 16th Note: Solid circle with a tail & 2 flags or bars: 0.25 or $\frac{1}{4}$: `playNote(1, beats = 0.25)`
6. Dotted Half Note: Open circle with a tail. Dot adds an extra beat to half note: 3: `playNote(1, beats = 3)`
7. Dotted Quarter Note: Solid circle with a tail. Dot adds an extra half-beat: 1.5: `playNote(1, beats = 1.5)`
8. Dotted 8th Note: Solid circle with tail & 1 flag. Dot adds an extra quarter beat: 0.75. `playNote(1, beats = 0.75)`

Standard notation also includes *dotted notes*, where a small dot follows note symbol. With a dotted note, take original note's duration & add half of its value to it. So, a dotted quarter note is 1.5 beats long, a dotted half note is 3 beats long, etc.

There are also symbols representing different durations of silence or "rests".

Symbol: Name: Beats: TunePad code

1. Whole Rest: 4: `rest(beats = 4)`
2. Half Rest: 2: `rest(beats = 2)`

3. Quarter Rest: 1: `rest(beats = 1)`
4. 8th Rest: 0.5 or $\frac{1}{2}$: `rest(beats = 0.5)`
5. 16th Rest: 0.25 or $\frac{1}{4}$: `rest(beats = 0.25)`

- 2.4. Time signatures. In standard notation, notes are grouped into segments called *measures* (or bars). Each measure contains a fixed number of beats, & duration of all notes in a measure should add up to that amount. Relationship between measures & beats is represented by a fraction called a *time signature*. Numerator (or top number) indicates number of beats in measure, & denominator (bottom number) indicates beat duration.

  – Trong ký hiệu chuẩn, các nốt nhạc được nhóm thành các đoạn gọi là *nhịp* (hoặc ô nhịp). Mỗi ô nhịp chứa 1 số phách cố định, & thời lượng của tất cả các nốt nhạc trong 1 ô nhịp phải cộng lại bằng số lượng đó. Mối quan hệ giữa các ô nhịp & nhịp được biểu diễn bằng 1 phân số gọi là *nhịp điệu*. Tử số (hoặc số trên cùng) biểu thị số phách trong ô nhịp, & mẫu số (số dưới cùng) biểu thị thời lượng của phách.

  1. $\frac{4}{4}$: 4-4 Time or "Common tTime": There are 4 beats in each measure, & each beat is a quarter note. This time signature is sometimes indicated using a special symbol
  2. $\frac{2}{2}$: 2-2 Time or "Cut Time": There are 2 beats in each measure, & beat value is a half note. Cut time is sometimes indicated with a 'C' with a line through it.
  3. $\frac{2}{4}$: 2-4 Time: There are 2 beats in each measure, & quarter note gets beat.
  4. $\frac{3}{4}$: 3-4 Time: There are 3 beats in each measure, & quarter note gets beat.
  5. $\frac{3}{8}$: 3-8 Time: There are 3 beats in each measure, & 8th note gets beat.

  Most common time signature is $\frac{4}{4}$. So common, in fact, referred to as *common time*. Often denoted by a C symbol shown in table. In common time, there are 4 beats to each measure, & quarter note "gets beat" meaning: 1 beat is same as 1 quarter note.

  Vertical lines separate measures in standard notation. In example, there are 2 measures in 4/4 time (4 beats in each measure, & each beat is a quarter note).

  If have a time signature of 3/4, then there are 3 beats per measure, & each beat's duration is a quarter note. Some examples of songs is 3/4 time are *My Favorite Things* from *The Sound of Music, My 1st Song* by Jay Z, *Manic Depression* by Jimi Hendrix, & *Kiss from a Rose* by Seal.

  If those notes were 8th notes, it would look like Fig.

  Other common time signatures include 2/4 time (with 2 quarter note beats per measure) & 2/2 time (with 2 *half note* beats in each measure). With 2/2 there are actually 4 quarter notes in each measure because 1 half note has same duration as 2 quarter notes. For this reason, 2/2 time is performed similarly to common time, but is generally faster. It is referred to as *cut time* & is denoted by a C symbol with a line through it.

  Can adjust time signature of your TunePad project by clicking on time indicator in top bar (see Fig. 2.1).

- 2.5. Percussion sounds & instruments. Working with rhythm, come across lots of terminology for different percussion instruments & sounds. A quick rundown on some of most common drum sounds that you'll work with in digital music (Fig. 2.6: Drums in a typical drum kit.)

  Drum names: Description: TunePad note number

  1. Kick or bass drum: Kick drum (or bass drum) makes a loud, low thumping sound. Kicks are commonly placed on beats 1 & 3 in rock, pop, house, & electronic dance music. In other genres like hip-hop & funk, kick drums are very prominent, but their placement is more varied: 0 & 1
  2. Snare: Snare drums make a recognizable sharp staccato sound that cuts across frequency spectrum. They are built with special wires called snares that give drums its unique snapping sound. Snare drums are commonly used on beats 2 & 4: 2 & 3
  3. Hi-hat: Hi-hat is a combination of 2 cymbals sandwiched together on a metal rod. A foot pedal opens or closes cymbals together. In closed position hi-hat makes a bright tapping sound. In open position cymbal is allowed to ring out. Hi-hats have become an integral part of rhythm across almost all genres of popular music.: 4 (closed), 5 (open)
  4. Low, mid, high tom: Tom drums (tom-toms) are cylindrical drums that have a less snappy sound than snare drum. Drum kits typically have multiple tom drums with slightly different pitches (e.g. low, mid, & high).: 6, 7, 8
  5. Crash cymbal: A large cymbal that makes a loud crash sound, often used as a percussion accent: 9
  6. Claps & shakers: Different TunePad drum kits include a range of other percussion sounds common in popular music including various claps, shakers, & other sounds.: 10 & 11

  * 2.5.1. 808 Drum kit. Released in early 1980s, Roland 808 drum machine was a hugely influential sound in early hip-hop music (& other genres as well). 8.8 used electronic synthesis techniques to create synthesis replicas of drum sounds like kicks, snares, hats, toms, cowbells, & rim shots. Tinkerers would also open up 808s & hack circuits to create entirely new sounds. Today 808s usually refers to low, booming bass lines that were 1st generated using tweaked versions of 808s' kick drums. TunePad's default drum kit uses samples that sound like original electronically synthesized 808s (Fig. 2.7: Roland 808 drum sequencer.).
  * 2.5.2. Selecting TunePad instruments. When coding in Tunepad, sound that your code makes will depend on instrument you have selected. If coding a rhythm, can choose from several different drum kits including an 808 & rock kits. Can change instrument by clicking on selector shown below Fig. 2.8: Changing an instrument's voice in TunePad.

- 2.6. Coding rhythm in Python.
  - 2.6.1. Syntax errors. Python is a text-based language, i.e., you're going to be typing code that has to follow strict grammatical rules. When speak a natural language like English, grammar is important, but can usually bend or break rules & still get your message across. When say something ambiguous it can be ironic, humorous, or poetic. This isn't case in Python. Python has no sense of humor & no appreciation for poetry. If make a grammatical mistake in coding, Python gives a message called a *syntax error*. These messages can be confusing, but they're there to help you fix your code in same way that a spell checker helps you fix typos. Here's what a syntax error looks like in TunePad (Fig. 2.9: Example of a Python syntax error in TunePad. This line of code was missing a parenthesis symbol.)
    This line of code was missing a parenthesis symbol, which generated error message "bad input on line 5". Notice Python is giving hints about where problems are & how to fix them, but those hints aren't always that helpful & can be frustrating for beginners.
  - 2.6.2. Flow of control. A Python program is made up of a list of statements. For most part, each statement goes on its own line in your program. Python will read & perform each line of code from top to bottom in order that you write them. In programming this is called *flow of control*. This is similar to way you would read words in a book or notes on a line of sheet music. Difference: programming languages also have special rules that let you change flow of control. Those rules include *loops* (which repeat some part of your code multiple times), *conditional logic* (which runs some part of your code only if some condition is met), & *user-defined functions* (which lets you create your own functions that can be called). Talk about these special "control structures" later in book.
- 2.7. Calling functions. Almost everything you do in Python involves *calling* functions. A function (sometimes called a command or an instruction) tells Python to do something or to compute a value. E.g., `playNote` function tells TunePad to make a sound. There are 3 things you have to do to call a function:

  1st, have to write name of function. Functions have 1-word names (with no spaces) that can consist of letters, numbers, & underscore _ character. Multi-word functions will either use underscore character between words as in `my_multi_word_function()` or each new word will be capitalized as in `playNote()`.

  2nd, after type name of function, have to include parentheses. Parentheses tell Python that you're calling a function.

  3rd, include any *parameters* that you want to *pass* to function in between left & right parentheses. A parameter provides extra information or tells function how to behave. E.g., `playNote` statement needs at least 1 parameter to tell it which note or sound to play. Sometimes functions accept multiple parameters (some of which can be optional). `playNote` function accepts several optional parameters described in next sect. Each additional parameter is separated with a comma (Fig. 2.10: Calling `playNote` function in TunePad with 2 parameters inside parentheses.)
- 2.8. playNote functions. `playNote` function tells TunePad to play a percussion sound or a musical note. `playNote` function accepts up to 4 parameters contained within parentheses.

```
playNote(1, beats = 1, velocity = 100, sustain = 0)
```

Name: Description

- `note`: This is a *required* parameter that says which note or percussion sound to play. Kind of sound depends on which instrument you have selected in TunePad for this code. Can play more than 1 note at same time by enclosing notes in square brackets.
- `beats`: An *optional* parameter that says how long to play note. TunePad *playhead* will be moved forward by duration given. This parameter can be a whole number (like 1 or 2), a decimal number (like 1.5 or 0.25), or a fraction (like 1/2).
- `velocity`: An *optional* parameter that says how loud to play note or sound. A value of 100 is full volume, & a value of 0 is no volume (muted). Velocity is a technical term in digital music that means how fast or how hard you hit instrument. You might imagine it as how loud a drum sounds based on how hard it gets hit.
- `sustain`: An *optional* parameter that allows a note to ring out for an additional number of beats without advancing playhead.
- 2.8.1. Optional parameters. Sometimes parameters are *optional*, i.e., they have a value that gets provided by default if you don't specify one. For `playNote`, only note parameter is required. If don't pass other parameters, it provides values for you by default. Can also include *names* of parameters in a function call. E.g., all 4 of lines below do exactly same thing; they play a note for 1 beat. 1st 2 use parameters without their names. 2nd 2 include names of parameter, followed by equals sign =, followed by parameter value.

```
playNote(60) # the beats parameter is optional
playNote(60, 1) # with the beats parameter set to 1
playNote(60, beats = 1) # with a parameter name for beats
playNote(note = 60, beats = 1) # with a parameter name for note and beats
```

- 2.8.2. Comments. In code above, some of text appears after hashtag # symbols on each line. This text is called a *comment*. A comment is a freedom note that programmers add to make their code easier to understand. Comment text is ignored by Python, so you can write anything you want after hashtag symbol on a line. Can also use a hashtag at beginning of a line to temporarily disable code. This is called "commenting out" code.

- 2.9. `rest` function. Silence is an important element of music. `rest` function generates silence, or a break in sound. It only takes 1 parameter, which is length of time the rest is held. So `rest(beats = 2)` will trigger a rest for a length of 2 beats. If don't provide a parameter, `rest` uses a value of 1.0 by default.

```
rest() # rest for one beat
rest(1.0) # rest for one beat
rest(0.25) # rest for one quarter beat
rest(beats = 0.25) # rest for one quarter beat
```

- 2.10. Examples of `playNote, rest`. Try a few examples of `playNote, rest` to get warmed up. This rhythm plays 2 8 notes (beats = 0.5) followed by a quarter note (beats = 1). Pattern then repeats a 2nd time. Here's how would code this in TunePad with a kick drum & snare:

```
playNote(1, beats = 0.5) # play a kick drum (1) for half a beat
playNote(1, beats = 0.5)
playNote(2, beats = 1) # play snare (2) for one beat
playNote(1, beats = 0.5) # play kick (1) for half a beat
playNote(1, beats = 0.5)
playNote(2, beats = 1) # play snare (2) for one beat
```

Here's another example that plays a quarter note followed by a rest of 0.5 beats followed by an 8 note (beats = 0.5). Pattern is repeated 2 times in a row:

```
playNote(2, beats = 1) # play a snare drum (2) for one beat
rest(beats = 0.5) # rest for half a beat
playNote(1, beats = 0.5) # play a kick drum (1) for half a beat
playNote(2, beats = 1) # play a snare drum (2) for one beat
rest(beats = 0.5) # rest for half a beat
playNote(1, beats = 0.5)
```

A 3rd example that plays 8 notes in a row, each an 8 note (beats = 0.5).

- 2.11. Loops. All of examples in prev sect included repeated elements. &, if listen closely, can hear repeated elements at all levels of music. There are repeated rhythmic patterns, recurring melodic motifs, & storylines defined by song sects that get repeated & elaborated. It turns out: there are many circumstances in both music & computer programming where we want to repeat something over & over again.

  To show how can take advantage of some of capabilities of Python, start with last example from prev sec where we wanted to tap out a run of 8th notes (0.5 beats) on hi-hat. 1 way to program that rhythm would be to just type in 8 `playNotes` in a row.

```
for i in range(8):
    playNote(4, beats = 0.5)
    print(i)
```

This will get job done, but there are a few problems with this style of code. 1 problem: it violates 1 of most important character traits of a computer programmer – laziness! A lazy programmer is someone who works smart, not hard. A lazy programmer avoids doing repetitive, error-prone work. A lazy programmer knows that there are some things that computer can do better than a human can.

In Python (& just about any other programming language), when want to do something multiple times, can use a loop. Python has a number of different kinds of loops, but, in this case, our best option is sth called a `for` loop. Version of code on right repeats 8 times in a row. For each iteration of loop, TunePad `playNote` function gets called.

With original code on left, had to do a lot of tying (or, more likely, copying & pasting) to enter our program – a warning sign that we're not being lazy enough. We generated a lot of repetitive code, which makes program harder to read (not as legible), error prone, & not as elegant as it could be. Right-side code accomplishes same thing with just 3 lines of code instead of 8.

Finally, code on left is harder to change & reuse. What if wanted to use a different drum sound (like a snare instead of a hat)? Or, what if wanted to tap out a run of 16 16th notes instead of 8 8th notes? Would have to go through code line by line making same change over & over again. This is a slow, error-prone process that is definitely not lazy or elegant.

To see why this is better, try changing code on right so that it plays 16 16th notes instead of 8 8th notes. Or try changing drum sound from a hat to sth else. `print` statement on line 3 is just there to help you see what's going on with your code. If click on `Show Python Output` option, can see how variable called `i` (that gets created on line 1) counts up from 0 to 7 Fig. 2.11: How to show print output of your code in a TunePad cell. More detail about anatomy of a `for` loop (Fig. 2.12: Anatomy of a `for` loop in Python.) A `for` loop with range function:

* begins with `for` keyword

* includes a loop *variable* name; this can be anything you want (above it is `i`). Each time loop goes around, loop variable is incremented by 1.
* includes `in` keyword
* includes `range` function that says how many times to repeat
* includes a colon :
* includes a block of code indented by 4 spaces

Python uses *indentation* to determine what's *inside* loop, meaning it's sect of code that gets repeated multiple times. Intended block of code is repeated total number of times specified by `range`. Try adding a few extras to prev example. In version below, add a run of 16th notes for last beat.

```
for i in range(6):
    playNote(4, beats = 0.5)
for i in range(4):
    playNote(4, beats = 0.25)
```

But there are lots of other things we could do as well. If wanted to play an even faster run, could use code like:

```
for i in range(8):
    playNote(4, beats = 0.125)
```

Or, if wanted to play a triplet that divides a half-beat into 3 equal parts, could do sth like this:

```
for i in range(3):
playNote(4, beats = 0.25 / 3) # divide into 3 parts
```

If open this example in TunePad, can experiment with different combinations of numbers to get different effects: https://tunepad.com/examples/loops-and-hats

- 2.12. Variables. A *variable* is a name you give to some piece of information in a Python program. Can think of a variable as a kind of nickname or alias. Similar to loops, variables help make your code more elegant, easier to read, & easier to change in future. E.g., code on left plays a drum pattern without variables, & code on right plays same thing with variables. Notice how variables help make code easier to understand because they give us descriptive names for various drum sounds instead of just numbers.

```
playNote(0)
playNote(4)
playNote(2)
playNote(4)
```

```
kick = 0
hat = 4
snare = 2

playNote(kick)
playNote(hat)
playNote(snare)
playNote(hat)
```

In version on right defined a variable called `kick` on line 1, a variable called `hat` on line 2, & a variable called `snare` on line 3. Each variable is *initialized* to a different number for corresponding drum sound. Also possible to change value of a variable later in program by assigning it a different number.

```
kick = 0
playNote(kick) # plays sound 0
kick = 1       # set kick to a different value
playNote(kick) # plays sound 1
```

Variable names can be anything you want as long as they're 1 word long (no spaces) & consist only of letters, numbers, & underscore character `_`. Variable names cannot start with a number, & they can't be same as any existing Python keyword.

As begin to get comfortable with code & to exercise your creativity, find yourself wanting to experiment with sounds. Might want to try different sounds for same rhythmic pattern, maybe change a high-hat sound to a shaker to get a more organic feel. Using variables makes it easy to experiment by changing values around.

Another example with a hi-hat pattern. Imagine really like this pattern, but wondering how it would sound with a different percussion instrument. Maybe you want to change 4 sound to a shaker sound (like 11). Nice thing about variables: can give

them just about any name you want as long as Python is not already using that name for something else. This way you can make name meaningful to you. So, for our shaker example could create a variable with a meaningful name like `shake` & set it equal to 11. When use variable `shake` you are inserting whatever number is currently assigned to it.

```
for i in range(8):
    playNote(4,
    beats = 0.5)
for i in range(8):
    playNote(4,
    beats = 0.25)
for i in range(4):
    playNote(4,
    beats = 0.5)

shake = 11
for i in range(8):
    playNote(shake,
    beats = 0.5)
for i in range(8):
    playNote(shake,
    beats = 0.25)
for i in range(4):
    playNote(shake,
    beats = 0.5)
```

As progress with coding, find that loops & variables help create a smoother workflow that gives you more flexibility, freedom, & creative power. Try out using variables with exercise https://tunepad.com/examples/variables.

- 2.13. More on syntax errors. Python code is like a language with strict grammatical rules called syntax. When make a mistake in coding – & everyone makes coding mistakes all time – Python will give feedback about what error is & approximately what line it's on. E.g., if been trying to code exercises in this chap, may have seen a message like Fig. 2.13: Example of a Python syntax error. Command 'ployNote' should instead say 'playNote'.

  This is telling us: there is an error on line 6 that can be fixed by changing text, "ployNote". When using a variable or function in your code, Python is expecting you to type it *exactly* the same as it was defined. A simple typo can stop your program from running, but it's also easily fixed. Here just need to update line to say `playNote` instead of `ployNote`.

  Other syntax errors are trickier. Message in Fig. 2.14: Example of a Python syntax error. Here problem is actually on line 1, not line 2. Message in Fig. 2.14 is confusing because problem is actually on line 1 even though syntax error says line 2. Problem is a missing right parenthesis on line 1.

  1 technique coders use to find source of errors like this: comment out lines of code before & after an error. E.g., to comment out1st line of code above, could change it to look like this:

```
# playNote(60
rest(1)
```

  Adding hashtag at beginning of 1st line means Python ignores it, in this case fixing syntax error & giving us another clue about source of problem.

  Another surprisingly helpful trick: just paste your error message verbatim into your favorite search engine. There are huge communities of Python coders out there who have figured out how to solve almost every problem with code imaginable. You can often find a quick fix to your problem just by browsing through a few of top search results.

  If want practice fixing syntax errors in your code, can try 1 of mystery-melody challenges on TunePad: https://tunepad.com/examples/mystery-melody.

- 2.14. `playhead`. Timing of notes in TunePad is determined by position of an object called `playhead`. In early days of music production, recordings were made using analog tape. Sound wave signals coming from a microphone or some other source were physically stored on magnetic tape using a mechanism called a *record head*. As tape moved by, record head would inscribe patterns of magnetic material inside of tape, thus creating a recording of music. To play recording back, a `playhead` would pick up fluctuations in tape's magnetic material & convert it back into sound waves for listeners to hear. Fast forward to digital realm. No longer have playheads or record heads, but maintain that metaphor when referring to notion of sound moving in time. Concept of a playhead is common across audio production software as point in time where audio is playing.

  – Thời gian của các nốt nhạc trong TunePad được xác định bởi vị trí của 1 đối tượng được gọi là `playhead`. Vào những ngày đầu của quá trình sản xuất âm nhạc, các bản ghi âm được thực hiện bằng băng analog. Tín hiệu sóng âm phát ra từ micrô hoặc 1 số nguồn khác được lưu trữ vật lý trên băng từ bằng 1 cơ chế được gọi là *record head*. Khi băng di chuyển qua, đầu ghi sẽ khắc các mẫu vật liệu từ tính bên trong băng, do đó tạo ra bản ghi âm nhạc. Để phát lại bản ghi âm, `playhead` sẽ thu các dao động trong vật liệu từ tính của băng & chuyển đổi nó trở lại thành sóng âm để người nghe có thể nghe. Chuyển

nhanh đến thế giới kỹ thuật số. Không còn đầu phát hoặc đầu ghi nữa, nhưng vẫn duy trì phép ẩn dụ đó khi đề cập đến khái niệm âm thanh chuyển động theo thời gian. Khái niệm về đầu phát phổ biến trong các phần mềm sản xuất âm thanh như 1 điểm thời gian mà âm thanh đang phát.

In TunePad, when place a note with `playNote` function, it advances playhead forward in time by duration of note specified by `beats` parameter. There are several functions available to get information about position of playhead & move it forward or backward in time.

Function: Description

* `getPlayhead()`: Returns current position of playhead in beats. Note: `getPlayhead` returns elapsed number of beats, i.e. if playhead is at beginning of track function will return 0. If 1.5 beats have elapsed, `getPlayhead` will return 1.5. If 40 beats have elapsed, it will return 40, & so on.
* `getMeasure()`: Returns current measure as an integer value. Note: `getMeasure` returns an elapsed number of measures. So, if playhead is at beginning of track or anywhere before end of 1st measure, function will return 0. If playhead $\geq$ start of 2nd measure, `getMeasure` will return 1, & so on.
* `getBeat()`: Returns an elapsed number of beats *within current measure* as a decimal number. E.g., if playhead has advanced by a quarter beat within a measure, `getBeat` will return 0.25. Value returned by `getBeat` will always $<$ total number of beats in a measure.
* `fastForward(beats)`: Advance playhead forward by given number of beats relative to current position. Note: this is identical to `rest` function. Negative beat values move playhead backward.
* `rewind(beats)`: Move playhead back in time by given number of beats. This can be a useful way to play multiple notes at same time. Beats parameter specifies number of *beats* to move playhead. Negative values of beats move playhead forward.
* `rest(beats)`: Advance playhead forward by given number of beats without playing a sound. This is identical to `fastForward` function.
* `moveTo(beats)`: Move playhead to an arbitrary position. `beats` parameter specifies point that playhead will be placed as an elapsed number of beats. E.g., `moveTo(0)` will move playhead to beginning of a track (zero elapsed beats). `moveTo(1)` will place playhead at end of 1st beat & right before start of 2nd beat.

Can control where playhead is relative to music we make by using `moveTo, fastForward, rewind` commands. `rewind` & `fastForward` functions move playhead backward or forward relative to current point in time. `moveTo` function takes playhead & moves it to an arbitrary point in time. In TunePad, playhead represents an *elapsed* number of beats. So, to move to beginning of a track, would use `moveTo(0)`, i.e. 0 elapsed beats. To move to beginning of 2nd beat, would use `moveTo(1)`, i.e. 1 elapsed beat. These commands are useful for adding multiple overlapping rhythms to a single TunePad cell. See more on how these commands can be used in Chap. 8.

- 2.15. Basic drum patterns. Code some foundational drum patterns. There is also a link to code in TunePad so that you can play around with beat & make it your own.
  * 2.15.1. 4-on-the-floor. 4-on-the-floor pattern is a staple of House, EDM, disco, & pop music. It has a driving dance beat defined by 4 kick drum hits on each beat (thus 4 beats on floor). This beat is simple but versatile. Can spice it up by moving hi-hats around & adding kicks, snares, & other drums in unexpected places. Can make basic pattern with just 3 instruments: kick, snare, & hats.
    Kick drums are lowest drum sound in a drum kit. Start by laying down kick drums on each beat of measure. These low sounds give this pattern a driving rhythmic structure that sounds great at higher tempos. Then add snare hits on even beats (2 & 4). Snare adds energy & texture to beat. Finally, add hi-hats. These are highest pitch instruments in most drum beats, & they help outline groove to emphasize beat. Can find this example at https://tunepad.com/examples/four-on-the-foor.

```
# define instrument variables
kick = 0
snare = 2
hat = 4

# lay down four kicks (on the floor)
playNote(kick)
playNote(kick)
playNote(kick)
playNote(kick)

moveTo(0) # reset playhead to the beginning

# add snares on the even beats
rest(1)
playNote(snare)
rest(1)
playNote(snare)
```

```
moveTo(0) # reset playhead to the beginning

# hi-hat pattern with a loop!
for i in range(8):
    playNote(hat, 0.5)
```

∗ 2.15.2. Blues. Blues is a genre of music that evolved from African American experience, starting as field songs, evolving into spirituals, & eventually became Blues. This beat is in 3/4 time, i.e. there are 3 beats in each measure. Can find this example at https://tunepad.com/examples/blues-beat.

```
# define instrument variables
kick = 0
snare = 2
hat = 4

# lay down kick and snare pattern
playNote(kick, beats = 0.5)
playNote(kick, beats = 0.25)
playNote(snare, beats = 0.25)
rest(.25)
playNote(kick, beats = 0.25)
playNote(kick, beats = 0.25)
rest(.25)
playNote(kick, beats = 0.25)
playNote(snare, beats = 0.25)
playNote(kick, beats = 0.25)
playNote(kick, beats = 0.25)

moveTo(0) # reset playhead to beginning

# add hi-hats
playNote (4, beats = 0.25)
for i in range(3):
    rest(0.25)
    playNote (hat, beats = 0.25)
    playNote (hat, beats = 0.25)
rest(0.25)
playNote(hat, beats = 0.25)
```

∗ 2.15.3. Latin. Latin beats are known for their syncopated rhythms that emphasize so-called "weak beats" in a measure. This drum pattern is 2 measures long. Our kick pattern sounds much like a heartbeat & solidly grounds our entire beat. Our snare is playing a *clave* pattern, which is common in many forms of Afro-Cuban music e.g. salsa, mambo, reggae, reggaeton, & dancehall. In 2nd measure, have hits on 1st beat & 2nd half of 2nd beat (counts 1 & 2.5). Finally, add a hi-hat on every 8th note. Can find this example at https://tunepad.com/examples/latin-beat.

```
# define instrument variables
kick = 0
snare = 3
hat = 4

# lay down kicks for the heartbeat
for i in range(2):
    playNote(kick, beats = 1.5)
    playNote(kick, beats = 0.5)
    playNote(kick, beats = 1.5)
    playNote(kick, beats = 0.5)

moveTo(0) # reset playhead

# add snare
rest(1.0)
playNote(snare)
playNote(snare)
rest(1.0)
playNote(snare, beats = 1.5)
```

```
playNote(snare, beats = 1.5)
playNote(snare)

moveTo(0) # reset playhead

# lay down hi-hats
for i in range(16):
    playNote(hat, beats = 0.5)
```

* 2.15.4. Reggae. A common reggae beat is *1-drop beat*, which gets its name due to fact there's no hit on 1st beat. Rather, accent is on 3rd beat which contributes to strong backbeat & laid back feel in reggae. Using swung 8th notes for our hi-hats, & adding an open hi-hat hit on very last note to add texture. 1st hit is held for $\frac{2}{3}$ of beat & 2nd for $\frac{1}{3}$. Both our kick & snare hit on 3rd beat. Can find this example at https://tunepad.com/examples/reggae-beat.

```
# define instrument variables
kick = 0
snare = 2
hat = 4
open_hat = 5

# lay down kick and snare together
rest(2)
playNote([kick, snare])

moveTo(0) # reset playhead

# lay down swung hi-hat pattern
for i in range(3):
    playNote(hat, 2.0 / 3) # two-thirds
    playNote(hat, 1.0 / 3) # one-third

playNote(hat, 2.0 / 3)
playNote(open_hat, 1.0 / 3)
```

* 2.15.5. Other common patterns. Here are a few other drum patterns in different genres that you can try coding for yourself. Hip-hop (late-1990s) 90 bpm, Hip-hop (mid-2000s) 85 bpm, basic pop/rock 130 bpm, Trap (mid-2010s) 130 bpm (double dots mean stuttered hi-hats), pop/hip-hop 70 bpm, West coast beat (late 2010s) 100 bpm, Dance/EDM/Hip-hop (circa 1982) 130 bpm, Hip-hop (mid-1990s) 85 bpm.

◦ 2.16. Drum sequencers. A drum sequencer is a tool for creating drum patterns. Early sequencers like Roland 808 were physical pieces of hardward. Now most people use software-based sequencers, although basic principles are the same: Sequencers look like a grid with rows for different drum sounds & columns for short time slices (usually 16th notes or 32nd notes).

TunePad includes a drum sequencer (Fig. 2.15: TunePad composer interface provides drum & bass sequencers.) that can be helpful for playing around with different rhythmic ideas tunepad.com/composer. Can add drum sounds at different time slices by clicking on gray squares of grid, & once have a pattern you like you can convert it into Python code. When converting a drum sequencer pattern to code, it can be helpful to code column by column instead row by row. What that means: we work left to right across drum pattern. For each column, look at all sounds that hit at that time slice. Can then cue up each of those sounds using a simple `playNote` statement. A quick example pattern.

Look at 1st column & see there's a single kick drum.

```
playNote(0, beats = 0.25)
```

Look at 2nd column & see that it's empty, so rest:

```
rest(0.25)
```

3rd column includes both a hat (note 4) & a kick (note 0). To play these together, can use a single `playNote` command with both sounds enclosed in square brackets like this:

```
playNote([ 0, 4 ], beats = 0.25)
```

A special Python structure: list – a convenient way to play more than 1 sound at same time. If keep going with this column-by-column strategy, complete code:

```
playNote(0, beats = 0.25)
```

```
rest(0.25)
playNote([ 0, 4 ], beats = 0.25)
rest(0.25) # kick + hat
playNote(2, beats = 0.25)
rest(0.25)
playNote(4, beats = 0.25)
playNote(10, beats = 0.25)
rest(0.25)
playNote(10, beats = 0.25)
playNote([ 0, 4 ], beats = 0.25) # kick + hat
playNote(0, beats = 0.25)
playNote(2, beats = 0.25)
rest(0.25)
playNote(4, beats = 0.25)
rest(0.25)
```

Coding column by column can be a little quicker & produces more compact code.

Note 1. *Term beat can also refer to main groove in a dance track ("drop beat") or instrumental music that accompanies vocals in a hip-hop track ("she produced a beat for a new artist") in addition to other meanings.*

- Interlude 2: Custom Trap Beat. In this interlude, run with skills you picked up in preceding chap to create a custom Trap beat. This beat will use kick drum, snare, & hi-hats. Can follow along online by visiting https://tunepad.com/interlude/trap-beat.

1. **Step 1: Defining variables.** Start by logging into TunePad & creating a new project called "Custom Trap Beat". Add a new *Drums* instrument to your project. In this call, declare *variables* for your drum sounds.

```
# variables for drums
kick = 1
snare = 2
hat = 4
```

2. **Step 2: Basic drum pattern.** Code for a basic drum pattern. Add this code to your Drum call after variables:

```
# kick and snare
playNote(kick, beats = 0.75)
playNote(kick, beats = 0.25)
playNote(snare, beats = 1.5)
playNote(kick, beats = 0.75)
playNote(kick, beats = 0.75)
playNote(kick, beats = 2)
playNote(snare, beats = 1)
```

Break down each of these lines 1 by 1 [Table] When done, pattern should look sth like Fig. 2.16: Basic drum pattern.

3. **Step 3: Add hi-hat rolls & stutters.** Now add a new `Drum Cell` to your project for hi-hat rolls & stutters. To add our hi-hat runs, 1st review `for loops` in Python Fig. 2.17: Declaring a for loop for hi-hat runs in Python.
Indented block of code is run total number of times specified by *range* of loop. Try this example pattern in your project:

```
for i in range(4):
    playNote(hat, beats = 0.25)

for i in range(4):
    playNote(hat, beats = 0.25 / 2)

for i in range(8):
    playNote(hat, beats = 0.25)

for i in range(5):
    playNote(hat, beats = 0.25 / 5)

playNote(hat, beats = 0.25)
```

Your cell should now have a pattern like Fig. 2.18: Hi-hat stutter patterns.

4. **Step 4: Customize.** After trying example in Step 3, make up your own stutter pattern to go with your kick & snare drums. Can use any combination of beats, but make sure it adds up to a multiple of 4 beats so that your beat loops correctly! Here are a few for loops that play stutters at different speeds:

```
# couplet
for i in range(2):
    playNote(hat, beats = 0.25 / 2) # divide in half

# triplet
for i in range(3):
    playNote(hat, beats = 0.25 / 3) # divide into 3 parts

# quad
for i in range(4):
    playNote(hat, beats = 0.25 / 4) # divide into 4 parts

# fifthlet?
for i in range(5):
    playNote(hat, beats = 0.25 / 5) # divide into 5 parts
```

Try out different instrument sounds by changing values of variables & switching to a different drum kit. Can also experiment with changing tempo. For more inspiration, this TunePad project has several popular hip-hop beat patterns that you can experiment with https://tunepad.com/interlude/drum-examples.

- 3. Pitch, harmony, & dissonance. Chap. 2 introduced basics of rhythm & how to use Python programming language to code beats with percussion sounds. In this chap, explore topics of pitch, harmony, & dissonance – or what happens when you bring tonal instruments & human voice into music. Start with physical properties of sound (including frequency, amplitude, & wavelength) & why different musical notes sound harmonious or dissonant when played together. Also talk about different ways to represent pitch, including frequency value, musical note names, & MIDI (Musical Instrument Digital Interface) note numbers that we can use with Python code & TunePad.

  – Chương 2 giới thiệu những điều cơ bản về nhịp điệu & cách sử dụng ngôn ngữ lập trình Python để mã hóa nhịp điệu với âm thanh bộ gõ. Trong chương này, hãy khám phá các chủ đề về cao độ, sự hòa hợp, & sự bất hòa – hoặc điều gì xảy ra khi bạn đưa nhạc cụ có âm & giọng nói của con người vào âm nhạc. Bắt đầu với các đặc tính vật lý của âm thanh (bao gồm tần số, biên độ, & bước sóng) & lý do tại sao các nốt nhạc khác nhau nghe có vẻ hòa hợp hay bất hòa khi chơi cùng nhau. Ngoài ra, hãy nói về các cách khác nhau để biểu diễn cao độ, bao gồm giá trị tần số, tên nốt nhạc, & số nốt MIDI (Giao diện kỹ thuật số nhạc cụ) mà chúng ta có thể sử dụng với mã Python & TunePad.

  - 3.1. Sound Waves. All sound, no matter how simple or complex, is made up of waves of energy that travel through air, water, or some other physical medium. If could see a sound wave, it might look sth like ripples of water from a pebble dropped in a still pound. Pebble is like source of sound, & ripples are sound waves that expand outward in all directions. Any source of sound (car horns, cell phone rings, chirping birds, or a plucked guitar string) sends vibrating waves of air pressure out at around 343 meters per sec (speed of sound) from source. It's not that air molecules themselves travel from source of sound to our ears; it's that small localized movements in molecules create fluctuations in air pressure that propagate outward over long distances.

    – Mọi âm thanh, dù đơn giản hay phức tạp, đều được tạo thành từ các sóng năng lượng truyền qua không khí, nước hoặc 1 số môi trường vật lý khác. Nếu có thể nhìn thấy sóng âm, nó có thể trông giống như gợn sóng nước từ 1 viên sỏi thả vào 1 pound đứng yên. Viên sỏi giống như nguồn âm thanh, & gợn sóng là sóng âm lan ra ngoài theo mọi hướng. Bất kỳ nguồn âm thanh nào (tiếng còi xe, chuông điện thoại di động, tiếng chim hót hoặc dây đàn guitar gảy) đều phát ra sóng rung động của áp suất không khí với tốc độ khoảng 343 mét 1 giây (tốc độ âm thanh) từ nguồn. Không phải bản thân các phân tử không khí di chuyển từ nguồn âm thanh đến tai chúng ta; mà là các chuyển động cục bộ nhỏ trong các phân tử tạo ra sự dao động trong áp suất không khí lan truyền ra ngoài trên những khoảng cách xa.

    Once those waves reach human ear, they are captured by outer ear & funneled to a seashell-shaped muscle in inner ear called *cochlea*. This muscles has tiny hairs that resonate at different frequencies causing messages to get sent to brain that we interpret as sound.

    Remark 1 (Protect your hearing)**.** *As musicians or music producers, your sense of hearing is 1 of your most precious assets. Always wear ear protection when you're exposed to loud sustained sounds! Loud sounds can damage your inner ear permanently, meaning you can start to close your ability to hear.*

    All sound waves have following properties: *frequency, wavelength, & amplitude.*

  - 3.2. Frequency. Frequency refers to number of times a complete waveform passes through a single point over a period of time or how fast wave is vibrating. It is measured by cycles per sec in a unit called *hertz* (Hz). 1 cycle per sec is equivalent to 1 Hz, & 1000 cycles are equivalent to 1000 Hz, or 1 kHz (pronounecd kilohertz). Higher frequency, higher pitch of sound.

    Fig. 3.1: Sound is made up of compression waves of air molecules that expand outward at a speed of around 343 m/s. Frequency

of a sound wave refers to how fast it vibrates: amplitude refers to intensity of sound; & wavelength refers to length of 1 complete cycle of waveform.

- 3.3. Wavelength. Wavelength refers to length of 1 complete cycle of a wave in physical space. This is distance from 1 peak or zero crossing to next. Can't actually see sound waves, but wavelength can be calculated by dividing speed of sound (≈ 343 m/s) by its frequency. So, for pitch of a *Concert A* note (440 Hz), length of waveform would be ≈ 78 cm or 2.6 ft.

$$\frac{343 \text{ m/s}}{440 \text{ Hz}} = 0.78 \text{ m} = 78 \text{ cm} = 2.56 \text{ ft}.$$

On most pianos, wavelength of lowest bass note is almost 40 feet long! In contrast, wavelength of highest note is only around 3 inches. Longer wavelength, lower the note.

Lower frequency sound also tends to travel longer distances. Think of a car playing loud music. As it approaches, you can hear fat sound of a bass guitar or a kick drum long before you can hear other instruments. Using this property, people in West Africa were able to transmit detailed messages over long distances using a language of deep drum sounds. A drummer called a "carrier" would drum out a rhythmic pattern on a huge log drum that carried messages like "all people should gather at market place tomorrow morning". All those within hearing range, which under ideal conditions could be as far as 7 miles, would receive message.

– Âm thanh tần số thấp hơn cũng có xu hướng truyền đi xa hơn. Hãy nghĩ đến 1 chiếc ô tô đang phát nhạc lớn. Khi nó đến gần, bạn có thể nghe thấy âm thanh to của 1 cây đàn ghi-ta bass hoặc trống đá rất lâu trước khi bạn có thể nghe thấy các nhạc cụ khác. Sử dụng đặc tính này, người dân ở Tây Phi có thể truyền tải các thông điệp chi tiết trên những khoảng cách xa bằng ngôn ngữ của âm thanh trống sâu. 1 tay trống được gọi là "người mang" sẽ đánh 1 mẫu nhịp điệu trên 1 chiếc trống gỗ lớn mang theo các thông điệp như "tất cả mọi người nên tập trung tại chợ vào sáng mai". Tất cả những người trong phạm vi nghe được, trong điều kiện lý tưởng có thể cách xa tới 7 dặm, sẽ nhận được thông điệp.

- 3.4. Amplitude. Amplitude is related to volume of a sound, or how high peaks of waveform are Fig. 3.1. You can think of this as how much energy passes through a fixed amount of space over a fixed amount of time. Human ear perceives a vast range of sound levels, from sounds that are softer than a whisper to sounds that are louder than a pain-inducing jackhammer. In order to communicate volume of sound in a manageable way, music producers & engineers use a unit of loudness called *decibels* (dB). Whispered voice level might be 30 dB, while jackhammer sound would be about 110 dB. Loud noises > 120 dB can cause immediate harm to ears.

– Biên độ liên quan đến âm lượng của âm thanh hoặc độ cao của các đỉnh sóng Hình 3.1. Bạn có thể nghĩ về điều này như lượng năng lượng đi qua 1 lượng không gian cố định trong 1 khoảng thời gian cố định. Tai người cảm nhận được 1 phạm vi rộng lớn các mức âm thanh, từ âm thanh nhẹ hơn tiếng thì thầm đến âm thanh to hơn tiếng búa khoan gây đau đớn. Để truyền đạt âm lượng âm thanh theo cách dễ quản lý, các nhà sản xuất âm nhạc & kỹ sư sử dụng 1 đơn vị độ lớn gọi là *decibel* (dB). Mức giọng nói thì thầm có thể là 30 dB, trong khi âm thanh của búa khoan sẽ là khoảng 110 dB. Tiếng ồn lớn > 120 dB có thể gây hại ngay lập tức cho tai.

- 3.5. Dynamics. Variation of amplitude levels from low to high within a musical composition is referred to as dynamics. Difference between softest sound to loudest sound is called *dynamic range* of music. You can look at *waveform* of an audio signal to get a quick sense for its dynamic range. In general, lower heights mean lower amplitude & higher heights mean higher amplitude. Loudness of a sound is also dependent on frequency. So, looking at a waveform alone won't tell you how loud sth will sound to listeners (Fig. 3.3: A waveform with varying amplitude).

– Động lực học. Sự thay đổi mức biên độ từ thấp đến cao trong 1 bản nhạc được gọi là động lực học. Sự khác biệt giữa âm thanh nhỏ nhất đến âm thanh to nhất được gọi là *dynamic range* của âm nhạc. Bạn có thể xem *waveform* của tín hiệu âm thanh để có cảm nhận nhanh về dải động của nó. Nhìn chung, độ cao thấp hơn có nghĩa là biên độ thấp hơn & độ cao cao hơn có nghĩa là biên độ cao hơn. Độ to của âm thanh cũng phụ thuộc vào tần số. Vì vậy, chỉ xem dạng sóng sẽ không cho bạn biết âm thanh nào đó sẽ to như thế nào đối với người nghe (Hình 3.3: Dạng sóng có biên độ thay đổi).

- 3.6. Bandwidth. Bandwidth refers to range of frequencies present in audio. As in case of dynamic range, can think of this as difference between highest & lowest frequencies. Humans with good hearing can distinguish sounds between 20 Hz & 20000 Hz. Most audio formats designed for music support frequencies up to 22 kHz (pronounced 22 kilohertz or 22000 hertz) so that they can capture full range of human hearing.

– Băng thông đề cập đến phạm vi tần số có trong âm thanh. Giống như trường hợp của phạm vi động, có thể coi đây là sự khác biệt giữa tần số cao nhất & thấp nhất. Con người có thính giác tốt có thể phân biệt âm thanh giữa 20 Hz & 20000 Hz. Hầu hết các định dạng âm thanh được thiết kế cho âm nhạc đều hỗ trợ tần số lên đến 22 kHz (phát âm là 22 kilohertz hoặc 220000 hertz) để chúng có thể thu được toàn bộ phạm vi thính giác của con người.

Musical instruments naturally fall within range of human hearing at different places on frequency spectrum; this is referred to as instrument's bandwidth. *Instrument bandwidth* is important to music producers as they arrange a musical composition. In addition to quality of sound of instruments, those in different bandwidths can complement each other. Like a cello & a flute, or a bass & a saxophone. Music producers are keenly aware of influence of low- & high-frequency instruments on their listeners. Musical instruments in bass register are often foundation of composition, holding everything together.

– Nhạc cụ tự nhiên nằm trong phạm vi nghe của con người ở các vị trí khác nhau trên phổ tần số; điều này được gọi là băng thông của nhạc cụ. *Băng thông nhạc cụ* rất quan trọng đối với các nhà sản xuất âm nhạc khi họ sắp xếp 1 bản nhạc. Ngoài chất lượng âm thanh của các nhạc cụ, những nhạc cụ có băng thông khác nhau có thể bổ sung cho nhau. Giống như đàn cello & sáo, hoặc đàn bass & saxophone. Các nhà sản xuất âm nhạc nhận thức sâu sắc về ảnh hưởng của các nhạc cụ có tần số thấp & cao đến người nghe của họ. Nhạc cụ có âm trầm thường là nền tảng của bản nhạc, giữ mọi thứ lại với nhau.

- 3.7. Pitch. Within spectrum of human hearing, specific frequencies, ranges of frequencies, & combinations of frequencies are essential for creating music. This sect covers some combinations of musical tones common in Western music culture. Then work in TunePad to try out different combinations & explore those relationships through well-known musical compositions.

  – Trong phổ thính giác của con người, các tần số cụ thể, phạm vi tần số, & sự kết hợp của các tần số là điều cần thiết để tạo ra âm nhạc. Giáo phái này bao gồm 1 số sự kết hợp của các giai điệu âm nhạc phổ biến trong văn hóa âm nhạc phương Tây. Sau đó, hãy làm việc trong TunePad để thử các sự kết hợp khác nhau & khám phá các mối quan hệ đó thông qua các tác phẩm âm nhạc nổi tiếng.

  While music producers & engineers often think in terms of frequencies (hertz), musicians use pitch & intervals to describe musical tones & relationships between them. Pitches are individual notes like F, G, A, B, C, D, E as seen on piano keyboard. Interval between each adjacent note on a traditional keyboard is called a half step or a semitone. These base pitches can also have *accidentals*. Accidentals are like modifiers to notes that raise or lower base pitch. A note with a sharp # applied has its pitch raised by a semitone, which a note with a flat ♭ applied is lowered by a semitone. Black notes on a piano are notes with accidentals. E.g., moving a C# (black key) is a half step. Moving directly from a C to a D (both white keys) is called a whole step. Moving from a B to a C or an E to an F is also a half step because there's no black key in between (Fig. 3.4: A half step is distance between 2 adjacent piano keys, measured in semitones.)

  – Trong khi các nhà sản xuất âm nhạc & kỹ sư thường nghĩ theo tần số (hertz), thì các nhạc sĩ sử dụng cao độ & khoảng cách để mô tả các cung bậc âm nhạc & mối quan hệ giữa chúng. Cao độ là các nốt riêng lẻ như F, G, A, B, C, D, E như thấy trên bàn phím piano. Khoảng cách giữa mỗi nốt liền kề trên bàn phím truyền thống được gọi là nửa cung hoặc nửa cung. Các cao độ cơ bản này cũng có thể có *dấu hóa ngẫu nhiên*. Dấu hóa ngẫu nhiên giống như các dấu hiệu bổ nghĩa cho các nốt làm tăng hoặc giảm cao độ cơ bản. 1 nốt có dấu thăng # được áp dụng có cao độ được tăng lên 1 nửa cung, trong khi 1 nốt có dấu giáng ♭ được áp dụng sẽ hạ xuống 1 nửa cung. Các nốt đen trên đàn piano là các nốt có dấu hóa ngẫu nhiên. E.g., di chuyển 1 C# (phím đen) là nửa cung. Di chuyển trực tiếp từ C sang D (cả hai đều là phím trắng) được gọi là 1 cung trọn vẹn. Di chuyển từ B sang C hoặc từ E sang F cũng là nửa cung vì không có phím đen nào ở giữa (Hình 3.4: Nửa cung là khoảng cách giữa 2 phím đàn piano liền kề, được đo bằng nửa cung.)

- 3.8. Musical Instrument Digital Interface. 1 takeaway from prev sect: note names are confusing. There are multiple names for same pitch (G# is same as A♭), & note names are repeated every octave. To help make things less ambiguous, computers & digital musical instruments use a standardized format called *MIDI*, which stands for Musical Instrument Digital Interface. MIDI is a protocol, or set of rules, for how digital musical instruments communicate. Digital musical instruments send message to your computer or to other musical instruments. Typical MIDI controllers look like piano keyboards or drum pads but can take many other forms as well. When play a MIDI instrument, it sends information about a note's pitch, timing, & volume along with other messages about vibrato, pitch bend, pressure, panning, & clock signals. This table show 2 octaves of notes with their typical frequency values [Table]. Appendix contains a complete table with note names, frequency values, & MIDI numbers.

  TunePad uses MIDI numbers to designate pitch. To play a C0, lowest pitch on TunePad keyboard, use code `playNote(12)`. To play a C4, a middle C in center of an 880key piano, use code `playNote(60)`. MIDI notes go all way up to note G9 with note value 127.

  Now experiment with pitch in TunePad. Try creating a new piano instrument in TunePad & adding this code:

```
# code for first piano cell
playNote(48)
playNote(55)
playNote(60)
playNote(55)
```

  This program plays 4 notes: 48 is a C, 55 is a G, & 60 is a middle C. Now add a 2nd piano instrument to same project so that you have 2 cells. Add this code to 2nd cell:

```
# code for second piano cell
playNote(72, beats = 4) # C5
playNote(79, beats = 4) # G5
playNote(76, beats = 4) # E5
playNote(79, beats = 4) # G5
```

  This Python program looks similar to 1st one, but we've changed length of each note using *beats* parameter. In this case, asking TunePad to play 4 notes, each 4 beats long. Try playing both piano parts at same time. Can also make our notes shorter instead of longer. Add a 3rd piano instrument with notes that are each 1 half beat long. Try playing all 3 pianos together.

```
# code for third piano cell
playNote(36, beats = 0.5)
playNote(36, beats = 0.5)
playNote(43, beats = 0.5)
playNote(43, beats = 0.5)
```

```
playNote(48, beats = 0.5)
playNote(48, beats = 0.5)
playNote(43, beats = 0.5)
playNote(43, beats = 0.5)
```

- 3.9. Harmony. *Harmony* in music can be defined as a combination of notes that, when played together, have a pleasing sound. Although opinions about what sounds good in music are highly subjective, certain combinations of notes played together can elicit predictable psychological responses – some combinations of notes sound *harmonious* while others some *discordant*. Musicians use this phenomenon to create an emotional tone for their compositions.

  – *Harmony* trong âm nhạc có thể được định nghĩa là sự kết hợp các nốt nhạc, khi chơi cùng nhau, tạo ra âm thanh dễ chịu. Mặc dù ý kiến về những gì nghe hay trong âm nhạc là rất chủ quan, nhưng 1 số sự kết hợp các nốt nhạc chơi cùng nhau có thể gợi ra những phản ứng tâm lý có thể dự đoán được – 1 số sự kết hợp các nốt nhạc nghe *hài hòa* trong khi 1 số khác lại *không hài hòa*. Các nhạc sĩ sử dụng hiện tượng này để tạo ra giai điệu cảm xúc cho các sáng tác của họ.

  In Western music, much of our conception of pitch is built on different mathematical ratios. Consider string of an instrument like a guitar or violin. Plucking open A (2nd lowest) string plays an A, which has a frequency of 110 Hz. Now if touch string at its midpoint, dividing it in half, still hear an A an octave above previous one – twice frequency of 1st note, or 220 Hz. If touch string $\frac{1}{3}$ of way down & pluck it, result is an E above higher A. This E is exactly 3 times our original frequency, or 330 Hz. Likewise, dividing string into 4ths multiplies original frequency by 4. Can continue this division on string as follows Fig. 3.5: harmonic series.

  – Trong âm nhạc phương Tây, phần lớn quan niệm của chúng ta về cao độ được xây dựng dựa trên các tỷ lệ toán học khác nhau. Hãy xem xét dây của 1 nhạc cụ như đàn ghi-ta hoặc đàn violin. Gảy mở dây A (dây thấp thứ 2) sẽ tạo ra nốt A, có tần số 110 Hz. Bây giờ nếu chạm vào dây ở điểm giữa của nó, chia nó thành hai nửa, vẫn nghe thấy nốt A cao hơn 1 quãng tám so với nốt trước đó – gấp đôi tần số của nốt đầu tiên, hoặc 220 Hz. Nếu chạm vào dây $\frac{1}{3}$ xuống 1 khoảng & gảy nó, kết quả là nốt E cao hơn nốt A cao hơn. Nốt E này chính xác gấp 3 lần tần số ban đầu của chúng ta, hoặc 330 Hz. Tương tự như vậy, chia dây thành 4 quãng sẽ nhân tần số ban đầu với 4. Có thể tiếp tục phép chia này trên dây như sau Hình 3.5: chuỗi điều hòa.

  Resulting sequence of ascending pitches this produces is known as *harmonic series*. If 2 notes have a harmonic relationship, i.e., 2 frequencies s.t. result is a whole number. Harmonic series is simply set of frequencies that have a harmonic relationship to a *fundamental pitch* (initial note). Our initial experiment with string illustrates this relationship.

  To find frequencies that make up harmonic series for a given pitch, multiply its frequency by set of whole numbers. For A1, which is 55 Hz, 1st 8 harmonics would be [Table].

  In table, MIDI value is given for each harmonic of A1. Notice these values are given in decimal format. In TunePad, `playNote` accepts both whole & decimal values. Whole numbers are a data type referred to as *integer* values, or just as *ints*. Decimals are a separate data type referred to as *floating point* values, or just *floats*.

  Listen to an example https://tunepad.com/examples/harmonic-series. Notes with frequencies that form simple ratios, e.g. 2:1, 3:2, 4:3, or 5:4, tend to sound good together. E.g., can take note A4 (440 Hz) & add a frequency that is 1.5 times its value, giving us an E4 (660 Hz). This results in a ratio of 3:2 & a pleasant sound. However, if add a frequency that is 1.3 times value of 440 Hz, end up with 572 Hz, which creates a not-so-pleasant combination of tones. It's not an accident that there is no corresponding musical note to 572 Hz on piano keyboard.

- 3.10. Intervals. In music, an *interval* is distance between 2 notes. These notes can either be played simultaneously or not. If they are played simultaneously, pitches are called a *dyad* or a *chord*. Otherwise, they are a *melodic* interval. An interval is always measured from lowest note. Intervals have 2 different components: *generic interval* & quality. Generic interval is distance from 1 note of a scale to another; this can also be described as number of letter names between 2 notes, including both notes in question. E.g., generic interval of C4 & E4 has C4, D4, & E4 in between. That's 3 notes, so we have a 3rd. Generic interval between F#3 & G3 is a second. Generic interval between G2 & G3 in an 8th – also known as an octave. Quality can be 1 of 5 options: Perfect, Major, Minor, Augmented, or Diminished. Each quality has a distinct sound & can generate different emotional responses. Some common intervals in music along with their frequency ratios & half steps. [Table]

  These intervals are based on harmonic series, but this isn't exactly how most instruments are tuned. Talk more about this below. Also, naming of ratios (5th, 4th, Major 3rd, etc.) will make more sense in Chap. 5 where talk about scales & keys. Notice that there's 1 particularly nasty-looking ratio called *Tritone interval* (45:32). This interval has historically been referred to as *Devil in Music* & was frequently avoided in music composition for its dissonant qualities.

  – Các khoảng này dựa trên chuỗi hài hòa, nhưng đây không phải là cách chính xác mà hầu hết các nhạc cụ được lên dây. Hãy nói thêm về điều này bên dưới. Ngoài ra, việc đặt tên cho các tỷ lệ (5, 4, 3 trưởng, v.v.) sẽ hợp lý hơn trong Chương 5, nơi nói về các thang âm & cung. Lưu ý rằng có 1 tỷ lệ trông đặc biệt khó chịu được gọi là *Quãng ba cung* (45:32). Khoảng này trước đây được gọi là *Devil in Music* & thường bị tránh trong sáng tác nhạc vì tính chất bất hòa của nó.

  1 of simplest & most common intervals is octave, which has a frequency ratio of 2 to 1 (2:1) – i.e., higher pitch completes 2 cycles in same amount of time that lower pitch completes 1 full cycle. Notes that are octave intervals from 1 another have same letter name & are grouped together on a piano keyboard. Notice repeating patterns where C is highlighted note, C3, C4, C5 (Fig. 3.4: A half step is distance between 2 adjacent piano keys, measured in semitones.). To illustrate, can begin with *middle C* (C4), which is $\approx$ 262 Hz, & then move to a C5, which is an octave above it at 524 Hz. Can see C5 is double C4

frequency forming octave ratio, 2:1. Waveforms representing 2 notes forming this octave are plotted in Fig. 3.6: 2 waves at an interval of an octave. Can see that for every single complete cycle of 262 Hz wave, C4, there are 2 full cycles of waveform for octave above it, 524 Hz C5. Easier to count cycles if look at 0-crossings.

What does an octave look like in code? As have seen, each note on piano keyboard is a half step, & there are 12 half steps between octaves. Try counting notes between C4 & C5. Remember, black keys count!

TunePad tracks notes on keyboard by half steps, so can easily play any octave interval without having to figure out exact note number. E.g., this code plays a middle C (60) & a C 1 octave higher.

```
note = 60
playNote(note)
playNote(note + 12)
```

This code assigned number 60 to variable `note` on 1st line. 3rd line played a note 1 octave higher by adding 12 to original `note` variable (not 72 is played). Expanding on this, can substitute any number you want for `note` & generate an octave above it by adding 12.

Octaves sound good together in music & are used in many popular songs. E.g., in song *Over the Rainbow* composed by Harold Arlen from *Wizard of Oz*, beginning 2 notes are an octave apart.

```
# First two bars of "Over the Rainbow"
# Composed by Henry Arlen
playNote(60, beats = 2) # note C4
playNote(60 + 12, beats = 2) # note C5
playNote(71, beats = 1)
playNote(67, beats = 0.5)
playNote(69, beats = 0.5)
playNote(71, beats = 0.5)
rest(0.5)
playNote(72, beats = 1)
```

Try this example at https://tunepad.com/examples/rainbow.

Another interval relationship important to Western music is ratio of 3:2, also known as perfect 5th, which has 7 half steps between notes. With this interval ratio, there are 3 complete cycles of higher frequency for every 2 periods of lower frequency (Fig. 3.7: Ratio between note C 262 Hz & note G 393 Hz is considered a perfect 5th.)

Henry Mancini uses a perfect 5th (G3 392 Hz & D4 587 Hz) in 1st 2 notes in melody for song *Moon River*. Code 1st few bars of *Moon River* using a variable called `root_note` to set starting note. This allows us flexibility to easily play song beginning from any note on keyboard & relationship between notes stays same no matter which note you start with. Try changing value of variable `root_note` to another MIDI note. This can come in handy when you are composing for a singer who would rather have song in another key or octave.

```
# First bars of "Moon River"
# Composed by Henry Mancini
root note = 55
playNote(root note, beats = 3)
playNote(root note + 7, beats = 1)
playNote(root note + 5, beats = 2)
playNote(root note + 4, beats = 1.5)
playNote(root note + 2, beats = 0.5)
playNote(root note, beats = 0.5)
playNote(root note - 2, beats = 0.5)
playNote(root note, beats = 2)
```

See this example at https://tunepad.com/examples/moon.

- 3.11. Dissonance. Dissonance refers to combinations of notes which when combined have an unpleasant sound that creates tension. Interval of a minor second (or 1 half step) is a complex frequency ratio of about 9.5:1. This combination gives you a sense of suspense. Can hear effect of dissonance used in composition by John Williams for movie *Jaws*. Use `for` loop for this example, as 2 notes are repeated.

```
# bass line for the theme from Jaws
# composed by John Williams
for i in range(8):
    playNote(40, beats = 0.5) # E2
    playNote(41, beats = 0.5) # F2
```

Intervals that are dissonant are unstable, leaving listener with impression that notes *want* to move elsewhere to resolve to more stable or *consonant* intervals.

Can try this example in TunePad to hear how notes that are 1 half step apart crunch when played together.

```
# half step - the notes are just 1 number away
playNote(41, beats=1, sustain=3)
playNote(42, beats=1, sustain=2)
rest(2)
playNote([41, 42], beats=4)
rest(2)

# whole step - these notes are 2 numbers away
playNote(41, beats=1, sustain=3)
playNote(43, beats=1, sustain=2)
rest(2)
playNote([41, 43], beats=4)
rest(2)
```

Listen to this example https://tunepad.com/examples/dissonance.

Another example of use of dissonant intervals comes from horror movie *Halloween* (1978). Theme song by John Carpenter creates a sense of suspense & deep unease with use of dissonant intervals e.g. Tritone (ratio 45:32).

- 3.12. Temperaments & Tuning. Follow along at https://tunepad.com/examples/temperaments .

Intervals in prev sects were based on ratios called perfect or pure intervals. Waves of so-called perfect intervals align at a simple integer ratio. If 2 tones form a perfect interval, it will result in a louder sound, as amplitudes are added. If 1 of tones is out of tune, then there will be interference between 2 waves. This interference manifests as an audible rhythmic swelling or "wah-wah" between waves, which we call *beating*. Farther 2 tones are from being perfect, faster beating. If tones are apart far enough, might even hear this beating as a 3rd tone, called a *combination tone*. Pure & impure intervals are not a value judgment but a description of natural phenomena.

Using notes based on these simple ratios seems to make a lot of sense – it's based on simple mathematical relationships that we know sound good to human ear. But, it turns out: quickly run into problems using this system when start trying to tune an instrument like a piano. E.g., say trying to tune an A4 against a fixed lower tone on a keyboard using pure ratios. If tuning this A4 against an F4 at $\approx$ 349 Hz, our intervals form a major 3rd at a 5:4 ratio of frequencies. This results in our A4 being $\approx$ 436.26 Hz. But, if tune our A4 against an F#4 at 370 Hz, this produces a minor 3rd, which is at a 6:5 ratio of frequencies. Now our A4 is 444 Hz instead of 436.25 Hz! How can it be that same note maps to different frequencies?

Question of how to map frequency – of which there are endless possible values – to a fine set of notes means that we have to both arbitrarily choose a starting point & also decide at what intervals to increment. This is basis for what are called *temperaments*. Temperaments are systems that define sizes of different intervals – how tones relate to 1 another. In choosing tones in an octave, must compromise between our melodic intervals & our harmony. Ideally, want a system with as consistent melodic intervals & that is as close to perfect harmonic intervals as possible. In a system based on perfect ratios – also referred to *Just Intonation* – divisions, or semitones, of an octave are not evenly distributed. I.e., there are unique sets of tunings for every note we choose as base note of our octave. Just Intonation also does not form a closed loop of an octave. This is getting into weeds a bit, but if derive each note's frequency by tuning ratio of a perfect 5th (3:2) from prev note, do not end up at same place 1 octave higher. In fact, tuning ratio of 3:2 12 times brings us back to our exact starting note only after 7 octaves. Because Just Intonation has too many mathematical snares to be represented by 12 notes of keyboard, it's not a stable tuning system & not a temperament. By definition, a temperament is a calculated deviation from Just Intonation that maps each note to exactly 1 frequency while still getting as close as possible to pure intervals.

Most contemporary music, including TunePad, is based on a system called Equal Temperament. Octave at a pure 2:1 ratio serves as foundation, which is then divided into 12 equal half steps. Most often, Western harmony is built primarily from 3rds, 5ths, & octaves. Every octave (& unison) is a pure interval in Equal Temperament. Perfect 4ths & 5ths are *nearly* pure intervals. Major & minor 3rds are quite far from perfect, but because we have grown so accustomed to hearing these intervals, they do not sound off to our ears. Because Equal Temperament is, well, equal, every chord will have same sound in every key. Each semitone is equally sized, & every note maps to exactly 1 frequency. Furthermore, each semitone is divided into 100 cents, which we can use to further specify intonation. With our intervals decided, now only have to choose a starting pitch from which to tune others. Most of time in North America, system is aligned to A440, i.e. A4 is equal to exactly 440 Hz.

Keyboard instruments have fixed pitch, while singers & instruments e.g. violin or flute have flexible tuning. In acoustic performance, pitch can vary due to many factors. No instrument is perfectly in tune. Tuning can be affected by factors e.g. air pressure & temperature. Even a performer's physiology can affect tuning. Often, performers will tune harmonies using Just Intonation s.t. a chord uses pure intervals & is more pleasing. Many musicians will do this without even being aware that they are doing it – Just Intonation just *feels* in tune.

Important to remember that decision to tune to A440 & to divide octave into 12 equal semitones is only 1 possibility in response to debates about musical tuning that date back thousands of years, & it's only 1 of a myriad of ways that music

can be tuned. There are many alternate tuning systems, both historical & contemporary from both Western & non-Western cultures, which are still in use today.

– Điều quan trọng cần nhớ là quyết định lên dây A440 & chia quãng tám thành 12 nửa cung bằng nhau chỉ là 1 khả năng để đáp lại các cuộc tranh luận về cách lên dây nhạc có từ hàng ngàn năm trước, & đó chỉ là 1 trong vô số cách để lên dây nhạc. Có nhiều hệ thống lên dây thay thế, cả lịch sử & đương đại từ cả nền văn hóa phương Tây & không phải phương Tây, vẫn được sử dụng cho đến ngày nay.

- Interlude 3: Melodies & Lists. For this interlude, code a short sect of a remix of Beethoven's *Für Elise* created by artist & YouTuber Kyle Exum (Bassthoven, 2020). Because this song has a more intricate melody, learn how to play sequences of notes written out as Python *lists*. Talk more about lists in next chap, but for now, can think of them as a way to hold > 1 note in a single variable.

1. **Step 1: Variables.** Create a new Keyboard instrument & add some variables for our different note names.

```
A = 69 # set variable A equal to 69
B = 71
C = 72
D = 74
E = 76
Eb = 75 # E flat
Gs = 68 # G sharp
_ = None
```

Last line is a little strange. It defines a variable called `_`

- In Python, underscore `_` character is a valid variable name.
- Set this variable to have a special value called `None`.
- Calling `playNote` with this value is same thing as a rest. It plays nothing.

2. **Step 2: Phrases.**
   - For this song, going to define 4 musical phrases that get repeated to make melody.
   - Each phrase gets its own variable.
   - Each variable will hold lists of notes in order they should be played.
   - You create a Python list by enclosing variables inside of square brackets.
   - Use underscore character `_` mean play nothing.
   - Sometimes subtract 12 from a note, i.e., to play note on octave lower.

```
# four basic phrases that repeat throughout
p1 = [ E, Eb, E, Eb, E, B, D, C, A, _, _, _ ]
p2 = [ A, C - 12, E - 12, A, B, _, _, _ ]
p3 = [ B, E - 12, Gs, B, C, _, _, _, C, _, _, _ ]
p4 = [ B, E - 12, C, B, A, _, _, _, A, _, _, _ ]
```

3. **Step 3: Playing Phrases.** Now have defined our variables, can start to play melody. 1 way to do this: use a Python *for loop* to iterate through every note. 1 cool thing about Python: can join lists together using plus sign +. Here's what everything looks like together.

```
_ = None
A = 69
B = 71
C = 72
D = 74
E = 76
Eb = E - 1 # E flat
Gs = A - 1 # G sharp

# 4 basic phrases that repeat throughout
p1 = [ E, Eb, E, Eb, E, B, D, C, A, _, _, _ ]
p2 = [ A, C - 12, E - 12, A, B, _, _, _ ]
p3 = [ B, E - 12, Gs, B, C, _, _, _, C, _, _, _ ]
p4 = [ B, E - 12, C, B, A, _, _, _, A, _, _, _ ]
p5 = [ A,_,_,_,A,_,_,_,A,_, _, _, A, _, _, _ ]

for note in p1 + p5 + p2 + p3 + p1 + p2 + p4:
```

```
    playNote(note, beats = 0.5)

for note in p1 + p2 + p3 + p1 + p2 + p4:
    playNote(note, beats = 0.5)
```

4. **Step 4: Bass!** Add a Bass to your project & change voice to `808 Bass`. Can copy code below for bass pattern Fig. 3.8: Select 808 Bass voice.
5. **Step 5: Drums.** To finish up, layer in a simple drum pattern that works well with melody. Create a new `Drum instrument` & add this code.

```
rest(12)
for i in range(4):
    playNote(21)
    rest(2)
    playNote(21)
    rest(1)
    playNote(16, beats = 0.5)
    rest(1)
    playNote(16, beats = 0.5)
    rest(1)
    playNote(21)
    rest(2)
    playNote(21)
    rest(1)
    playNote(28, beats = 0.5)
    rest(1)
    playNote

for i in range(16):
    playNote(0)
    playNote(2, beats = 0.5)
    playNote(2, beats = 0.5)
    playNote(10)
    playNote(0)
```

Can try this project online https://tunepad.com/interlude/bassthoven.

- 4. Chords – Hợp âm. Chords are an essential building block of musical compositions. Skillful use of chords can set foundation of a song & create a sense of emotional movement. However, even though basic ideas behind chords are easy to understand, there's an overwhelming amount of terminology & technical detail that can take years to learn. Using code helps us cut through layers of complicated terminology to reveal elegant structures beneath. With code, chords are nothing more than lists of numbers that follow consistent patterns. Work through chords using Python *lists & functions*. Learn some of traditional music terminology & what it means, but also build your own toolkit of computer code to use for new compositions.
  - 4.1. Chords. Can follow along with interactive online examples at https://tunepad.com/examples/chord-basics. In Chap. 3, introduced idea of harmony & dissonance. ≥ 2 notes have a harmonic relationship if their frequencies have integer ratios. E.g., when 2 notes are 1 octave apart, higher note vibrates exactly 2 complete cycles for every 1 cycle of lower note (a 2:1 ratio). When 2 notes are a 5th apart, their frequencies have a 3:2 ratio. Higher note vibrates exactly 3 times for every 2 complete cycles of lower note.

    Building on this idea of harmonic relationships between notes, a *chord* is more than 1 note played together at same time. In Python can think of a chord as a *list* of numbers representing MIDI (Musical Instrument Digital Interface) note values. E.g., this code plays a C major chord in TunePad.

```
Cmaj = [ 48, 52, 55 ] # notes C, E, G
playNote(Cmaj)
```

    Here `Cmaj` is a variable. Instead of assigning that variable to a single number, we're assigning it a *list* of numbers. In Python, a list is a set of values enclosed in square brackets & separated by commas. Then on 2nd line we play all 3 notes together using `Cmaj` variable Fig. 4.1: C major chord with MIDI note numbers.

    Can also play same chord using just a single line of code where pass list of numbers directly to `playNote` function.

```
playNote([ 48, 52, 55 ])
```

    But, using variable is nice because it helps make our code easier to read & understand. Here are a few other chord examples:

```
Cmaj = [ 48, 52, 55 ] # C major chord
Fmaj = [ 53, 57, 60 ] # F major chord
Gmaj = [ 55, 59, 62 ] # G major chord
```

A chord's name comes from 2 parts. 1st part is *root* note of chord – usually 1st note in list. E.g., a F major chord starts with note 53 (or an F on piano keyboard). & G major chord starts with note 55, a G on keyboard.

2nd part of a chord's name is its type or *quality*. In our examples, `Cmaj, Fmaj, Gmaj` are all *major* chords. Later in this chap review several common chord types & how to create them in code. Each chord type has a consistent pattern. E.g., all major chords follow exact same pattern: take root note, add 4 to get 2nd note, & then add 7 to get last note. Can build a major chord up from any base note you want as long as it follows this same pattern Fig. 4.2: Creating chords as lists of numbers in Python. Each major chord follows same pattern.

Another way to write this in code: define a single root note variable & then create chord based on root:

```
root = 48
playNote([ root, root + 4, root + 7 ]) # C major
root = 52
playNote([ root, root + 4, root + 7 ]) # F major
```

1 thing to notice about this code: 1st 2 lines & last 2 lines are almost identical. Just changing root note value. In fact, all we need to make any chord we want is its root note & pattern that defines its quality. Once we know patterns, rest is easy.

But, it would be tedious & error prone to write out `[root, root + 4, root + 7]` every time we wanted to use a major chord. Fortunately, Python gives us a powerful tool for exactly this kind of situation: *user-defined functions*.

- 4.2. User-defined functions. In 1st few chaps of this book, using functions to play music: `playNote, rest, moveTo` are all functions provided to TunePad. When want to use a function, just type its name & list parameters to send it inside parentheses. With Python, can also create our own functions to build up a musical toolkit. Creating functions also helps make code shorter & easier to understand because we'll be able to use same segments of code over & over again without having to copy & paste. A quick example that creates a major chord based on a root note.

```
def majorChord(root):
    chord = [ root, root + 4, root + 7 ]
    return chord
```

Now that have defined function, can use it as a shortcut in TunePad.

```
Cmaj = majorChord(48)
Fmaj = majorChord(53)
playNote(Cmaj, beats = 2)
playNote(Fmaj, beats = 2)
```

There's a lot going on with these few lines of code. Break it down line by line.

* Line 1 starts with `def` keyword. This is short for "define", & it tells Python that we're about to define a function.
* Next is name of function we're defining. In this case, calling it `majorChord`, but we could use any name want as long as it follows Python's naming rules.
* After function name, need to list out all of function's *parameters* enclosed inside parentheses. In this case, there's only 1 parameter called *root*. If need > 1 parameter, separate each parameter with commas. Can think of parameters as special kinds of variables that are only usable inside of a function.
* After parameter list, need colon character :. This tells Python: *body* of function is coming next Fig. 4.3: How to declare a user-defined function in Python.
* Line 2 starts body of function. Here just creating a variable called *chord* & assigning it to a list of numbers that define a major chord. Numbers are root note, root note +4, & root note +7. An important thing to notice: this line of code is intended by 4 spaces. Just like with loops, indentation tells Python: these lines are part of function body. They're considered to be inside function, not outside.
* Line 3 is also indented by 4 spaces because it's also part of function body. This line uses special `return` keyword to say what value function produces. In this case, returning `chord` variable, list of 3 numbers that make up our major chord. When Python gets to `return` keyword, it immediately exits function & returns value given on that line.
* Also possible to define a function with no return value. In this case, Python provides a special return value called `None`.

That's it for our function def. Now can see how it's used on lines 4–7. Notice can call our new function multiple times to generate different chords, letting us reuse code to create more readable & elegant programs. In rest of chap, use this same template to begin to build up a library of chord types, each with its own function.

Can define variables inside a function just like you might otherwise. A variable's *scope* refers to where we can use this variable or function. Can think of this as level of indentation for a function. Variables defined within def of a function can only be used in that function; their scope is within that function. Following code will result in an error on line 4.

```
def majorChord(root):
    chord = [ root, root + 4, root + 7 ]
majorChord(60)
playNote(chord) # ERROR
```

`chord` variable lives only in our `majorChord` function. Can refer to this as a local variable. Alternatively, there are global variables which can be used anywhere after they've been defined. In code below, `root, chord` are global variables, & therefore can be freely used in body of any functions:

```
root = 60
chord = [ root, root + 4, root + 7 ]
def songPar1():
    playNote(chord)
    playNote(root)
songPart1()
```

Before move on, let's define 1 more function that shows how we can use our new functions inside of other functions:

```
def playMajorChord(root, duration):
    chord = majorChord(root)
    playNote(chord, beats = duration)
```

This function uses our `majorChord` function to both build a chord from a root note & then calls `playNote` to play chord. This new `playMajorChord` function takes 2 parameters, `root` note & a `duration` that says how long to play note.

Note 2. *By convention, Python function & variable names use lowercase letters, with different words separated by underscore character, e.g.,* `play_major_chord`. *This style is referred to as "*`snake_case`*". However, this book uses a different style called "camelCase" where names start with a lowercase letter & uppercase letters are used to start new words (as in* `playMajorChord`*). Use camelCase in this book to be consistent with other programming language conventions e.g. JavaScript, but should feel free to use* `snake_case` *for your own Python programs if want to.*

- 4.3. Common chord types. This sect reviews some of most commonly used chord types in modern musical genres. For each chord, provide pattern of numbers that defines its quality, an example of a chord of that type on piano keyboard, & a TunePad function that generates chords from a root note. Also describe chord in terms of musical intervals. Names for intervals can be a little confusing, especially when combined with note names or MIDI note values.

  Common chord types/qualities.

  * Major triad: Major 7th: Suspended 2
  * Minor: Minor 7th: Suspended 4
  * Diminished: Dominant 7th: Augmented

  To hear these chords in action, go to https://tunepad.com/examples/chord-functions.

  * 4.3.1. Major triad. Major chords are commonly described as cheerful & happy. They consist of a root note, root + 5, & root + 7. In music theory, 2nd note of a major triad is called a *major 3rd*, & 3rd note is called a *perfect 5th*. This can be referred to as a *triad* due to fact there there are 3 distinct notes Fig. 4.4: C major chord.
    - Pattern: [0, 4, 7]
    - Intervals: Major 3rd, Perfect 5th
    - Notation: C major, CMaj, CM, C
    - Python Function:

      ```
      def majorChord(root):
          return [ root, root + 4, root + 7 ]
      ```

  * 4.3.2. Minor triad. Minor chords convey more of a somber tone. They're very similar to major chords except that 2nd note adds 3 to root note instead of 4. In music theory, 2nd note is called a *minor 3rd* (instead of a major 3rd). But, even with this small change, difference in mood is dramatic Fig. 4.5: D minor chord.
    - Pattern: [0, 3, 7]
    - Intervals: Minor 3rd, Perfect 5th
    - Notation: D minor, Dmin, Dm C
    - Python Function:

      ```
      def minorChord(root):
          return [ root, root + 3, root + 7 ]
      ```

* 4.3.3. Diminished triad. Diminished chords instill tension & instability in music. They're often used as a way to transition between chords in a progression. There are a few different types of diminished chords, but simplest, 3-note variety, is almost identical to minor triad except last note is decreased (diminished) by 1. This is called a *diminished 5th interval* Fig. 4.6: B diminished chord.
  · Pattern: [0, 3, 6]
  · Intervals: Minor 3rd, Diminished 5th
  · Notation: B dim, B°
  · Python Function:

```
def diminishedTriad(root):
    return [ root, root + 3, root + 6 ]
```

* 4.3.4. Major 7th. Major 7th chord starts with a major triad & then adds a 4th note to end of list (root + 11). This addition is called a *major 7th interval*. Extra note creates a more sophisticated & contemplative feeling Fig. 4.7: C major 7th chord.
  · Pattern: [0, 4, 7, 11]
  · Intervals: Major 3rd, Perfect 5th, Major 7th
  · Notation: $\text{Cmaj}^7, \text{CM}^7, \text{CMa}^7$
  · Python Function:

```
def major7th(root):
    return [ root, root + 4, root + 7, root + 11 ]
```

* 4.3.5. Minor 7th. A minor 7th starts with a minor triad & adds a minor 7th (root + 10). This chord is a bit more moody than major 7th in feel Fig. 4.8: D minor 7th chord.
  · Pattern: [0, 3, 7, 10]
  · Intervals: Minor 3rd, Perfect 5th, Minor 7th
  · Notation: $\text{Dmin}^7, \text{Dm}^7$
  · Python Function:

```
def minor7th(root):
    return [ root, root + 3, root + 7, root + 10 ]
```

* 4.3.6. Dominant 7th. Just like major & minor 7ths, can create a dominant 7th by combining some of our earlier building blocks. Dominant 7th starts with a major triad & adds a minor 7th to get pattern $[0, 4, 7, 10]$. Combination of major & minor intervals can create a feeling of restlessness Fig. 4.9: G dominant 7th chord.
  · Pattern: [0, 4, 7, 10]
  · Intervals: Minor 3rd, Perfect 5th, Minor 7th
  · Notation: G^7
  · Python Function:

```
def dominant7th(root):
    return [ root, root + 4, root + 7, root + 10 ]
```

* 4.3.7. Suspended 2 & suspended 4. Suspended triad chords start with a major triad, but shift middle note up or down. A sus2 chord combines a root note, a major 2nd (+2), & a perfect 5th Fig. 4.10: Csus2 chord.
  · Pattern: [0, 2, 7]
  · Intervals: Major 2nd, Perfect 5th
  · Notation: Csus2, C^{sus2}
  · Python Function:

```
def sus2(root):
    return [ root, root + 2, root + 7 ]
```

  A sus4 chord shifts middle note in other direction to a major 4th (+5) Fig. 4.11: Csus4 chord.
  · Pattern: [0, 5, 7]
  · Intervals: Major 4th, Perfect 5th
  · Notation: Csus4, C^{sus4}
  · Python Function:

```
def sus4(root):
return [ root, root + 5, root + 7 ]
```

  Can try both versions of suspended chord in interactive tutorial.

* 4.3.8. Augmented triad. Last type of chord we'll cover here is called an *augmented triad*. This is just a major triad with a "sharpened 5th": last note is raised from a perfect 5th (+7) to an augmented 5th (+8). This chord can add a feeling of suspense or anxiety Fig. 4.12: C augmented chord.
  - Pattern: [0, 4, 8]
  - Intervals: Minor 3rd, Perfect 5th
  - Notation: Caug, C+
  - Python Function:

```
def augmentedTriad(root):
    return [ root, root + 4, root + 8 ]
```

  There are many, many other kinds of chords we can explore that add extra notes or use notes in different patterns. Extended chords bring in intervals like 9ths, 11ths, & 13ths, & inverted chords shift position of root note. With 88 keys on a piano keyboard & dozens of chord qualities to choose from, there are hundreds & hundreds of possible chords we can use in a given song. How do we reduce this complexity to generate music that sounds good? There are 3 answers to this question:
  - 1st, *musical keys* are like templates that give us a collection of chords & notes that will sound good together. Once we know what key we're in, set of possible chords becomes much more manageable. Next chap will cover main ideas behind keys & scales.
  - 2nd, there are standard *chord progressions* that are used consistently in different genres of music. A chord progression is a sequence of chords that set up a compositional structure for a piece of music. In Chap. 6, show how to generate progression of chords from common templates.
  - 3rd answer is simply hard-won experience. As develop your musical ear, will become more & more familiar with chord types & progressions & how they're used in different genres. This experience will help you begin to innovate & improvise.

  Common interval names: [Table: Semitones: Name]

  Note 3. *Function names have to start with a letter (lowercase or uppercase). Names can include letters, numbers, & underscore _ character. Unicode characters are also allowed.*

- Interlude 4: Playing Chords. In this interlude we're going to explore a few options for playing chords using TunePad & Python. When composing harmony of your song, have more to consider than just what chords to choose. Also have to consider how to play these chords. Subtle variation in timbre, harmonics, & timing can make a huge difference in sound that you ultimately produce. In Chap. 4, saw how to play chords using a list & a single `playNote` statement like this:

```
playNote([48, 53, 55], beats = 4.0)
```

  This is your most basic & most mechanical sounding option. Here are a few other ideas & techniques to experiment with. Can follow along online with this TunePad project https://tunepad.com/interlude/play-chords.

  - Option 1: Block chords. With block chords you play every note in a chord at exact same time & for exact same duration. This approach is simple & can add a strong rhythmic feel to your music. But in some situations using block chords can sound harsh & overly mechanical. Here's a simple function that takes a chord (a list of numbers) & plays each note at same time for an equal duration:

```
def block(chord, beats):
    playNote(chord, beats)
```

  - Option 2: Rolled chords. Sometimes when humans play chords, they introduce subtle variations in timing between note onsets. This variation can be intentional & exaggerated or simply a natural result of playing or strumming chords by hand. This style is called a *rolled* chord. Might roll a chord if want to bring out a change in harmony or if you want to emulate a strummed instrument. Artists like Dr. Dre have used rolled chords on piano to create iconic sounds. Can also combine rolled chords & block chords. If your chord progression is changing chords, can draw attention to this by rolling chord that changes. This function rolls a chord by adding a short, fixed delay between each note onset.

```
def rolled(chord, duration):
    delay = 0.1 # how far to space out note start times
    offset = 0 # accumulated delay
    for note in chord:
        playNote(note, beats = delay, sustain = duration - offset)
        offset += delay # keep track of accumulated delay
    fastForward(duration - offset)
```

Function uses a couple of "bookkeeping" variables called `delay, offset`. `delay` variable just says how long to pause before each successive note in chord is played. `offset` variable keeps track of total amount of delay that we've introduced in for loop. Use `offset` variable in 2 places. 1st, on line 6, adjust `sustain` parameters so that all of notes in chord are released at exact same time (sustain parameter lets not ring out longer than what gets passed in beats parameter). 2nd, on line 9, adjust playhead position after loop finishes. This makes it so that calling rolled function advances playhead forward by *exact amount* specified in `duration` parameter. 1 quick note: line 7 uses plus-equal operator + =. This is a short hand way of saying `offset = offset + delay`.

- Option 3: Random rolled chords. Can take this technique a step further by *randomly* varying note onset times. To do this, use 1 of Python's utilities from random module: `uniform` function. This function takes an input of 2 numbers & generates a random decimal number between those 2 numbers. Will use this to generate an offset between each successive note in chord. As with previous example, use `sustain` parameter of `playNote` to hold out each note for remaining duration of original inputted beats, subtracting total cumulative amount of offset each time.

```
from random import uniform
def rolled(chord, duration):
    max_delay = 0.15
    offset = 0
    for note in chord:
        next_delay = uniform(0, max_delay)
        playNote(note, beats = next_delay, sustain = duration - offset)
        offset += next_delay
    fastForward(duration - offset)
```

This code is a little more complicated than previous example, talk through it 1 line at a time. On 1st line, importing `uniform` function. On line 2, defining our rolled function with 2 parameters: a list of notes in a chord & number of total beats. On line 3, defining a constant value that defines maximum value that our offset between 2 notes can take. Higher values will create a more spaced-out sound, & lower values will create more closed, tighter-sounding chords. On line 4, initialize a variable to track total offset at each step, which starts at 0. Starting on line 5, iterate through each note of chord. At each step, calculate offset to next note & then call `playNote`. Finally, on line 9, move playhead remaining number of beats.

If this code seems confusing, that's okay. Dive more into understanding this kind of code in later chaps. Can treat this function as another tool in your coding toolkit.

- Option 4: Arpeggios. Another way of playing chords: play 1 note at a time. This method is called an *arpeggio*. Order you play notes doesn't matter. Can start at any note & play notes of chord in any order to get sound that's right for your track. In code below, starting with lowest note & working up in increasing order of pitch.

```
def arpeggio(chord, total_beats):
    duration = total_beats / len(chord)
    for note in chord:
        playNote(note, beats = duration)
```

On 1st line, set up our function def, which takes 2 parameters: a list of notes in a chord & total number of beats to play chord. On 2nd line, calculate duration of each individual note by evenly dividing total number fo beats by number of notes in chord. On lines 3 & 4, use a for loop to iterate through list of notes in chord, playing each note for same number of beats.

- Option 5: Patterned arpeggios. This arpeggio function is ok, but try to make sth a little more interesting. Remember, don't have to play each note evenly & can switch up order of chord. Define a function that takes a 7th chord, which has exactly 4 notes, & play it over duration of a measure. Going to use concept of *indexing*, which read more about in next chap. For now, just know that indexing is how we access specific elements of a list. To index a list in Python, use square brackets around position of element want to use. These positions start at 0, so if have a variable `chord` & want to access 1st element of list, root, would type `chord[0]`. With this in mind, define a function. Have 4 beats to work with & 4 notes – that's indices 0–3. A quick example of what's possible, but you can experiment with different note variations & note orders to get different effects.

```
def my_pattern(chord):
    playNote(chord[0], 0.75)
    playNote(chord[1], 0.25)
    playNote(chord[2], 0.5)
    playNote(chord[1], 0.5)
    playNote(chord[3], 0.5)
    playNote(chord[2], 0.5)
    rest(1)
```

In this function, we aren't doing anything fancy to iterate through chord, & don't need to calculate our beats each time. Our beat values for lines 2–8 add up to exactly 4.0 beats. If following along, can try tweaking these to be different values that still up to 4.0 beats. Can also try tweaking indices of notes to change which chord tone plays.

- 5. Scales, keys, & melody. *Scales* are patterns of notes played 1 at a time in ascending or descending order of pitch. Most scales span 1 octave using some subset of 12 possible notes on piano keyboard. When scale completes octave, pattern starts over. *Keys* are similar to scales except ordering of notes doesn't matter, & they contain all of notes in scale regardless of octave that you start on. Keys are like templates that help us select notes & chords that we know will sound good together. Keys give harmonic & melodic structure to music.

  - 5.1. Chromatic scale. Building blocks of scales are half steps & whole steps. Half steps are smallest interval commonly used in music & distance between 2 notes that are next to each other in pitch & on piano keyboard. Whole steps are made up of 2 half steps.

    Most basic scale is chromatic scale. In this scale, every note is exactly 1 half step up from previous note. This scale can start on any note & spans an octave in 12 notes. Starting with a C on piano keyboard, would have following notes: C C# D D# E F F# G G# A A# B. Or, using MIDI (Musical Instrument Digital Interface) note numbers we could also write: 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59.

    Playing a chromatic scale in TunePad is easy using a loop:

    ```
    # loop from 48 up to (but not including) 60
    for note in range(48, 60):
        playNote(note)
    ```

    If wanted to play a chromatic scale starting on a different root note, could just change numbers in `range` function above.

  - 5.2. Major & minor scales. Perhaps most important scales in Western music are major & minor scales. These scales each use 7 out of 12 possible notes in an octave. There are 12 major scales & 12 minor scales, 1 for each possible starting pitch. After 7th note, next note would be 1st note – or *tonic* – an octave up. Scales are named by their tonic & quality in same way that chords are named. A major scale starting on note D would be called *D Major*.

    Major scales are commonly described as cheerful & happy (like major chords). Major scale is made up of following intervals: *whole step, whole step, half step, whole step, whole step, whole step, half step*. Major scale starting on C would have notes shown in Fig. 5.1: Whole step & half step intervals of C major scale.

    In MIDI version, can see whole steps skip up by 2 notes, which half steps skip up by 1 step. [Table: Note names: MIDI numbers: Intervals].

    Minor scales also use 7 notes out of each octave, but in a different order than major scales. This difference in intervals contributes to different emotional connotation of scale. Minor scales are commonly described as sad, melancholy, & distant. A minor scale starting on C would have following notes: [Table: Note names: MIDI numbers: Intervals].

    Major & minor scales are both examples of *modes*. Modes are simply different ways of ordering intervals of a scale.

  - 5.3. Pentatonic scales. Pentatonic scales are a subset of notes of major & minor scales. There are 5 notes in a pentatonic scale. These scales have no half step intervals, which results in less dissonance between notes. Many common melodies are based on pentatonic scales, especially in folk & pop music. Melody of *Amazing Grace* is pentatonic, as is Ed Sheeran's *Shape of You*.

    There are both major & minor pentatonic scales. Major pentatonic is created by omitting 4th & 7th notes of major scale. Minor pentatonic omits 2nd & 6th notse of minor scale. Can experiment with sound of pentatonic scale by playing only black keys of piano keyboard, which forms either an F# major pentatonic scale or a D# minor pentatonic scale Fig. 5.2: C Major Pentatonic Scale & F# Major Pentatonic Scale. F# Major pentatonic scale uses only black keys of keyboard. D# Minor pentatonic scale starts with D# & uses only black keys as well.

  - 5.4. Building scales in TunePad. Building scales in TunePad is similar to building chords. Because scales are just patterns of intervals (spaces between notes), can create short functions to generate scales. Every major scale has an identical pattern of intervals, & same is true for minor scales as well. Only thing that changes is starting note. To generate scales in TunePad, all we need to do is decide what note to start on & then apply pattern to this starting note.

    1 of advantages of thinking about music in terms of computer code: we don't have to memorize endless scales & combinations of notes & chords that make up different keys. Professional musicians train for years to learn how to play different scales without having to think about it so that they can fluidly switch from 1 key to another. This is part of what makes improvisational musicians so impressive. Playing a solo means knowing exactly which notes & chords can be played & how those notes & chords relate to a genre or theme of a piece being performed.

    A quick example of generating a scale with Python code in TunePad:

    ```
    def majorScale(tonic):
        intervals = [ 0, 2, 4, 5, 7, 9, 11 ]
        return [ i + tonic for i in intervals ]
    ```

    Example above uses a new Python concept called a *list comprehension*. A list comprehension is a shorthand way to create a list in Python. Line 2 uses a list comprehension to create a new list consisting each element of intervals list added to tonic value: `[ i + tonic for i in intervals]` This is equivalent to writing:

```
result = [ ]
for i in intervals:
    result.append(i + tonic)
```

2nd version is a little more cumbersome to write than 1st version using list comprehension, although either version is fine to use.

* 5.4.1. Major scale.
  - Intervals: [0, 2, 4, 5, 7, 9, 11]
  - Notation: C major, CMaj, CM, C
  - Python Function with List Comprehension:

```
def majorScale(tonic):
    intervals = [ 0, 2, 4, 5, 7, 9, 11 ]
    return [ i + tonic for i in intervals ]
```

An alternative Python function with a loop instead of a list comprehension:

```
def majorScale(tonic):
    intervals = [ 0, 2, 4, 5, 7, 9, 11 ]
    scale = [ ]
    for i in intervals:
        scale.append(i + tonic)
    return scale
```

A 3rd variation with no loop & no list comprehension:

```
def majorScale(tonic):
    return [ tonic, tonic + 2, tonic + 4, tonic + 5, tonic + 7, tonic + 9, tonic + 11 ]
```

* 5.4.2. Minor scale.
  - Intervals: [0, 2, 3, 5, 7, 8, 10]
  - Notation: C minor, Cmin, Cm
  - Python Function

```
def minorScale(tonic):
    intervals = [ 0, 2, 3, 5, 7, 8, 10 ]
    return [ i + tonic for i in intervals ]
```

* 5.4.3. Major pentatonic scale.
  - Intervals: [0, 2, 4, 7, 9]
  - Python Function:

```
def majorPentScale(tonic):
    intervals = [ 0, 2, 4, 7, 9 ]
    return [ i + tonic for i in intervals ]
```

* 5.4.4. Minor pentatonic scale.
  - Intervals: [0, 3, 5, 7, 10]
  - Python Function:

```
def minorPentScale(tonic):
    intervals = [ 0, 3, 5, 7, 10 ]
    return [ i + tonic for i in intervals ]
```

* 5.4.5. Chromatic scale.
  - Intervals:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
  - Python Function:

```
def minorPentScale(tonic):
    intervals = [ 0, 3, 5, 7, 10 ]
    return [ i + tonic for i in intervals ]
```

Try these functions at https://tunepad.com/examples/build-scales.

- 5.5. Playing scales in TunePad. Now have seen how to build a scale, can use functions from prev sect to play music. Unlike with chords, notes of a scale aren't usually played all at once. Most basic way to play a scale: play 1 note at a time, in ascending

order. Somehow we have to access each element of list individually & pass that to `playNote` Fig. 5.3: A representation of a list with values & indices.

Each element can be accessed using its position in list – called an *index*. In coding, 1st element of a list is at index 0; 2nd element of list is at index 1, 3rd at index 2; & so on. In Python can also access last element of a list at index −1. Accessing individual elements of a list is referred to as *indexing*. In code, do this by using square brackets & index number. Can also use this technique to change value of individual elements in a list.

```
notes = [ 60, 62, 64 ]
notes[2] = 66 # replace the value 64 with 66
playNote( notes[0] )
playNote( notes[1] )
playNote( notes[2] )
```

In line 1, define a new list called *notes* with 3 values. In line 2, replace value at index 2 with new value of 66. In lines 3–5, play each note of list, 1 at a time. *1 of most confusing parts about computer programming for beginners: lists start at index 0 & end at an index 1 less than length of list.* However, with a little practice this become less & less confusing.

Note 4. *If try to index into a list with an index that doesn't exist, Python will stop running & complain with an* `IndexError`. *Because indices start at 0, valid indices are 0 all way up to length of list minus 1.*

Another way to iterate through a list is by using a *for loop*. Previously, when have seen loops, have used them to do exact same operation multiple times in a row. Recall syntax of a for loop:

```
for var in range(start, stop):
```

Can replace `range(start, stop)` part of a for loop with a list instead. This will execute body of loop once for every element in list. There's a special variable here called "loop variable" that gets set to value of each consecutive element in list every time the loop repeats. In code above, `var` is loop variable, but can use any valid Python variable name. E.g., a loop that plays all notes of a major scale starting on note 60.

```
for note in majorScale(60):
    playNote(note)
```

In this example, `note` is our loop variable. For every iteration of loop it gets set to next note in scale. Give this a try at https://tunepad.com/examples/play-scales.

- 5.6. Other kinds of scales. There are many other types of scales, but variants of major & minor scale are most common in popular music. Other scales that we don't cover here include set of church modes, whole tone scales, diminished scales, & modes of limited transposition.

  – Các loại thang âm khác. Có nhiều loại thang âm khác, nhưng các biến thể của thang âm trưởng & thứ là phổ biến nhất trong âm nhạc đại chúng. Các thang âm khác mà chúng tôi không đề cập ở đây bao gồm bộ các chế độ nhà thờ, thang âm toàn cung, thang âm giảm, & chế độ chuyển cung hạn chế.

  Above have discussed scales common to Western music, but concept of collections of notes is cross-cultural. Arabic maqam is system of melodic modes in traditional Arabic music used in both compositions & improvisations. In Indian classical music Rage are collections of melodic modes & motifs, each connoting a distinct personality or emotion. Gamelan music in Indonesia is organized by Pathet, which is a system of hierarchies of notes in which different notes have prominence. Composers from West have often borrowed – or in some cases, stolen – these scales for their own music. This raises many issues of appropriation & exploitation within music industry. Music industry has a long history of marginalizing groups while also profiting off of cultural traditions without properly compensating or recognizing musical provenance.

  – Ở trên đã thảo luận về các thang âm phổ biến trong âm nhạc phương Tây, nhưng khái niệm về tập hợp các nốt nhạc là liên văn hóa. Maqam tiếng Ả Rập là hệ thống các chế độ giai điệu trong âm nhạc Ả Rập truyền thống được sử dụng trong cả sáng tác & ngẫu hứng. Trong âm nhạc cổ điển Ấn Độ, Rage là tập hợp các chế độ giai điệu & họa tiết, mỗi chế độ biểu thị 1 tính cách hoặc cảm xúc riêng biệt. Âm nhạc Gamelan ở Indonesia được tổ chức theo Pathet, đây là 1 hệ thống phân cấp các nốt nhạc trong đó các nốt nhạc khác nhau có sự nổi bật. Các nhà soạn nhạc phương Tây thường mượn – hoặc trong 1 số trường hợp, đánh cắp – các thang âm này cho âm nhạc của riêng họ. Điều này làm nảy sinh nhiều vấn đề về chiếm đoạt & khai thác trong ngành công nghiệp âm nhạc. Ngành công nghiệp âm nhạc có lịch sử lâu dài về việc gạt ra ngoài lề các nhóm trong khi cũng kiếm lợi từ các truyền thống văn hóa mà không đền bù hoặc công nhận nguồn gốc âm nhạc 1 cách thỏa đáng.

- 5.7. Keys. When writing music, there are seemingly endless notes to choose from. Keys are 1 way to narrow down question of what note to choose. Keys are underlying organizational framework of most music & encode both melodic & harmonic structures & rules. Knowing these rules (& how to break them) helps us to write music that listeners can easily comprehend & appreciate.

  Concept of keys is closely related to that of scales. Keys are composed of all of notes in all of octaves that make up scale with same name. E.g., notes in key C major are same as notes in C major scale across all octaves. But, while scales are

usually played in increasing or decreasing order of pitch, ordering of notes in a key doesn't matter. Notes that are part of a given key are called *diatonic*, & remaining notes that are not part of that key are called *chromatic*.

Hundreds of years ago, different keys used to be associated with different emotions, so composers would choose specific keys that reinforce mood of their composition. This is because intervals in each key were slightly different due to system of tuning; different keys were actually aurally distinct from 1 another. In modern times, each key is made up of exact same intervals.

- 5.8. Circle of 5ths. Keys are organized according to *Circle of 5ths*. Circle of 5ths is essentially a pattern of intervals. Moving clockwise around circle is moving tonic note up by a 5th from previous key. This adds 1 raised – or sharp – note as 7th note of scale. Alternatively, moving counterclockwise raises tonic by a 4th & is often referred to as Circle of 4ths. This adds 1 flat note as 4th note of scale Fig. 5.4: Circle of 5ths arranges musical keys.

  Major & minor keys that share all of same notes are considered relative keys. For a minor scale, relative major scale starts on 3rd note; for a major scale, relative minor starts on 6th note. Relative minor key of C major is A minor, & relative major of A minor is C major.

  Keys that are adjacent on Circle of 4ths or 5ths – e.g., D major & G major – share nearly all of same notes & are considered *closely related*. Relative major or minor key for a given key is also considered closely related. Generally, when a song changes keys – also known as *modulating* – it goes to 1 of closely related keys. Because closely related keys share most of same notes, modulating to 1 of these keys is less jarring to listener.

- 5.9. Melody. *Melody* is central component of much of music we listen to. It's part of a song that gets stuck in your head. Much of what goes into great melody writing is intuition & practice, but knowing a bit of theory can help you get started. Melodies have 2 essential parameters: pitch & rhythm. These elements are of equal importance, but in this sect, mostly be looking at pitch.

  When it comes to writing melodies, understanding that you are working within confines of keys, scales, & a harmonic chord progression is a great place to start. Can use our song's harmony to provide a structural scaffold. Often, melodies place chord tones from harmony on strong beats of measure (beats 1 & 3). These tones are consonant with harmony, i.e. they sound pleasing. Simplest melody might stick solely to these chord tones. In example below, only play chord tones of C major & D minor.

```
# over C major
playNote(55, 0.75)
playNote(55, 0.25)
playNote(52, 1)
playNote(48, 0.5)
rest(1.5)

# over D minor
playNote(57, 0.75)
playNote(57, 0.25)
playNote(53, 1)
playNote(50, 0.5)
```

  Follow along with these examples at https://tunepad.com/examples/simple-melody.

  Dissonance is also a powerful tool in melody writing. This can add interest & variation & sometimes have an intense emotional impact on listeners. A melody with no dissonance, that only plays chord tones, becomes boring. 1 way to utilize dissonance: add notes in between our chord tones to fill in our melody. Can choose notes that correspond with scale based on our song's key. In example below, now filling in space between last 2 notes of each measure:

```
# over C major
playNote(55, 0.75)
playNote(55, 0.25)
playNote(52, 0.5)
playNote(50, 0.5) # passing tone
playNote(48, 0.5)
rest(1.5)

# over D minor
playNote(57, 0.75)
playNote(57, 0.25)
playNote(53, 0.5)
playNote(52, 0.5) # passing tone
playNote(50, 0.5)
```

  Sth to consider when writing a melody is *contour*. Contour describes shape melody takes: natural rises & falls in pitch. A melody can either move stepwise to adjacent notes or leap to more distant notes. This motion can either decrease or increase

in pitch. I.e., have 4 types of motion a melody might take, each with different connotations. E.g., might hear a melody that opens with a large leap as more emotional. Can expect most melodies to be within range of about an octave to an octave & a half. In example below, combine idea of leaps to chord tones & passing tones:

```
# over C major
playNote(55, 1)
playNote(64, 1.5) # large leap
playNote(60, 1.5)

# over D minor
playNote(57, 1)
playNote(65, 1.5) # large leap
playNote(67, 1) # passing tone
playNote(69, 0.5)
```

If consider contour & pitch content as a vertical phenomenon, can think of melodic form as a horizontal structure. Can break melodies into parts called *phrases*. If melodies are paragraphs, then phrases are like musical sentences. They are complete thoughts that are punctuated & combined to form more complete & cohesive ideas. Phrases are often 2, 4, or 8 bars in duration. These phrases are combined to form larger structures, which become overall song form. Explore this more in Chap. 9.

Principles of repetition & variation work in opposition to 1 another. In writing melodies, there generally needs to be enough repetition so that a listener has sth to latch onto. But with too much repetition, a melody becomes boring. A catchy melody is result of striking a balance between these 2 forces.

1 way to build intuition about melody writing: analyze melodies from artists you like & want to emulate. Critical listening skills that you develop from analyzing existing melodies is directly applicable to writing your own melodies. Experimentation & improvisation are other great ways to build up this intuition. Can try tapping out rhythms to serve basis of a melody, or play around on a piano or another instrument. Try playing around with our automatic melody generator at https://tunepad.com/examples/melody-gen.

- Interlude 5: Lean On Me. BILL WITHERS. Practice using chords by recreating a small part of piano harmony from song *Lean on Me* by BILL WITHERS (1972), Columbia Records. A simplified version of chord structure that you can try entering into a TunePad project.

```
# Chord Variables
Cmaj = [ 48, 52, 55 ]
Dmin = [ 50, 53, 57 ]
Emin = [ 52, 55, 59 ]
Fmaj = [ 53, 57, 60 ]
Bdim = [ 47, 50, 53 ]
```

[Table: Chord: Beats: Python]. Can see full code here: https://tunepad.com/interlude/chord-progressions. 1 thing to notice is how chord progression mirrors emotion of song as a whole. WITHERS mixes upbeat ("I'll be your friend. I'll help you carry on") with harsh reality of life ("We all have pain. We all have sorrow"). Harmony starts on a major chord (C major) but then passes through a succession of minor chords (D minor, E minor) before eventually landing on more encouraging major chords for prolonged notes (F major). It's as if harmony is also saying: we're going to go through some hard times, but it'll work out in end.

Version above is slightly modified from original in that we're using simplified chords & a diminished B chord at end that slides into a C major. As listen to it, notice how B diminished feels unstable as if it needs to resolve into C major to bring harmony full circle to signal a transition in song.

More elegant code. 1 temping way to code this up would be to just type all of `playNote` functions, 1 after another. This works, but it's not necessarily most elegant way to express music. When you're coding, there's always $>$ 1 way to solve a problem, so it's good to get into habit of asking if there are other, easier ways to accomplish things. E.g., what if wanted to change velocity of all of chords? We'd have to edit 1 line at a time or use find & replace. As an alternative, what if we put all of chords into a list & then iterated through that list with a for loop?

```
chords = [ Cmaj, Cmaj, Dmin, Emin, Fmaj, Fmaj, Emin, Dmin, Cmaj ]
for chord in chords:
    playNote(chord)
```

This would be an improvement. If nothing else, would have reduced number of lines needed to play harmony. Obvious problem: it won't work because notes are different lengths. Some are long (4 beats) & others are short (1 beat). But this code plays all chords with equal duration.

If there were an easy way to iterate through 2 lists at same time, could make 1 list with chords & another with durations. Python includes exactly this kind of feature with sth called `zip` function. Think of it like a zipper that merges 2 Python lists together instead of 2 pieces of fabric. It walks through lists, element by element, & merges them together into pairs of values. Result looks sth like this:

```
chords = [ Cmaj, Cmaj, Dmin, Emin, Fmaj, Fmaj, Emin, Dmin, Cmaj ]
durations = [ 4, 1, 1, 1, 4, 1, 1, 1, 4 ]
for chord, duration in zip(chords, durations):
    playNote(chord, beats = duration)
```

Complete example or you can try it online: https://tunepad.com/interlude/lean-on-me.

```
CM = [ 48, 52, 55, 55 + 12 ]
Dm = [ 50, 53, 57, 57 + 12 ]
Em = [ 52, 55, 59, 59 + 12 ]
FM = [ 53, 57, 60, 60 + 12 ]
Bd = [ 47, 50, 53, 53 + 12 ]

chords = [ CM, CM, Dm, Em, FM, FM, Em, Dm, CM, CM, Dm, Em ]
durations = [ 4, 1, 1, 1, 4, 1, 1, 1, 4, 1, 1, 1, 3, 4 ]

for chord, duration in zip(chords + [ Em, Dm ], durations):
    playNote(chord, beats = duration)

for chord, duration in zip(chords + [ Bd, CM ], durations):
    playNote(chord, beats = duration)
```

- 6. Diatonic chords & chord progressions. Now that have some familiarity with chords, question is how to use them. How can we reduce hundreds of chords & thousands of combinations of chords down to a manageable set of options? How can we explore creative musical space that chords provide without feeling overwhelmed?

  – Hợp âm diatonic & tiến trình hợp âm. Bây giờ đã quen với hợp âm, câu hỏi đặt ra là làm thế nào để sử dụng chúng. Làm thế nào chúng ta có thể giảm hàng trăm hợp âm & hàng nghìn tổ hợp hợp âm xuống 1 tập hợp các tùy chọn có thể quản lý được? Làm thế nào chúng ta có thể khám phá không gian âm nhạc sáng tạo mà hợp âm cung cấp mà không cảm thấy choáng ngợp?

  1 answer to these questions: use keys to select subsets of chords that we know will sound good together. From there we can follow guidelines for arranging chords into sequential patterns called *progressions* that will support various harmonic elements that come together in a piece of music.

  This chap introduces traditional *Roman numeral* system for referring to chords that fit with a particular key along with methods for choosing chord progressions. Along way we'll code functions for creating chord progressions in any key using concept of Python *dictionaries*.

  - 6.1. Diatonic chords. A *diatonic* chord is any chord that can be played using only 7 notes of current key. E.g., if working in key of C major, diatonic chords consist of all of chords you can play with only white keys on piano keyboard: C D E F G A B. Main diatonic chords we can make with just these 7 notes Fig. 6.1: 7 diatonic chords of C major. Hear these chords at https://tunepad.com/examples/diatonic-chords.

    If have a piano keyboard handy, try playing with these 7 chords to get a feel for how they sound. What emotions do you feel as chords ring out? What patterns of chords sound good together? No matter what key we're in, there will always be 7 diatonic chords, 1 for each note in scale. To build a diatonic chord, just pick any note from scale as root of chord. Then go up 2 notes for "3rd" of chord, & up 2 notes again for "5th" of chord. Can keep moving up by 2 notes of scale to get "7th" & "9th" & so on.

    Can refer to these chords by their scale degree (1st, 2nd, 3rd, 4th, 5th, 6th, & 7th). A chord built on 5th note of a scale would be "5" chord for that key. If we're in any *major key*, 1st, 4th, & 5th diatonic chords will have a major quality (regardless of starting note of key). Further, 2nd, 3rd, & 6th chords will always have a minor quality, & 7th chord is diminished. E.g., in C major, would have following diatonic chords: CMaj Dmin Emin FMaj GMaj Amin Bdim. This pattern of chord qualities is same for any major key because all of major keys have same pattern of note intervals. Same idea is true for minor keys. Because all minor keys have same pattern of intervals, quality of chords stays consistent. 1st, 4th, & 5th diatonic chords are minor quality. 3rd, 6th, & 7th chords are major quality. 2nd chord is diminished. In C minor, e.g., would have following diatonic chords: Cmin Ddim E♭Maj Fmin Gmin A♭Maj B♭Maj. In popular music, chord progressions are made up almost entirely of diatonic chords. Recall from prev chap: melody of a song is also built on this harmonic scaffold. Melodies often use primarily notes from underlying chord progression, because these notes are more consonant & pleasing. Notes from outside harmony are typically used in passing as decoration or as a neighboring tone.

- 6.2. Roman numerals. Each of 12 major keys & 12 minor keys have 7 diatonic chords, giving us an overwhelming total of 168 diatonic chords. To reduce this complexity, musicians, producers, & composers use a system of *Roman numerals* to refer to different diatonic chords by their scale degree rather than their specific name. If chord has a major quality, it gets an uppercase Roman numeral. If it has a minor quality, it gets a lowercase Roman numeral. Each key also has 1 diminished chord, which is both lowercase & has an accompanied ° symbol.

  Note 5. *Roman numerals are a system of numbering which originated in ancient Rome. In this system, numbers are composed of combinations of letters. In music, only numerals corresponding to numbers one through 7 are used: I, II, III, IV, V, VI, & VII.*

  That gives us following Roman numerals ∀ of diatonic chords of any key. [Table: Scale degree (Major keys, Minor keys): 1st: 2nd: 3rd: 4th: 5th: 6th: 7th]. With this system there are just these 7 symbols to focus on instead of 168. Roman numerals take some getting used to, but they give us a language for thinking about chord progressions without having tp refer to name of each specific chord. In coding or mathematics, this kind of generalization is called an *abstraction*. Abstraction means removing specific details of individual situations & focusing instead on bigger picture patterns. In coding, use language constructs like variables, functions, & parameters to create abstractions that reduce complexity of our code & problems we're trying to solve.

- 6.3. Tendency tones & harmonic functions. Now know how to refer to diatonic chords by their names, how do order them into pleasing chord progressions? Ordering of chords within a progression isn't random. Notes that make up chords have different tendencies relative to 1 another – i.e. listeners hears them as wanting to resolve in expected ways when played with other notes in a chord progression. Strongest of these tendencies is pull of 7th note of a scale back to root note of scale (tonic). In most cases, hear this note as wanting to resolve upward by a half step back to tonic. If this note does not resolve, often hear progression as incomplete. 2nd & 5th notes of a scale also have a strong pull back to tonic.

  Tendencies of individual notes give each chord a characteristic or *function*. Can think of a chord's function as its desire – how it relates to prev chords & how it *wants* to move music forward. Chords are described as having 3 primary functions: *tonic, predominant, & dominant*. Most chord progressions typically progress from tonic to predominant to dominant back to tonic. [Table: Tonic chords: Predominant chords: Dominant chords].

  Chords that have same function also share many of same notes. E.g., iii (3) chords & vi (6) chords share 2 of same notes with tonic chords (I), so they're grouped together. Predominant ii & IV chords also share 2 notes in common, as do dominant V & vii° chords Fig. 6.2: Dominant V & vii° chords share 2 notes in common. This is easy to see when you line piano diagrams up vertically.

  Chord functions might best be described as rough guidelines that help inform our decisions about composition. There are also many variations to basic chords. Chords can be inverted (i.e., root is no longer lowest note), extended with additional notes to add color, or combined with "chromatic" chords that include notes from outside of main key. Secondary dominant chords are dominant chords borrowed from other related keys. Knowing which chord to use in any given context comes down to musical experience, taste, & conventions of different genres.

- 6.4. Chord progressions. Many common chord progressions follow scheme of *tonic* → *predominant* → *dominant*. Tonic brings stability & grounding. Predominant is a departure from this stability that builds tension. Predominant pulls toward dominant, which eventually resolves back to tonic. After a chord progression finishes, it starts over. Different genres of music have different harmonic rules & standard chord progressions, but here are flow charts that help visualize common chord progression patterns Fig. 6.3: Flowcharts for generating chord progressions in major & minor keys.

  An example of how to use these charts. If start with tonic I chord on top, might move down to dominant V chord, & then slide up to tonic prolongation vi chord – a subtle tease with resolution. Could then go to predominant IV chord before progression resolves back to tonic I chord & repeats. This progression would be I → V → vi → IV (1, 5, 6, 4), which is an extremely common pattern in popular music Fig. 6.4: Example of using flowchart to generate a chord progression.

  Contemporary pop music uses many of same progressions as early rock & Roll & Blues music – much of early rock music came out of Blues. As a result, many early rock songs are built on Blues progressions, most notably I-IV-V. 1 of most ubiquitous progressions – especially in early rock – is "doo wop" progression I-vi-IV-V. Same chords can be reordered to form our I-V-vi-IV example. [Table: Common major progressions: Common minor progressions]

  Hip-hop songs center more around rhythm & vocals & tend to use shorter progressions, often in a minor key with only 1 or 2 chords e.g. i-V, i-VI, & i-ii°. Hip-hop developed alongside advances in recording technology that allowed early artists to remix samples from other songs, &, as a result, genre also borrows progressions from pop & rock music.

  When writing chord progressions, 1 tactic: borrow from existing songs to help you develop your own ear & begin to think critically about harmony. Can also experiment on your own. Use harmonic conventions to narrow down some of options, but also try breaking rules as you become more confident.

- 6.5. Chord inversions. An *inverted* chord is just like an ordinary chord except that root note is no longer lowest pitch. Take C major as an example. When root note C is also lowest note of chord, say chord is in root position Fig. 6.5: C major chord in root position, 1st inversion & 2nd inversion.

  When 3rd of chord is lowest note, chord is in its 1st inversion. In case of C major, i.e., E is now lowest note. When 5th of chord is lowest, it's 2nd inversion, & so on. Each inversion has exactly same notes as root chord, but ordering of notes by their pitch is different.

- 6.6. Voice leading. *Voice leading* deals with relationship between notes in consecutive chords in a progression. Principle behind voice leading: treat each note of a chord as an individual melodic voice. Imagine 3 human vocalists each singing 1

individual note of a chord. Because considering each voice independently, idea: minimize leaps 1 person's voice has to make between chords so that progression is smoother & easier to sing. By considering different possible inversions of each chord we can create more of a dovetailing effect with subtle shifts between successive chords. Not only will this improve sound of your progressions, but it will also improve potential playability of music on instruments like guitar, piano, or vocal harmony. 2 figures show same progression with & without voice leading Fig. 6.6: Chord progression I-V-vi-IV without voice leading & with voice leading. Chords V, vi, & IV are inverted to reduce pitch range & to minimize movement of individual voices "singing" notes of chords. Hear these examples at https://tunepad.com/examples/voice-leading.

- 6.7. Python dictionaries. In Python, *dictionaries* or *maps* are unordered sets of data consisting of values referenced by *keys*. These keys aren't same as musical keys. They're more like kind of keys that open locked doors. Each different key open its own door.

  Dictionaries are extremely useful in programming because they provide an easy way to store multiple data elements by name. E.g., if wanted to store information for a music streaming service, might need to save song name, artist, release date, genre, record label, song length, & album artwork. A dictionary gives you an easy method for storing all of these elements in a single data object. (like *.bib files)

```
track_info = {
    "artist" : "Herbie Hancock",
    "album" : "Head Hunters",
    "label" : "Columbia Records",
    "genre" : "Jazz-Funk",
    "year" : 1973,
    "track" : "Chameleon",
    "length" : 15.75 }
```

  Dictionaries are defined using curly braces with keys & values separated by a colon. Different entries are separated by commas. After defining a dictionary, can change existing values or add new values using associated keys. Similar to way we access values in a list with an index, use square brackets & a key to access elements in a dictionary.

```
track_info["artwork"] = "https://images.ssl-images-amz.com/
images/81KRhL.jpg"
```

  In this line, because key "artwork" hasn't been used in dictionary yet, it creates a new key-value pair. If "artwork" had been added already, it would change existing value. 1 thing to notice: values in a dictionary can be any type including strings, numbers, lists, or even other dictionaries. Dictionary keys can also be strings or numerical values, but they must be unique for each value stored.

- 6.8. Programming with diatonic chords. With Python code, there are many different ways to determine diatonic chords for a given key. Here are `majorChord, minorChord, diminishedChord` functions from Chap. 4 again.

```
def majorChord(root):
    return [root, root + 4, root + 7]

def minorChord(root):
    return [root, root + 3, root + 7]

def dimChord(root):
    return [root, root + 3, root + 6]
```

  Can use these functions to define variables for each diatonic chord in C major:

```
I = majorChord(48)
ii = minorChord(50)
iii = minorChord(52)
IV = majorChord(53)
V = majorChord(55)
vi = minorChord(57)
vii0 = diminishedChord(59)
```

  This code is clear & readable, but not as reusable as it could be. What if want to play in a different key? Or in a different octave? Would have to change *each* line of code. As an alternative, could write a function that takes tonic as an input & returns a dictionary that maps Roman numerals to individual diatonic chords.

```
def buildChords(tonic):
    numerals_lookup = { "I" : majorChord(tonic),
                        "ii" : minorChord(tonic+2),
```

```
                    "iii" : minorChord(tonic+4),
                    "IV" : majorChord(tonic+5),
                    "V" : majorChord(tonic+7),
                    "vi" : minorChord(tonic+9),
                    "vii0" : diminishedChord(tonic+11)}
        return numerals_lookup
```

Method above works for major keys, but what if wanted it to work with minor keys as well? Can add another parameter & an `if-else` statement to handle this as well.

```
def buildChords(tonic, mode):
    if mode == "major":
        numerals_lookup = {"I" : majorChord(tonic),
                           "ii" : minorChord(tonic+2),
                           "iii" : minorChord(tonic+4),
                           "IV" : majorChord(tonic+5),
                           "V" : majorChord(tonic+7),
                           "vi" : minorChord(tonic+9),
                           "vii0" : diminishedChord(tonic+11)}
    else:
        numerals_lookup = {"i" : minorChord(tonic),
                           "ii0" : diminishedChord(tonic+2),
                           "III" : majorChord(tonic+3),
                           "iv" : minorChord(tonic+5),
                           "v" : minorChord(tonic+7),
                            "VI" : majorChord(tonic+8),
                           "VII" : majorChord(tonic+10)}
    return numerals_lookup
```

These are far from only solution for creating diatonic chords for different keys. In general, there are almost an endless number of ways to solve complex problems in programming. Figuring out which approach is best for a given circumstance takes practice & experience, but your goal is usually to write code that is as simple & easy to understand as possible. Try this code at https://tunepad.com/examples/chord-dictionary.

- Interlude 6: Random Chord Progressions. A short Python example that generates & then plays random chord progressions using charts in Fig. 6.3. Can start with a table that maps each chord to a simplified set of possible transition chords. Table below on left uses Roman numerals, & table on right uses Arabic numbers to show same thing. Note these tables don't include all of possibilities from flow charts above, but most of possible transitions are included. Now can turn this transition table into a computer algorithm using Python.

  - **Step 1: Random chord algorithm.** Create a new piano cell in a TunePad project & add this code.

```
from random import choice # import the choice function

progression = [ 1 ] # create a list with just one chord
chord = choice([3, 4, 5, 6]) # choose a random next chord

while chord != 1: # repeat while chord is not equal to 1
    progression.append(chord) # add the next chord to the list
    if chord == 2: # if the current chord is 2
        chord = choice([1, 5]) # then choose a random next chord
    elif chord == 3: # else if the current chord is 3
        chord = 4 # ...
    elif chord == 4:
        chord = choice([1, 2, 5])
    elif chord == 5:
        chord = choice([1, 6])
    else: # the chord is 6
        chord = choice([ 2, 3, 4 ])
print(progression)
```

Break it down line by line. Line 1 imports a function called *choice* from Python's `random` module. `choice` function selects 1 element from a list at random. Can think of it as picking a random card from a deck. Line 3 creates a variable called `progression` that consists of a list with only 1 element in it. This list will hold our finished chord progression, & start it

with tonic chord 1. Line 4 picks next chord at random. Use our transition table to select from 3–6 as possible next chords in sequence. Save random choice in a variable called `chord`. Line 6 is a new kind of Python loop called a `while loop`. This loop repeats indefinitely until a certain condition is met. In our case, going to repeat loop until our `chord` variable = 1. Line 7 is part of while loop. It adds our new `chord` to end of `progression` list using `append` function. 1st time through loop, `progression` list will have 2 elements, 1 & whatever random chord was selected on line 4. Each additional time through loop, line 7 will add another chord to list. Line 8 asks *if* our random chord = 2. If so, it selects a random next chord based on values in our transition table (on line 9). Line 10 only gets used if line 8 is False. This line says: otherwise, if value of `chord` is 3, then set next chord to 4. Lines 12, 14, & 16 handle next set of options for value of `chord`, following transition table.

Once loop completes, line 18 print out result. A sample output might be `[1, 5, 6, 3, 4, 2]`, but since this uses random selection, output is likely to be different each time code runs.

- Step 2: Play chords. So how do our random chord progressions sound? Can add a few more lines of code at end to play our progression in TunePad. Start by defining our diatonic chords in a dictionary. Instead of using Roman numerals as keys, we're going to use chord numbers.

```
tonic = 48
chords = {
    1 : majorChord(tonic),
    2 : minorChord(tonic + 2),
    3 : minorChord(tonic + 4),
    4 : majorChord(tonic + 5),
    5 : majorChord(tonic + 7),
    6 : minorChord(tonic + 9),
    7 : diminishedChord(tonic + 11) }
```

Next can iterate over our `progression` list, playing each chord in turn.

```
for chord in progression:
    playNote(chords[chord], beats = 2)
```

Can try this code on TunePad https://tunepad.com/interlude/random-chords.

- 7. Frequency, fourier, & filters. Chap. 3 introduced idea that different instruments & voices naturally fall into different ranges of frequency spectrum, from low sounds like a bass to high sounds like hi-hats. In this chap, further explore frequency spectrum, this time with an emphasis on techniques for mixing multiple layers of a musical composition into a cohesive whole. Show how sound can be decomposed into its component frequencies & how can use filters & other tools to shape sonic parameters e.g. frequency, loudness, & stereo balance. Also show how to apply these standard filters & effects in TunePad using Python code.

  - 7.1. Timbre. (Âm sắc) All sound is made up of waves of air pressure that travel outward from a sound's source until they eventually reach our inner ears. Sound waves that vibrate regularly, or periodically, are special kinds of audio signals which human brain interprets as musical pitch. Rate of vibration (or frequency) determines how high or low pitch sounds. 1 surprising things about musical notes: they are almost never composed of just 1 frequency of sound. In fact, what people hear as 1 musical note is actually a whole range of frequencies stacked on top of 1 another. E.g., Fig. 7.1: Sound energy generated by a flute playing a single note. Sound contains a series of spikes at regular "harmonic" frequency intervals. shows sound energy generated by a flute playing a single note. Figure shows energy level at different frequencies across whole range of human hearing (from about 20 Hz–20000 Hz). Frequency level is shown on horizontal axis & energy level is shown on vertical axis. Spikes in graph show sounds generated by flute at different frequency levels. So even though we only hear a single note, there are actually a whole range of frequencies present in sound, 1 for each spike.

    Frequency combinations like this allow people to distinguish different kinds of instruments from 1 another. It's how your brain can tell difference between a trombone & cello, even when they're playing exact same note. Many frequencies of a single sound are called its *frequency spectrum*, e.g., Fig. 7.1, & they create what is called *timbre* (pronounced "TAM-ber") – often called *tone color* or *tone quality*. Timbre is like fingerprint of a sound.

    – Các tổ hợp tần số như thế này cho phép mọi người phân biệt các loại nhạc cụ khác nhau. Đó là cách não của bạn có thể phân biệt được sự khác biệt giữa kèn trombone & cello, ngay cả khi chúng chơi cùng 1 nốt nhạc. Nhiều tần số của 1 âm thanh duy nhất được gọi là *phổ tần số* của nó, ví dụ, Hình 7.1, & chúng tạo ra cái gọi là *âm sắc* (phát âm là "TAM-ber") – thường được gọi là *màu sắc âm thanh* hoặc *chất lượng âm thanh*. Âm sắc giống như dấu vân tay của âm thanh.

    * **Timbre.** Unique fingerprint of a sound that results from how we perceive multiple frequencies combining together.
    * **Fundamental frequency.** Lowest (& usually) loudest frequency that we perceive as pitch of a note.
    * **Partials or overtones.** Other frequencies beyond fundamental frequency that are also present when we hear a note.
    * **Harmonic frequency.** Any frequency that is close to an integer multiple of fundamental frequency.
    * **Inharmonic frequency.** Any partial that is not an integer multiple of fundamental frequency.

What we perceive as pitch of a note is usually lowest of frequencies present. This is known as *fundamental frequency*, or often simply *fundamental*. Also usually loudest of frequencies. Remaining frequencies are referred to as *partials* or *overtones*. If frequency of partial is close to an integer multiple of fundamental, when partial is considered to be a *harmonic* of fundamental. Otherwise, partial is considered *inharmonic*. Most pitched or melodic instruments – e.g. saxophones, flutes, & guitars – have very harmonic spectrums Fig. 7.1. Non-pitched or percussive instruments often have very inharmonic spectrums, i.e., you don't really perceive pitch of these instruments. You can hear how this sounds here: https://tunepad.com/examples/spectrums.

To help make this more clear, consider a sound consisting of following frequencies: 200 Hz, 400 Hz, & 500 Hz Fig. 7.2: Frequency combinations: fundamentals, partials, harmonic,& inharmonic. Fundamental of sound would be 200 Hz, because it's lowest (& loudest) frequency. 400 Hz frequency would be 1st partial & would be considered a harmonic because it's an integer multiple of fundamental 400 Hz/200 Hz = 2. 500 Hz frequency would be 2nd partial, but it's not a harmonic because 500 Hz/200 Hz = 2.5. Add 100 Hz as new fundamental frequency. In this case, 200 Hz, 400 Hz, & 500 Hz would all be considered harmonics of 100 Hz because they are all simple integer ratios of 100 Hz (2, 4, & 5). Listen to an example https://tunepad.com/examples/timbre.

Almost all sounds consist of a complex combination of frequencies. 1 exception to this rule is sine wave (Fig. 7.2 shows combinations of sine waves). Since waves are made up of just 1 frequency with no other partials, & they are often described as sounding clear or pure because of this. Sine waves are easy to generate using electronics or a computer, but they rarely occur in nature, i.e., they can also sound artificial & harsh.

It turns out: any periodic sound can be described as a combination of a (possibly infinite) number of sine waves forming partial frequencies. Sounds that have very few partials, like a whistle, are often very close to sine waves. Sounds that have many partials, like a saxophone, have much richer & more complex waveforms. 1 way to imagine this: sine waves are like primary colors of paint that you can mix together to form every other color. Fig. 7.3: A square wave (or any other audio signal) can be described as a series of sine waves making up partial frequencies. shows how multiple sine waves at different harmonic frequencies can combine to approximate a more complex signal like a square wave.

- 7.2. Envelopes. There are other complex properties of sound waves that contribute to an instrument's timbre. 1 of most important of these is how volume of a sound evolves & changes over duration of a note. This is called sound's *envelope*. A simplified envelope is commonly described using 4 stages: Attack, Decay, Sustain, & Release or ADSR for short Fig. 7.4: ADSR envelope.

  ADSR envelope has both time components & amplitude (loudness) components. When play a note on piano or another instrument, *attack* is time from when key is 1st pressed to when note reaches its maximum volume. *Decay* is time it takes note to reach a lower secondary volume. *Sustain* is loudness of this 2nd volume. Finally, *release* is how long it takes for note to completely fade out. So, attack, decay, & release are all measures of time, while sustain is a measure of loudness.

  – Phong bì ADSR có cả thành phần thời gian & thành phần biên độ (độ lớn). Khi chơi 1 nốt nhạc trên đàn piano hoặc 1 nhạc cụ khác, *attack* là thời gian từ khi phím được nhấn lần đầu tiên đến khi nốt nhạc đạt đến âm lượng tối đa. *Decay* là thời gian nốt nhạc đạt đến âm lượng thứ cấp thấp hơn. *Sustain* là độ lớn của âm lượng thứ 2 này. Cuối cùng, *release* là thời gian nốt nhạc mất bao lâu để mờ dần hoàn toàn. Vì vậy, attack, decay, & release đều là các phép đo thời gian, trong khi sustain là phép đo độ lớn.

  A sound like a snare drum has a sharp attack & a quick release, while sounds like cymbals or chimes have fast attacks but slower releases that ring out over longer periods of time. Other sounds like violins have both slower attacks & releases. Attack, decay, & release sects of an envelope can also be curved instead of straight lines, which sometimes better approximates sound of real musical instruments. But important to remember: ADSR envelopes are always simplifications of reality. E.g., sustain of a piano note actually gradually decreases in volume over time until note is finally released. Revisit idea of ADSR envelopes in Chap. 10 to see how this can be applied when creating synthesized musical instruments.

  – Âm thanh như trống snare có 1 cú đánh sắc nét & 1 cú nhả nhanh, trong khi âm thanh như chũm chọe hoặc chuông có các cú đánh nhanh nhưng các cú nhả chậm hơn, vang lên trong thời gian dài hơn. Các âm thanh khác như đàn violin có cả các cú đánh chậm hơn & các cú nhả. Các phần tấn công, suy giảm, & giải phóng của 1 lớp vỏ cũng có thể cong thay vì các đường thẳng, đôi khi gần đúng hơn với âm thanh của các nhạc cụ thực. Nhưng điều quan trọng cần nhớ: Các lớp vỏ ADSR luôn là sự đơn giản hóa của thực tế. E.g., độ duy trì của 1 nốt nhạc piano thực sự giảm dần về âm lượng theo thời gian cho đến khi nốt nhạc cuối cùng được nhả ra. Xem lại ý tưởng về các lớp vỏ ADSR trong Chương 10 để xem cách áp dụng điều này khi tạo ra các nhạc cụ tổng hợp.

- 7.3. Fourier. Jean-Baptiste Joseph Fourier was a French mathematician & physicist whose work in 19th century led to what we now call *Fourier Analysis*, a process through which we can decompose a complex sound signal into its constituent individual frequencies. Idea: can take any complex sound & determine all of frequencies that contribute to energy in signal – basically finding a set of sine waves that can be combined to represent a more complex waveform. Composition of sound signal by its frequency components is called signal's *spectrum*, & it can be generated through a mathematical operation called *Fourier transformation* – an essential part of all modern music production. For any given slice of time, spectrum might look like Fig. 7.1. But can also spread this information out over many time slices to visualize frequency & amplitude of a signal as it changes over longer periods of time. This visualization is called a *spectrogram* Fig. 7.5: A spectrogram shows intensity of frequencies in an audio signal over time. Heatmap colors correspond to intensity or energy at different frequencies. Time is represented on horizontal axis & frequency in kilohertz on vertical axis. A spectrogram typically shows time on horizontal axis, frequency on vertical axis, & intensity of different frequencies using heatmap colors. Warmer colors indicate more energy, while cooler colors intricate less energy.

– Jean-Baptiste Joseph Fourier là 1 nhà toán học & vật lý người Pháp, công trình của ông vào thế kỷ 19 đã dẫn đến cái mà ngày nay chúng ta gọi là *Phân tích Fourier*, 1 quá trình mà qua đó chúng ta có thể phân tích 1 tín hiệu âm thanh phức tạp thành các tần số riêng lẻ cấu thành nên nó. Ý tưởng: có thể lấy bất kỳ âm thanh phức tạp nào & xác định tất cả các tần số góp phần tạo nên năng lượng trong tín hiệu – về cơ bản là tìm 1 tập hợp các sóng sin có thể kết hợp để biểu diễn 1 dạng sóng phức tạp hơn. Thành phần của tín hiệu âm thanh theo các thành phần tần số của nó được gọi là *phổ* của tín hiệu, & nó có thể được tạo ra thông qua 1 phép toán gọi là *Biến đổi Fourier* – 1 phần thiết yếu của mọi hoạt động sản xuất âm nhạc hiện đại. Đối với bất kỳ lát cắt thời gian nào, phổ có thể trông giống như Hình 7.1. Nhưng cũng có thể phân tán thông tin này thành nhiều lát cắt thời gian để trực quan hóa tần số & biên độ của tín hiệu khi nó thay đổi trong khoảng thời gian dài hơn. Trực quan hóa này được gọi là *phổ đồ* Hình 7.5: Phổ đồ hiển thị cường độ tần số trong tín hiệu âm thanh theo thời gian. Màu sắc bản đồ nhiệt tương ứng với cường độ hoặc năng lượng ở các tần số khác nhau. Thời gian được biểu diễn trên trục ngang & tần số tính bằng kilohertz trên trục dọc. 1 quang phổ thường hiển thị thời gian trên trục ngang, tần số trên trục dọc, & cường độ của các tần số khác nhau bằng cách sử dụng màu sắc bản đồ nhiệt. Màu ấm hơn biểu thị nhiều năng lượng hơn, trong khi màu lạnh hơn biểu thị ít năng lượng hơn.

This representation helps producers see & understand properties of sounds e.g. timbre & loudness. A spectrogram might show unwanted noise in background, or point out: audio is heavy on lower frequencies & sounds tiny. Through years of training, music producers can interpret spectrograms to visually understand how various frequency bands contribute to a mix.

– Biểu diễn này giúp nhà sản xuất thấy & hiểu các đặc tính của âm thanh, ví dụ như âm sắc & độ to. 1 phổ đồ có thể hiển thị tiếng ồn không mong muốn trong nền hoặc chỉ ra: âm thanh nặng ở tần số thấp & nghe rất nhỏ. Qua nhiều năm đào tạo, nhà sản xuất âm nhạc có thể diễn giải phổ đồ để hiểu trực quan cách các dải tần số khác nhau đóng góp vào bản phối.

- 7.4. Mixing & mastering. Recording all parts of a song is only 1 part of process of creating a finished piece of music that's ready to be shared with world. A music producer still has task of making all of various sonic layers work together as a cohesive whole. How does bass line complement rhythm? Does it interfere with percussion sounds? Are vocals getting drowned out by instrumentals? Are instruments competing with 1 another? Is overall mix too muddy or harsh or boomy? Process of *mixing* is about overall compositional structure of a song & finding balance between individual musical elements that have been recorded, sampled, or generated. Of course, *rough* mixes get put together throughout creation process as different parts of a song are recorded. E.g., a recording studio would need a rough mix of drums, bass, & keyboards before overdubbing vocals. But final mix is when all of elements are balanced, placed in space, & blended together to make an artistic statement. Mixing can be a complex process that involves planning, deep listening, & a lot of patience to get it right.

  – Trộn & master. Thu âm tất cả các phần của 1 bài hát chỉ là 1 phần của quá trình tạo ra 1 bản nhạc hoàn chỉnh, sẵn sàng chia sẻ với thế giới. Nhà sản xuất âm nhạc vẫn có nhiệm vụ làm cho tất cả các lớp âm thanh khác nhau hoạt động cùng nhau như 1 tổng thể gắn kết. Dòng âm trầm bổ sung cho nhịp điệu như thế nào? Nó có can thiệp vào âm thanh của bộ gõ không? Giọng hát có bị lấn át bởi nhạc cụ không? Các nhạc cụ có cạnh tranh với nhau không? Bản phối tổng thể có quá đục, quá gắt hay quá ầm ầm không? Quá trình *mix* là về cấu trúc sáng tác tổng thể của 1 bài hát & tìm kiếm sự cân bằng giữa các yếu tố âm nhạc riêng lẻ đã được thu âm, lấy mẫu hoặc tạo ra. Tất nhiên, các bản phối *rough* được kết hợp lại trong suốt quá trình sáng tác khi các phần khác nhau của 1 bài hát được thu âm. Ví dụ: 1 phòng thu âm sẽ cần bản phối thô của trống, bass, & keyboard trước khi thu âm giọng hát. Nhưng bản phối cuối cùng là khi tất cả các yếu tố được cân bằng, đặt trong không gian, & hòa trộn với nhau để tạo nên 1 tuyên bố nghệ thuật. Việc phối nhạc có thể là 1 quá trình phức tạp đòi hỏi phải lập kế hoạch, lắng nghe sâu sắc & rất nhiều kiên nhẫn để làm đúng. *Deep listening* is process of paying close attention to relationship between musical elements in a song. This involves simultaneously being aware of compositional structure & frequency bandwidth of individual sounds. With deep listening, you are paying attention to different components & how they relate to each other. Is there a call-&-response between guitar & trumpet? Are they playing together? Should separate them by adjusting equalization to bring 1 out in foreground or will a simple change in volume do trick? Perhaps putting them in separate spaces within stereo spectrum will work, or using a 100 ms delay effect to place it in lateral space away from listener. There is a *lot* to experiment with, & it takes time to perfect art of mixing. It takes practice to develop this listening skill, but with practice you will become keenly aware of nuances. E.g., hear when instruments with same frequency bandwidth overlap. With digital tools that allow real-time frequency spectrum analysis, can actually see where they overlap. Don't worry about getting it right 1st time; mixing is a process that can require several iterations before you reach that "sweet" spot.

  – *Nghe sâu* là quá trình chú ý kỹ đến mối quan hệ giữa các yếu tố âm nhạc trong 1 bài hát. Điều này bao gồm việc đồng thời nhận thức được cấu trúc sáng tác & băng thông tần số của từng âm thanh. Với việc nghe sâu, bạn sẽ chú ý đến các thành phần khác nhau & cách chúng liên quan đến nhau. Có sự tương tác qua lại giữa guitar & trumpet không? Chúng có chơi cùng nhau không? Nên tách chúng ra bằng cách điều chỉnh cân bằng để đưa 1 ra phía trước hay chỉ cần thay đổi âm lượng là đủ? Có lẽ việc đặt chúng vào các không gian riêng biệt trong phổ âm thanh nổi sẽ hiệu quả hoặc sử dụng hiệu ứng trễ 100 ms để đặt chúng vào không gian bên ngoài, cách xa người nghe. Có rất nhiều để thử nghiệm, & cần thời gian để hoàn thiện nghệ thuật phối nhạc. Cần phải thực hành để phát triển kỹ năng nghe này, nhưng với sự thực hành, bạn sẽ nhận thức sâu sắc về các sắc thái. E.g., hãy lắng nghe khi các nhạc cụ có cùng băng thông tần số chồng lên nhau. Với các công cụ kỹ thuật số cho phép phân tích phổ tần số theo thời gian thực, bạn thực sự có thể thấy chúng chồng lên nhau ở đâu. Đừng lo lắng về việc phải làm đúng ngay từ lần đầu tiên; pha trộn là 1 quá trình có thể đòi hỏi nhiều lần lặp lại trước khi đạt được điểm "ngọt ngào" đó.

  In this chap, going to refer to each individual layer of music as a *track* [This use of word "track" has a different meaning than a track on an album.]. Common practice to record vocals, drums, bass, & so on, on separate tracks & then mix them together to form final product. In TunePad, tracks are created using cells that can be assembled on a timeline as parts of a

song. On a traditional mixing board, each *track* is a multifaceted tool used to shape sound elements in order to blend them cohesively with other elements in song Fig. 7.6: Mixing console with magnetic tape.

– Trong chương này, chúng ta sẽ gọi từng lớp nhạc riêng lẻ là 1 *track* [Cách sử dụng từ "track" này có nghĩa khác với 1 track trong album.]. Thực hành phổ biến là ghi âm giọng hát, trống, bass, & v.v., trên các track riêng biệt & sau đó trộn chúng lại với nhau để tạo thành sản phẩm cuối cùng. Trong TunePad, các track được tạo bằng cách sử dụng các ô có thể được lắp ráp trên 1 dòng thời gian như các phần của bài hát. Trên 1 bảng trộn âm truyền thống, mỗi *track* là 1 công cụ đa năng được sử dụng để định hình các thành phần âm thanh nhằm hòa trộn chúng 1 cách gắn kết với các thành phần khác trong bài hát Hình 7.6: Bàn trộn âm có băng từ.

A few of most important audio parameters to consider when mixing are panning, frequency manipulation (equalization), & gain (loudness of each track). This is also where you can apply audio effects like reverb, echo, or chorus (some of which cover in next chap).

– 1 số thông số âm thanh quan trọng nhất cần xem xét khi trộn là panning, điều chỉnh tần số (cân bằng), & gain (độ lớn của từng track). Đây cũng là nơi bạn có thể áp dụng các hiệu ứng âm thanh như reverb, echo hoặc chorus (một số trong số đó sẽ được đề cập trong chương tiếp theo).

* 7.4.1. Mixing tools. Before widespread availability of digital production tools, recording studios used multi-track magnetic tape to record multiple elements of a song e..g. bass, drums, guitar, keyboards, & vocals. Large mixing consoles were then used to record & play back to each element on tape Fig. 7.6. Studio infrastructure would route signals from individual tracks to & from mixing board. Today this is mainly done using software & visualization tools that allow for more flexibility & precision.

  – Công cụ trộn. Trước khi các công cụ sản xuất kỹ thuật số được sử dụng rộng rãi, các phòng thu âm đã sử dụng băng từ đa rãnh để ghi lại nhiều thành phần của 1 bài hát, ví dụ như bass, trống, guitar, keyboard, & giọng hát. Sau đó, các bàn điều khiển trộn lớn được sử dụng để ghi lại & phát lại từng thành phần trên băng Hình 7.6. Cơ sở hạ tầng phòng thu sẽ định tuyến tín hiệu từ các rãnh riêng lẻ đến & từ bảng trộn. Ngày nay, điều này chủ yếu được thực hiện bằng phần mềm & các công cụ trực quan hóa cho phép linh hoạt hơn & độ chính xác.

* 7.4.2. Panning & stereo. Most of music that you listen to has multiple *channels* of audio data. Stereophonic, or *stereo*, recordings use 2 different channels (left & right) to recreate spatial experience of listening to music in natural acoustic environments [Recordings with only 1 channel of audio data are called *monophonic*, or *mono*, recordings.]. I.e., when listening to music with headphones or earbuds, what you hear in your left ear is subtly (or note so subtly) different from what you hear in your right ear. Try removing 1 of your earbuds next time listening to music to see if can hear difference. When experience live music, you have a physical position in space relative to various musicians, vocalists, & other audio sources in room. Your 2 ears are also pointed in opposite directions, meaning they receive different versions of same audio scene. Music producers use stereo spectrum to recreate this experience. In general, humans have evolved to process sound from these 2 sources to create a mental map of physical space surrounding us. Think about a truck that drives past you. Even if you can't see truck, your brain is able to tell you where truck was & roughly how fast it was going based on frequency, loudness, & phase differences from your left & right ears.

  – Quét & âm thanh nổi. Hầu hết âm nhạc mà bạn nghe đều có nhiều *kênh* dữ liệu âm thanh. Bản ghi âm âm thanh nổi, hay *âm thanh nổi*, sử dụng 2 kênh khác nhau (trái & phải) để tái tạo trải nghiệm không gian khi nghe nhạc trong môi trường âm thanh tự nhiên [Bản ghi âm chỉ có 1 kênh dữ liệu âm thanh được gọi là bản ghi âm *đơn âm*, hay *đơn âm*.]. E.g., khi nghe nhạc bằng tai nghe hoặc nút tai, những gì bạn nghe thấy ở tai trái sẽ hơi khác (hoặc hơi khác 1 chút) so với những gì bạn nghe thấy ở tai phải. Hãy thử tháo 1 nút tai ra vào lần tới khi nghe nhạc để xem bạn có thể nghe thấy sự khác biệt không. Khi trải nghiệm âm nhạc trực tiếp, bạn có 1 vị trí vật lý trong không gian so với nhiều nhạc sĩ, ca sĩ, & các nguồn âm thanh khác trong phòng. 2 tai của bạn cũng hướng về các hướng ngược nhau, nghĩa là chúng nhận được các phiên bản khác nhau của cùng 1 cảnh âm thanh. Các nhà sản xuất âm nhạc sử dụng phổ âm thanh nổi để tái tạo trải nghiệm này. Nhìn chung, con người đã tiến hóa để xử lý âm thanh từ 2 nguồn này để tạo ra bản đồ tinh thần về không gian vật lý xung quanh chúng ta. Hãy nghĩ về 1 chiếc xe tải chạy ngang qua bạn. Ngay cả khi bạn không thể nhìn thấy xe tải, não của bạn vẫn có thể cho bạn biết xe tải đang ở đâu & tốc độ ước chừng của nó dựa trên tần số, độ lớn, & độ lệch pha từ tai trái & tai phải của bạn. Gửi ý kiến phản hồi

  Panning of a track refers to its position in this stereo spectrum. In practice, this means how much of track comes out of left & right speakers. Producers can create more depth to a song & replicate live recordings by controlling panning of tracks. Can almost think of it like arranging musicians on a stage in front of a live audience. Humans are also better at perceiving directionality of sound at higher frequencies, i.e., can easily tell which direction a high-pitched hi-hat sound is coming from, but we have a hard time telling which direction a bass line is coming from. As a result, producers will often pan higher-pitched sounds to left or right, while leaving lower-pitched sounds more in center of a mix.

  – *Panning* của 1 bản nhạc đề cập đến vị trí của nó trong phổ âm thanh nổi này. Trong thực tế, điều này có nghĩa là có bao nhiêu bản nhạc phát ra từ loa trái & phải. Nhà sản xuất có thể tạo thêm chiều sâu cho 1 bài hát & sao chép các bản ghi âm trực tiếp bằng cách kiểm soát việc panning các bản nhạc. Gần như có thể nghĩ về nó giống như việc sắp xếp các nhạc công trên sân khấu trước khán giả trực tiếp. Con người cũng giỏi hơn trong việc nhận biết hướng của âm thanh ở tần số cao hơn, tức là có thể dễ dàng biết được âm thanh hi-hat cao độ phát ra từ hướng nào, nhưng chúng ta khó có thể biết được hướng của dòng âm trầm phát ra từ hướng nào. Do đó, nhà sản xuất thường sẽ pan các âm thanh có cao độ cao sang trái hoặc phải, trong khi để các âm thanh có cao độ thấp hơn ở giữa bản phối.

  In TunePad, can adjust pan, gain, & frequency elements of different cells using mixer interface shown in Fig. 7.7: Mixing interface in TunePad allows you to adjust gain, pan, & frequency response for each track in a mix. Also possible to apply these effects in code using Python's `with` construct. An example of a pan effect that shifts stereo balance of 2 `playNote`

instructions to far left speaker.

```
with pan(-1.0):
    playNote([ 31, 35, 38 ], beats = 4)
    playNote([ 31, 35, 38 ], beats = 4)
```

Values of pan parameter ranges from −1.0 (full left speaker) to 1.0 (full right speaker). A value of 0.0 evenly splits sound. `with` keyword in TunePad applies pan effect to all of statements indented directly below it.

* 7.4.3. Gain. *Gain* of a track is related to its loudness. Gain isn't quite volume, but works as kind of a multiplier to an audio signal's amplitude. When mixing boards were physical pieces of equipment, gain related to amount of power a signal had at each stage of signal flow. Now, gain has a similar meaning & can be used to make a track more or less prominent. E.g., a producer may choose to make bass drum of a dance track more prominent while decreasing gain of vocal melody. Gain is commonly measured in decibels. Negative values reduce loudness of a track, & positive values increase it from its original volume.

  – *Gain* của 1 bản nhạc liên quan đến độ to của bản nhạc đó. Gain không hẳn là âm lượng, nhưng hoạt động như 1 loại hệ số nhân với biên độ của tín hiệu âm thanh. Khi bảng trộn là các thiết bị vật lý, gain liên quan đến lượng công suất mà tín hiệu có ở mỗi giai đoạn của luồng tín hiệu. Bây giờ, gain có ý nghĩa tương tự & có thể được sử dụng để làm cho 1 bản nhạc nổi bật hơn hoặc ít nổi bật hơn. E.g., 1 nhà sản xuất có thể chọn làm cho tiếng trống trầm của 1 bản nhạc khiêu vũ nổi bật hơn trong khi giảm gain của giai điệu giọng hát. Gain thường được đo bằng decibel. Các giá trị âm làm giảm độ to của 1 bản nhạc, & các giá trị dương làm tăng nó so với âm lượng ban đầu của nó.

* 7.4.4. Frequency bands. When mixing tracks together, often helpful to break full frequency spectrum into *bands* that correspond to different ranges of frequencies. Each band is meant to capture a particular musical element, although, of course, this varies between genres & specific songs. Producers often split a mix into 7 bands: sub-bass, bass, low midrange, midrange, upper midrange, presence, & brilliance. 1 reason to think in terms of bandwidth: when sounds have same bandwidth an acoustic phenomenon called "masking" can occur. Masking is when 1 sound overpowers another sound s.t. sound that is overpowered is not audible.

  [Table: Band: Frequency range: Description]

  · Sub-bass: 20–60 Hz: Adds power & deepness to bass & drums

  · Bass: 60–250 Hz: Captures core fundamentals of bass & drum sound

  · Low midrange: 250–500 Hz: Captures overtones of lower instruments, as well as instruments like viola & alto saxophone

  · Midrange: 500–2000 Hz: Captures melodic instruments e.g. violin, flute, & human voice

  · Upper midrange: 2000–4000 Hz: Captures overtones of melodic instruments as well as core of some higher instruments

  · Presence: 4000–6000 Hz: Captures overtones of higher instruments as well as adding precision & clarity to sounds

  · Brilliance: 6000–20000 Hz: Captures upper overtones of all instruments

  Humans are most sensitive to frequencies between 1 kHz & 4 kHz. Looking at frequency values for each band, might notice: frequency ranges are not even close to same size. E.g., sub-bass band covers a range of only 40 Hz (from 20 Hz–60 Hz), while presence band covers 2000 Hz (from 4000 Hz–6000 Hz). Reason: human perception of pitch isn't linear. When move up 1 octave, doubling frequency of a pitch, i.e., each consecutive musical octave covers double frequency range (or bandwidth) of octave below it. As a result, higher frequency bands naturally cover large portions of frequency range & generate more energy.

  – Con người nhạy cảm nhất với tần số từ 1 kHz & 4 kHz. Khi xem xét các giá trị tần số cho từng băng tần, bạn có thể nhận thấy: các dải tần số thậm chí không gần bằng nhau. E.g., dải âm trầm phụ chỉ bao phủ 1 dải tần 40 Hz (từ 20 Hz–60 Hz), trong khi dải hiện diện bao phủ 2000 Hz (từ 4000 Hz–6000 Hz). Lý do: nhận thức của con người về cao độ không phải là tuyến tính. Khi di chuyển lên 1 quãng tám, tần số của 1 cao độ tăng gấp đôi, tức là mỗi quãng tám âm nhạc liên tiếp bao phủ gấp đôi dải tần số (hoặc băng thông) của quãng tám bên dưới nó. Do đó, các dải tần số cao hơn tự nhiên bao phủ các phần lớn của dải tần & tạo ra nhiều năng lượng hơn.

◦ 7.5. Filters & equalization. If 2 instruments overlap in their natural pitch range, it can be difficult to distinguish 1 from the other, which can lead to muddiness. Producer will want to ensure: each musical element is distinct & audible. Think of a painter recreating an ocean scene. Painter wants each element of scene to stand out clearly – perhaps sky, a boat, shore, & ocean itself. If ocean, sky, & land are all same shade of blue, a viewer won't be able to interpret & appreciate scene.

– Bộ lọc & cân bằng. Nếu 2 nhạc cụ chồng lên nhau trong phạm vi cao độ tự nhiên của chúng, có thể khó phân biệt nhạc cụ này với nhạc cụ kia, điều này có thể dẫn đến sự hỗn loạn. Nhà sản xuất sẽ muốn đảm bảo: mỗi yếu tố âm nhạc đều riêng biệt & có thể nghe được. Hãy nghĩ đến 1 họa sĩ đang tái hiện 1 cảnh đại dương. Họa sĩ muốn mỗi yếu tố của cảnh nổi bật rõ ràng – có thể là bầu trời, 1 chiếc thuyền, bờ biển, & chính đại dương. Nếu đại dương, bầu trời, & đất liền đều có cùng 1 sắc thái xanh, người xem sẽ không thể diễn giải & đánh giá cao cảnh đó.

Most important tools that a producer has to achieve balance across frequency spectrum are *filters & equalizers*. These tools reduce (*attenuate*) or increase (*boost*) certain frequency ranges in a track to make them more or less prominent in a mix, & producers will often "carve out" room in frequency spectrum for each track. Process of adjusting levels of frequency bands within a signal is called *equalization* (or *EQ*). When adjust bass & treble dials of sound system in your car, you are equalizing frequencies just as a producer might while adjusting sound of an instrument.

– Các công cụ quan trọng nhất mà nhà sản xuất phải có để đạt được sự cân bằng trên toàn bộ phổ tần số là *bộ lọc & bộ cân bằng*. Các công cụ này làm giảm (*làm suy yếu*) hoặc tăng (*tăng cường*) các dải tần số nhất định trong 1 bản nhạc để

làm cho chúng nổi bật hơn hoặc ít nổi bật hơn trong bản phối, & nhà sản xuất thường sẽ "tạo ra" chỗ trong phổ tần số cho mỗi bản nhạc. Quá trình điều chỉnh mức độ của các dải tần số trong 1 tín hiệu được gọi là *cân bằng* (hoặc *EQ*). Khi điều chỉnh núm xoay âm trầm & âm bổng của hệ thống âm thanh trong xe hơi của bạn, bạn đang cân bằng tần số giống như 1 nhà sản xuất có thể làm khi điều chỉnh âm thanh của 1 nhạc cụ.

Can think of an audio filter kind of like a filter that you would use to purify drinking water. A water filter is designed to let small particles (like water molecules & minerals) pass through while blocking larger particles (like bacteria). An audio filter achieves a similar effect except for sound, allowing certain frequencies of an audio signal to pass through unaffected while blocking or reducing other frequencies. A filter's *response curve* is a graph that shows which frequencies are allowed to pass through & which are filtered out. There are several types of filters common used in music production including lowpass, highpass, lowshelf, highshelf, bandpass, notch, & peaking. Describe several of these filters below along with an example of applying these filters with Python code in TunePad. Most production software (including TunePad) include built-in equalizer tools that let you combine & precisely adjust various filter types. Understanding how each filter works will help use these tools.

– Có thể nghĩ về bộ lọc âm thanh giống như bộ lọc mà bạn sử dụng để làm sạch nước uống. Bộ lọc nước được thiết kế để cho các hạt nhỏ (như phân tử nước & khoáng chất) đi qua trong khi chặn các hạt lớn hơn (như vi khuẩn). Bộ lọc âm thanh đạt được hiệu ứng tương tự ngoại trừ âm thanh, cho phép 1 số tần số nhất định của tín hiệu âm thanh đi qua mà không bị ảnh hưởng trong khi chặn hoặc giảm các tần số khác. *response curve* của bộ lọc là đồ thị hiển thị tần số nào được phép đi qua & tần số nào bị lọc ra. Có 1 số loại bộ lọc thường được sử dụng trong sản xuất âm nhạc bao gồm thông thấp, thông cao, kệ thấp, kệ cao, thông dải, khía, & peaking. Mô tả 1 số bộ lọc này bên dưới cùng với ví dụ về cách áp dụng các bộ lọc này bằng mã Python trong TunePad. Hầu hết các phần mềm sản xuất (bao gồm TunePad) đều bao gồm các công cụ cân bằng tích hợp cho phép bạn kết hợp & điều chỉnh chính xác nhiều loại bộ lọc khác nhau. Hiểu cách thức hoạt động của từng bộ lọc sẽ giúp sử dụng các công cụ này.

∗ 7.5.1. Lowpass filer. A *lowpass filter* allows frequencies below a certain threshold – called *cutoff frequency* – to pass through unaltered. Frequencies above this threshold are reduced (or attenuated). A frequency parameter specifies location of cutoff, & a Q parameter determines how sharp or steep this cutoff is Fig. 7.8: Lowpass filter response curve.

– Bộ lọc thông thấp. 1 *bộ lọc thông thấp* cho phép các tần số dưới ngưỡng nhất định – được gọi là *tần số cắt* – đi qua mà không bị thay đổi. Các tần số trên ngưỡng này bị giảm (hoặc suy yếu). 1 tham số tần số chỉ định vị trí cắt, & 1 tham số Q xác định mức độ sắc nét hoặc dốc của điểm cắt này Hình 7.8: Đường cong đáp ứng của bộ lọc thông thấp.

Lowpass filers might be applied if a track sounds too bright, or to remove some of higher partials of a bass instrument to make room for other instruments in a mix, or even to remove some unwanted studio sound e.g. buzzing from equipment.

– Bộ lọc thông thấp có thể được áp dụng nếu 1 bản nhạc nghe quá sáng hoặc để loại bỏ 1 số phần cao hơn của nhạc cụ trầm để tạo chỗ cho các nhạc cụ khác trong bản phối hoặc thậm chí để loại bỏ 1 số âm thanh phòng thu không mong muốn, ví dụ như tiếng ù từ thiết bị.

In TunePad, can add a lowpass filter directly in Python code. Example below applies a constant lowpass filter with a cutoff of 100 Hz to reduce higher frequencies in drums.

```
with lowpass(frequency = 100):
    playNote(0, beats = 1)
    playNote(2, beats = 1)
    playNote(0, beats = 1)
    playNote(2, beats = 1)
```

`with` keyword starts a special Python structure that applies an effect to all of statements indented below it. In this case, TunePad's lowpass filter is applied to 4 drum sounds.

All of filters that you can code in TunePad have same basic structure. Use `with` keyword followed by filter name. Filters have 1 required parameter & several optional parameters. Only required parameter is frequency, which represents cutoff frequency for each filter. Filters also have an optional Q parameter, which specifies how sharp or spread out frequency cutoff is around target frequency.

[Table: Parameter: Description: Required?]

· `Frequency`: Cutoff or central frequency specified in Hz: Yes

· `Q`: Typically sharpness of cutoff frequency: No

· `Beats`: How long effect lasts in beats: No

· `Start`: How long to delay in beats before starting effect: No

· `Gain`: Some filters like peaking, lowshelf, & highshelf use a gain parameter to specify intensity of boost or attenuation in decibels: No

∗ 7.5.2. Highpass filter. A *highpass filter* is opposite of a lowpass filter; it passes frequencies *above* cutoff & reduces frequencies below. As with lowpass filters, frequency parameter sets cutoff frequency & Q parameter specifies sharpness of cutoff Fig. 7.9: Highpass filter response curve.

A highpass filter might be applied if a track sounds muddy because it has too much bass, or to remove unwanted noise e.g. a low hum from equipment. A TunePad example that uses a highpass filter to cut out sounds lower than 4000 Hz (4kHz) for an instrument playing a melody:

```
with highpass(frequency = 4000):
    playNote(31, beats = 0.5)
```

```
playNote(35, beats = 0.5)
playNote(38, beats = 1)
playNote(36, beats = 1)
```

* 7.5.3. Bandpass filter. A *bandpass filter* reduces frequencies above & below a specified band of frequencies; a bandpass is equivalent of applying both a lowpass & a highpass filter. Specify middle of band using frequency parameter & width of band using Q parameter. Higher Q, sharper cutoff, & narrower band of frequencies that can pass through. Bandpass filters allow us to precisely target a track's frequency range. Might use a bandpass filter to bring out vocals or melody of a song by reducing everything else Fig. 7.10: Bandpass filter response curve.
  In example, apply a constant bandpass filter with a center frequency of 130 Hz (≈ C3 or MIDI 48) to a short melody to bring melody out in overall musical texture.

```
with bandpass(frequency = 130, Q = 0.7):
    playNote(48, beats = 0.5)
    playNote(52, beats = 0.5)
    playNote(55, beats = 1)
    playNote(53, beats = 1)
```

* 7.5.4. Notch filter. A *notch filter* is opposite of a bandpass filter. Rather than bringing out a band of frequencies, a notch filter *reduces* frequency band while all other frequencies pass through freely. Like with bandpass, frequency parameter specifies center of this frequency band & Q parameter sets width Fig. 7.11: Notch filter response curve.
  In example, apply a constant notch filter with a center frequency of 440 Hz (≈ A4or MIDI 69) to a short selection of chords to reduce prevalence in overall musical texture.

```
with notch(frequency = 440):
    playNote([69, 72, 76], beats = 4)
    playNote([69, 72, 76], beats = 4)
```

* 7.5.5. Peaking filter. Peaking filters are frequently used in *parametric equalizers* to boot or attenuate sounds at a target frequency. Parametric equalizers are a type of equalizer that offer precise control of center frequencies & Q (how spread out or tight filter is around center frequency). Using these filters, there's a 3rd parameter called *gain* that controls how much signal is boosted or attenuated. Gain is measured in decibels. A positive gain will boost frequencies targeted by filter, & a negative gain will attenuate them Fig. 7.12: Peaking filer response curve.
  – Bộ lọc đỉnh. Bộ lọc đỉnh thường được sử dụng trong *bộ cân bằng tham số* để khởi động hoặc làm suy yếu âm thanh ở tần số mục tiêu. Bộ cân bằng tham số là 1 loại bộ cân bằng cung cấp khả năng kiểm soát chính xác tần số trung tâm & Q (mức độ lan tỏa hoặc chặt chẽ của bộ lọc xung quanh tần số trung tâm). Khi sử dụng các bộ lọc này, có 1 tham số thứ 3 được gọi là *gain* kiểm soát mức độ tín hiệu được tăng cường hoặc suy yếu. Độ khuếch đại được đo bằng decibel. Độ khuếch đại dương sẽ tăng cường tần số mục tiêu của bộ lọc, & độ khuếch đại âm sẽ làm suy yếu chúng Hình 7.12: Đường cong phản hồi của bộ lọc đỉnh.
* 7.5.6. Lowshelf & highshelf filters. Lowshelf & highshelf filters boost or attenuate sounds beyond target frequency. They are called *shelves* due to plateau shape of their response curves. As with peaking filters, frequency parameter specifies cutoff, & gain parameter specifies how much boost or attenuation to give to frequencies beyond target. Negative gain values attenuate & positive gain values boost Fig. 7.13: Low shelf & high shelf response curves.
  – Bộ lọc Lowshelf & highshelf. Bộ lọc Lowshelf & highshelf tăng cường hoặc làm suy yếu âm thanh vượt quá tần số mục tiêu. Chúng được gọi là *shelves* do hình dạng cao nguyên của đường cong phản hồi của chúng. Giống như bộ lọc đỉnh, tham số tần số chỉ định điểm cắt, tham số & gain chỉ định mức tăng cường hoặc làm suy yếu nào để cung cấp cho các tần số vượt quá mục tiêu. Giá trị khuếch đại âm làm suy yếu & giá trị khuếch đại dương tăng cường Hình 7.13: Đường cong phản hồi low shelf & high shelf.

◦ 7.6. Mastering. After individual tracks have been adjusted in relation to 1 another, *mastering* is process of taking this final mix & polishing it by adjusting global parameters e.g. dynamic range & frequency. In early days, there were mastering engineers who specialized in process of mastering final mixes. In fact, there were studios dedicated to mastering, so can imagine that this last leg of production process deserves as much attention as the rest. Mastering is particularly important because you want your mix to sound good on as many devices as possible, so there is a delicate process of balancing elements in mix to optimize listening experience across different media. Want your mix to sound as good over speakers as it does over headphones.

– Mastering. Sau khi từng track riêng lẻ đã được điều chỉnh liên quan đến 1 track khác, *mastering* là quá trình thực hiện bản phối cuối cùng này & đánh bóng nó bằng cách điều chỉnh các thông số toàn cục, ví dụ như dải động & tần số. Vào những ngày đầu, có những kỹ sư mastering chuyên về quá trình mastering các bản phối cuối cùng. Trên thực tế, đã có những studio chuyên về mastering, vì vậy có thể tưởng tượng rằng giai đoạn cuối cùng của quy trình sản xuất này cũng đáng được quan tâm như phần còn lại. Mastering đặc biệt quan trọng vì bạn muốn bản phối của mình nghe hay trên càng nhiều thiết bị càng tốt, do đó, có 1 quy trình tinh tế để cân bằng các yếu tố trong bản phối nhằm tối ưu hóa trải nghiệm nghe trên các phương tiện khác nhau. Muốn bản phối của bạn nghe hay qua loa cũng như qua tai nghe.

Traditionally, mastering is done using tools like equalization, compression, limiting, & stereo enhancement. Recall from Chap. 3: dynamic range refers to difference between quietest & loudest volumes in a selection of audio. This can be adjusted

through use of *dynamic range compression*, or *compressors*. Compressors reduce highest volumes in a mix & amplify lowest volumes, which shrinks overall dynamic range of audio. This ensures: listener can hear full range of volumes clearly. Can think of this like an action movie where a character might whisper a secret right before an explosion. Dynamic range compression is 1 possible tool that could make sure that both of these sounds are clear to audience by reducing volume of explosion & increasing volume of whisper.

– Theo truyền thống, việc master được thực hiện bằng các công cụ như cân bằng, nén, giới hạn, & tăng cường âm thanh nổi. Nhớ lại từ Chương 3: dải động đề cập đến sự khác biệt giữa âm lượng nhỏ nhất & to nhất trong 1 lựa chọn âm thanh. Điều này có thể được điều chỉnh thông qua việc sử dụng *nén dải động* hoặc *máy nén*. Máy nén giảm âm lượng cao nhất trong bản phối & khuếch đại âm lượng thấp nhất, làm giảm dải động tổng thể của âm thanh. Điều này đảm bảo: người nghe có thể nghe rõ toàn bộ dải âm lượng. Có thể coi đây giống như 1 bộ phim hành động, trong đó 1 nhân vật có thể thì thầm 1 bí mật ngay trước khi xảy ra vụ nổ. Nén dải động là 1 công cụ khả thi có thể đảm bảo rằng cả hai âm thanh này đều rõ ràng đối với khán giả bằng cách giảm âm lượng của vụ nổ & tăng âm lượng của tiếng thì thầm.

Producer also considers frequency domain when creating final product. Instead of thinking on a track-by-track level as in mixing, producer can think in terms of different bands of frequencies. By this stage, our different bands of frequencies should already be well balanced, & goal: polish overall mix using EQ & filters.

– Nhà sản xuất cũng xem xét miền tần số khi tạo ra sản phẩm cuối cùng. Thay vì suy nghĩ theo từng track như trong quá trình trộn, nhà sản xuất có thể suy nghĩ theo các dải tần số khác nhau. Đến giai đoạn này, các dải tần số khác nhau của chúng ta đã được cân bằng tốt, & mục tiêu: đánh bóng bản phối tổng thể bằng EQ & bộ lọc.

Lastly, a final version of track is generated & exported into final format. A major consideration is where & how music is going to be distributed. A producer might think about a person watching a music video on a laptop through YouTube, vs. someone listening to radio in a car, vs. someone streaming audio online, vs. someone with a physical CD or even vinyl recording. Music for streaming & other forms of distribution is almost always *compressed*, i.e. size of final audio file is much, much smaller than original uncompressed audio data. Note: this is not related to dynamic range compression. Compressing audio means: some of data is discarded to decrease amount of information that has to be transmitted over internet to avoid buffering delays or to store more songs on a CD. There are complex computer algorithms that decide what data is discarded so that listeners won't even notice a reduction in quality. Examples of file formats that use this form of compression include `.mp3, .aac` files.

Some audio formats forgo this compression in favor of increased sound quality & fidelity. These files contain raw audio data. These audio files are generally larger, taking up more file space. Examples of this include `.wav & .aiff` file formats.

Mixing & mastering can be a tedious process requiring attention to detail & a keen, well-trained ear. Becoming an accomplished professional can take many years of experience, & have introduced just a few of parameters & tools at your disposal. Don't stress over this, especially at 1st! Mixing & mastering are 2 of most difficult concepts in producing music, but having a grasp of them can greatly elevate music you create. Best way to gain this familiarity is through experimentation & thoughtful listening. Listening critically to music that has been professionally produced will help develop your ear & unlock a whole new world of possibilities to your music.

– Mixing & mastering có thể là 1 quá trình tẻ nhạt đòi hỏi sự chú ý đến từng chi tiết & 1 đôi tai nhạy bén, được đào tạo bài bản. Để trở thành 1 chuyên gia thành đạt có thể mất nhiều năm kinh nghiệm, & đã giới thiệu 1 vài thông số & công cụ mà bạn có thể sử dụng. Đừng căng thẳng về điều này, đặc biệt là lúc đầu! Mixing & mastering là 2 trong số những khái niệm khó nhất trong sản xuất âm nhạc, nhưng nắm bắt được chúng có thể nâng cao đáng kể âm nhạc bạn tạo ra. Cách tốt nhất để có được sự quen thuộc này là thông qua thử nghiệm & lắng nghe 1 cách chu đáo. Việc lắng nghe 1 cách phê phán âm nhạc được sản xuất 1 cách chuyên nghiệp sẽ giúp phát triển đôi tai của bạn & mở ra 1 thế giới hoàn toàn mới về khả năng cho âm nhạc của bạn.

- 7.7. Dynamic effects in TunePad. Many of TunePad effects described can also be applied dynamically to create a wide variety of sounds. Basic idea: instead of passing 1 constant value for a parameter, instead pass a list of numbers that describes how that parameter will change over time. Duration of dynamic effect is specified using a `beats` parameter. Can try these dynamic effects & filters for yourself at https://tunepad.com/examples/efects. An example that uses a lowpass filter to create a wha-wha effect at beginning of a piano note. Cutoff frequency of filter moves rapidly back & forth between 200 Hz & 800 Hz over duration of 1 beat. Note itself plays for 3 beats, so only 1st beat of note has effect applied:

```
# creates a wha-wha effect by quickly changing
# the cutoff of a lowpass filter between 200 and 800hz
with lowpass(frequency = [200, 800, 200, 800, 200, 800], beats=1):
    playNote(47, beats=3)
```

Other effects like pan & gain work same way. Can create dynamic changes by passing in a list of values & `beats` parameters. E.g., this code gradually sweeps a lowpass filter from 20 Hz to 750 Hz & back again. At same time, it moves sound across stereo field from left to right & back again. To do this, it nests pan effect inside of lowpass filter effect.

```
with lowpass(frequency = [20, 750, 20], beats = 40):
    with pan(value = [-1, 1, -1], beats = 40):
        playNote(16, beats=40)
```

Interlude following this chap shows how to add other dynamic effects to TunePad projects using Python code.

- Interlude 7: Creative Effects. In this interlude going to work with audio effects – e.g. filtering & gain – as creative compositional tools. In Chap. 7, saw how static effects like filters & gain can be used in mixing process. These same effects can also be used as compositional tools to help craft a cohesive soundscape. They can build tension, add contrast, & drive a song forward. Can follow along online with this TunePad project https://tunepad.com/interlude/efects.
  - Option 1: Fades. Fades are 1 of most common creative effects. Often ear fades in or out at beginnings or ends of songs. These fades are essentially gradually moving from 1 volume to another: for a fade-in, low to high; for a fade-out, high to low.

    Can easily add fades to our music in TunePad using gain effect & a list. If gain class is passed a list, it moves evenly from each value to next, over whole duration of beats. For a fade-in, want to move from silent to full volume. Say have a function called `phrase` that plays 8 beats of a melody. Can use gain class with a list as values input:

    ```
    with gain([0.0, 1.0], beats = 8):
        phrase()
    ```

    Or, if want to fade out our phrase:

    ```
    with gain([1.0, 0.0], beats = 8):
        phrase()
    ```

    What if want to control speed of our fade? Because gain moves smoothly from each value, can shape our fade by adding more intermediate values $\in [0.0, 1.0]$. Can think of this as specifying more points along curve. If want our fade-in to be quieter for longer, could add additional values closer to 0:

    ```
    with gain([0.0, 0.05, 0.1, 0.15, 1.0], beats = 8):
        phrase()
    ```

    Can see difference graphically below Fig. 7.14: Graph of 2 methods for fading audio in.

    With method 1, ramp-up in volume is consistent & gradual over entire duration. But with method 2, ramp-up is slow until 7th beat. These last 2 beats have greatest increase in gain, increasing from 0.15–1.0.
  - Option 2: Filter sweeps. Filter sweeps are a great way to build tension in a song. Principle behind a filter sweep: apply a filter to a sect of a song – which blocks some of frequencies – & then gradually remove it, revealing full spectrum of audio. This can either highpass or lowpass filters, each imparting a different sound to our track. A lowpass filter will start with just low frequencies & will gradually reveal upper overtones while a highpass filter will do opposite.

    In TunePad, this is going to look similar to our fades. Pass our filter a list of 2 more numbers. However, these numbers will represent frequencies, so need to decide which values to use. Like with our fades, want a higher value & a lower value. Exact values depend largely on specific bandwidth of our instrument, desired speed of our sweep, & personal taste. For our highpass sweep, want our initial value to block most of frequencies, & our final value to include most of spectrum of audio. Our higher value can be a frequency near upper range of human hearing; for our lower value, could choose a frequency closer to bottom end of human hearing. Look now at code for a basic highpass filter sweep using our phrase function & values 22000 Hz & 22 Hz:

    ```
    with highpass([22000, 22], beats = 8):
        phrase()
    ```

    For lowpass filter sweep, also want to initially block most of component frequencies & gradually reveal upper frequencies of audio. Due to how lowpass filters work, lower number should be 1st. Can use initial value of 22 Hz to block most of frequencies & end at 4000 Hz, revealing most of our spectrum. Look now at code for a basic lowpass filter sweep, again using our phrase function with our chosen values:

    ```
    with lowpass([22, 4000], beats = 8):
        phrase()
    ```

    If want to have greater control of shape our our sweep, can use same principle behind our gain. Adding more values allows us to better sculpt our resulting sound. If want our filter to evolve slowly at 1st, can add additional values near our starting frequency. An example of this with our highpass sweep:

    ```
    with highpass([22000, 18000, 14000, 8000, 22], beats = 8):
        phrase()
    ```

    Most of audio spectrum is revealed between 8000 Hz & 22 Hz values, which occurs on last 2 beats. Try experimenting with different values.

- Option 3: Reverb. 1 of most important effects that producers use is reverb. When sound waves move through a physical space, some of those waves bounce back to listener. Waves that bounce back are heard as softer. Think of how sound reverberates through a concert hall, or even your bathroom. Can recreate this reverberation through applying reverb to our track.

  There are different mathematical strategies for applying reverb to audio. TunePad uses *convolution reverb*, which is 1 of most common varieties. This takes an audio sample called an *Impulse Response* from a real-world space that represents how different frequencies resonate through physical space & essentially maps this Impulse Response over our selection of audio.

  In TunePad, specify our impulse response by choosing from a selection of preset values:

  * Hall
  * Gallery
  * Museum
  * Library
  * Theater
  * Underpass
  * Space Echo 2

  This is 1st argument. Capitalization, spacing, & punctuation are ignored. Like other effect classes, can also optionally specify amount of beats effect should last & a delayed start parameter. This effect also uses a parameter called `wet`. This specifies how much reverb is applied. A value of 1.0 represents maximum reverb, & a value of 0.0 represents no reverb. Can also pass a list of numbers to create a change over time. Each number will be evenly distributed over duration of effect specified by `beats` parameter. In example, gradually applying "Underpass" reverb, which is most reverberant, to our phrase function:

  ```
  with reverb(impulse = "Underpass", wet = [0.0, 1.0], beats = 8):
      phrase()
  ```

  Experiment with different impulse responses!

- 8. Note-based production effects. This chap covers a variety of production effects that can add sophistication & depth to your sound. All of effects are variations on same theme – instead of playing 1 note, play a series of notes, each offset slightly in time or pitch. From this basic technique, work through a wide variety of effects including echos, arpeggiation, chorus sounds, a swung beat, & a phaser. To create these effects, define some of our own functions that make use of TunePad's `rewind`, `fastForward` features. This will also give us a good chance to review loops, variables, & parameters from earlier chaps. Once master these basic techniques, it will open a range of audio effects that you can expand & customize to define your own unique sound. Also cover how to combine these techniques with other effects & filters to add even more flexibility.

  - 8.1. Out-of-tune piano effect. Start with 1 of more straightforward effects, an out-of-tune piano. As always, can try this example by visiting https://tunepad.com/examples/out-of-tune. Recall from Chap. 3: space between separate notes on a 12-tone chromatic scale can be subdivided into even units called *cents* – just imagine 100 individual smaller notes between each adjacent key on piano keyboard. On an instrument like a violin or trombone, can play notes that are slightly out of tune or that glide between 1 note & another. Can do sth similar in TunePad by using decimal numbers instead of integer values when call `playNote` function:

    ```
    playNote(36.5) # plays an out-of-tune C
    ```

    For this line of code, a value of 36.5 sits exactly halfway between a C (MIDI value 36) & a C# (MIDI value 37). I.e., it's a pure C *detuned* by 50 cents Fig. 8.1: Intermediate pitches between C & C#.

    There are plenty of artistic reasons to create sounds that are out of tune – like if wanted to create an eerie melody for a horror movie. To get this kind of effect, could just randomly *detune* some of notes in our melody by small amounts, & get sth that sounded off key. But a more interesting approach that gives us additional texture (& eerie dissonant overtones) would be to play several notes at same time, each slightly detuned from 1 another. This actually approximates what a real out-of-tune piano sounds like. Notes on a piano are produced when a hammer mechanism inside instrument his multiple individual strings at same time (3 strings for most notes). When those individual strings are off from 1 another, hear an entirely different sound than you would get from just 1 string being out of tune. Honky tonk pianos are intentionally tuned so that 3 strings for each note are detuned slightly from 1 another to get warped harmonics. A simple Python function that approximates that dissonant sound in TunePad:

    ```
    def errieNote(note, beats = 1, velocity = 100):
        volume = velocity / 3.0
        for i in range(3):
            offset = random() -0.5
            playNote(note + offset, beats = beats, velocity = volume)
            rewind(beats)
        fastForward(beats)
    ```

Walk through this function 1 line at a time. If this example makes sense, all of other functions that we create later in this chap should be easier to understand. On line 1, use `def` keyword to define our own Python function.

```
def errieNote(note, beats = 1, velocity = 100):
```

Flip back to Chap. 4 for an in-depth description of function defs, but main thing to remember: creating our own functions lets us build up our own musical toolbox to help create more complex compositions. In this case adding `eerieNote` as our newest tool. Also define 3 *parameters* for this function called `note, beats, & velocity`, each of which is listed inside parentheses. Use `note` for pitch value want to play; `beats` for duration of note; & `velocity` to approximate overall volume of sound. Might notice: these parameters are exactly same as `playNote` function. That's intentional because it will make it easy to swap out `playNote` function for our new `eerieNote` function in other parts of our project. 1 other thing to notice: `beats, velocity` are examples of what's called an *optional* parameter. I.e., have defined a default value that Python will use if don't otherwise specify sth. Default value for `beats` is 1, & default value for `velocity` is 100. So, e.g., each of these lines of code will do same thing, & can use them interchangeably:

```
eerieNote(36)
eerieNote(36, beats = 1)
eerieNote(36, beats = 1, velocity = 100)
eerieNote(36, 1, 100)
eerieNote(36)
```

In all of these cases, `beats, velocity` are same as their default values. Can call function with other values too:

```
eerieNote(40, beats = 0.5, velocity = 50)
eerieNote(40, beats = 1.5)
eerieNote(40, velocity = 120)
eerieNote(40, 2.0, 50)
```

Go back to `eerieNote` function. On line 2 define a *variable* called `volume` that will help us adjust loudness of our individual notes.

```
volume = velocity / 3.0
```

Doing this because playing all notes at full volume would end up being much louder than sound of a single note. Set its value to `velocity` parameter divided by 3.0 because going to end up playing 3 notes instead of 1, so want each individual note to make up about $\frac{1}{3}$ of overall volume.

On line 3, set up our `for` loop to play 3 notes. Can also experiment with loops that repeat different numbers of times. Remember `range` is just a Python function that generates a sequence of numbers $[0, 1, 2]$ that variable i will walk through, 1 number at a time.

```
for i in range(0, 3):
```

On line 4, use Python's `random` function to generate a random decimal number somewhere between 0 & 1. Subtracting 0.5 will then gives us a number in range of negative -0.5 to 0.5. Save result in variable called `offset` that will use on next line of code to shift our note's pitch.

```
offset = random() - 0.5
```

Line 5 then plays note with our random pitch offset added in ($\in (-0.5, 0.5)$) to make it sound out of tune. Notice: use our `beats, volume` variables to control duration & volume of note that gets played, just as we might if were calling `playNote`.

```
playNote(note + offset, beats = beats, velocity = volume)
```

Last 2 lines of function make it so that all of notes get played at same time. On line 6, going to use a TunePad command called `rewind` to move playhead back to where we were before we played note. Remember: `playNote` automatically advances playhead forward by given number of beats, so need to rewind to get us back where we started. Effect of calling rewind is instantaneous; all it does is reposition playhead.

```
rewind(beats)
```

Calling `rewind` is important because want all 3 notes to play at exactly same time to give us right effect. Later in chap, experiment with playing notes that don't all get triggered at same time.

Lines 4–6 are all part of `for` loop – they all get repeated 3 times in a row because they're indented below loop statement on line 3. Finish function outside of loop with line 7 that calls another TunePad command called `fastForward`.

```
fastForward(beats)
```

As might guess, this does opposite of `reward` – instead of moving playhead backward it moves it forward. Call this last `fastForward` to make `eerieNote` behave exactly as if we had played a single note using our standard `playNote` function. Playhead will be moved by value of `beats`. This combination of using `rewind` inside of a loop with a `fastForward` at end of loop will be standard template for all of remaining effects that will cover in chap. Mix things up, but basic ideas will be same.

To put this all together, a childhood favorite that merrily reminisces about black plague. Sticking with horror movie theme, going to make this extra menacing by setting tempo way down to 60 BPM & playing whole thing an octave lower than usual. Snippet listed below defines entire song (including pitches, durations, & lyrics) in a Python list starting on line 1. Each note is represented by a Python structure called a `tuple`. Tuples are written as values separated by commas – just like lists except that you enclose values inside of parentheses instead of square brackets. Can use tuples exactly as we would use a list except values inside a tuple can't be changed. Saying same thing in Python lingo, tuples are *immutable* objects. Lines 18–19 step through notes in song, tuple by tuple, calling `eerieNote` for each one. Saying `note[0]` gets pitch of note & `note[1]` gets duration. Could also include this line inside loop if wanted to print lyrics.

```
print(note[3])


song = [
    (48, 1, "Ring"), (48, 0.5, "a-"), (45, 1, "round"), (50, 0.5, "the"),
    (48, 1.5, "ro-"), (45, 1, "sie, "), (47, 0.5, "a"), (48, 1, "pock-"),
    (48, 0.5, "et"), (45, 1, "full"), (50, 0.5, "of"), (48, 1.5, "po-"),
    (45, 1.5, "sies!"), (48, 1.5, "Ash-"), (45, 1.5, "es!"), (48, 1.5, "Ash-"),
    (45, 1, "es!"), (45, 0.5, "We"), (48, 1.5, "all"), (48, 1.5, "fall"),
    (41, 2.5, "down.")
]

def eerieNote(note, beats=1, velocity=100):
    volume = velocity
    for i in range(0, 3):
        offset = (random()-0.5) # random detune amount
        playNote(note + offset, beats, velocity=volume)
        rewind(beats)
    fastForward(beats)

for note in song:
    eerieNote(note[0] -12, beats=note[1])
```

Note values & associated TunePad commands for entire song: [Table: Note: Beats: Lyrics: Code]

- 8.2. Phaser effect. If play song in prev example, might hear some interesting & unexpected effects, especially on lower notes that come about from playing several notes together with very similar pitches. It turns out: this approximates a common musical effect called a *phaser* or *phase shifters*. Real phaser effects are produced by playing multiple versions of same sound together at same time, but changing frequency profile of each individual sound to get gaps or dips in spectrum.

  A very simple variation of `eerieNote` function replaces use of `random` function & instead just increments pitch `offset` variable by a fixed amount. This change gives us a cool approximation of a phaser effect that's easy to manipulate by changing number of simultaneous notes that get played or by changing pitch offset. Try this effect with some of built-in TunePad instruments, or use Sampler instrument to record & playback your own voice with `phaserNote` function.

```
def phaserNote(note, beats = 1, velocity = 100):
    note_count = 5
    volume = velocity / note_count
    offset = 0.0
    for i in range(0, note_count):
        playNote(note + offset, beats = beats, velocity = volume)
        offset += 0.15
        rewind(beats)
    fastForward(beats)
```

Can try this example online by going to https://tunepad.com/examples/phaser.

- 8.3. Echo effect. For next set of examples, going to move from pitch-based effects to time-based effects where spread multiple notes out over time. Simplest version of this is an *echo effect* that plays an initial sound at full volume followed by a rapid succession of softer notes that fall off into silence. This is an extremely common technique used in a wide variety of digitally

produced music. Can also throw in pitch manipulations & reverb effects to add another layer of complexity, but start with foundational repeated sound. As before, going to define our own function that looks similar to basic `playNote` function. Can follow along in TunePad by visiting https://tunepad.com/examples/echo.

```
def echoNote(note, beats = 1, delay = 0.125):
    volume = 100 # start at full volume
    offset = 0 # keep track of delays
    while volume > 1: # loop until silence
        playNote(note, beats = beats, velocity = volume)
        remind(beats - delay)
        volume *= 0.5 # reduce volume by 50%
        offset += delay # keep track of delays
    rewind(offset) # rewind by accumlated delay amount
    fastForward(beats) # move playhead to end of note
```

Line 1 looks similar to prev 2 examples except that we have added another optional parameter called `delay` that specifies how spread out each echoed note is in time. By default have set this to 0.125 beats, but by making this an optional parameter, have given composer ability to change this delay time on fly if they want to.

Line 2 should also look familiar except this time we are going to start with 1st note at full volume & then rapidly decay volume for each successive note. Then on line 3, define a variable called `offset` that keeps track of total amount of delay accumulated so far. This is a bookkeeping variable that's important for us to leave playhead in correct location after our function completes. Use this on line 9.

On line 4, set up a new kind of Python loop called a `while` loop. Until now, have always used `for` loops to iterate through a list or repeat sth for a *fixed* number of times. With a `while` loop on other hand, will repeat sth over & over again *until* a certain condition is met – in this case loop will repeat until volume is ≤ 1.

```
while volume > 1:
```

Basic idea: repeat sound until it's too quiet to hear. Trick to making this work: decrease value of `volume` inside while loop. Otherwise, loop would keep repeating over & over again forever (an infinite loop). Do this on line 7 where multiply volume by 0.5 (50% of its prev value). Could try other values here or even make this a parameter of function instead of a hard-coded value. If want sth to echo out longer, could set this up to 0.6 or even 0.7. Again, be careful here because if this value is 1.0 or higher note will echo forever & TunePad will complain that have created an infinite loop. Higher values can also cause note to echo out too long & run into other notes, creating unwanted interference.

Line 6 is a little tricky. Instead of rewinding all way back to beginning of note as we did in prev example, going to rewind by a smaller amount so that successive notes are spread out in time.

```
rewind(beats - delay)
```

This total amount = `delay` parameter that gets passed into function. Diagram below shows what this looks like over 4 iterations of `while` loop. Can also see value of `volume` as it decreases on each successive pass.

Last 2 lines of function are responsible for fixing up playhead position so that it's exactly at end of 1st note we played, which is why we kept track of number of total accumulated delay.

- 8.4. Chorus effect. A closely related effect to echo: randomly scatter notes around a central point in time. Our strategy for this effect should look familiar by now. Only tricky part: have to generate a random time offset ± a 32nd note that we use to fast forward. Because this number can be positive, negative, or 0, playhead will move forward, backward, or stay in place when call `fastForward`. Another way to think of it: `fastForward` with a negative number is same thing as calling `rewind` with a positive number. This function takes an optional parameter called `count` that says how many scattered notes we should play.

```
def chorusNote(note, beats = 1, count = 4):
    volume = 40
    spread = 0.125
    for i in range(0, count):
        offset = (random()-0.5) * spread
        fastForward(offset)
        playNote(note, beats - offset, volume)
        rewind(beats)
    fastForward(beats)
```

Then, when rewind on line 8, going to back up to where we started. Together this has effect of scattering several notes randomly around time the note was to be played. Can also try making spread an additional optional parameter that you pass into function. A simple stomp/clap pattern that uses `chorusNote`:

```
stomp = [ 0, 1 ]
clap = 22
chorusNote(stomp)
chorusNote(stomp)
chorusNote(clap)
rest(1)
```

This is a great effect to combine with a reverb effect to make it sound like you have a crowd in a large room. Another common technique: detune pitch of each note slightly from 1 another, which was a common technique for chorus effects applied to electric guitars in 1980s. Can try this function by going to https://tunepad.com/examples/chorus.

- 8.5. Arpeggiation effect. (Hiệu ứng rải âm) Another extremely common effect called arpeggiation is used to transform single sustained notes into rhythmic patterns that rapidly climb up & down a chord structure. Saw this in Interlude 4 as a method of playing chords. Arpeggios are commonly used across a wide variety of musical genres from classical to hip hop to dance music, & most digital audio workstations (DAWs) provide built-in arpeggiators with a variety of options to change speed, direction, & note pattern. Following pattern we saw with prev functions, can build an arpeggiator with TunePad by creating an `ARPNote` function with parameters for note pattern & speed. This effect combines both pitch-based & timing-based variations of notes.

```
def ARPNote(note, beats = 1, pattern = [0,4,7], speed = 0.125):
    offset = 0
    while offset < beats:
        for step in pattern:
            playNote(note + step, speed)
            offset += speed
    rewind(offset)
    fastForward(beats)
```

Before put this together in a complete example, look at `pattern` parameter. this parameter is a list that describes a pitch offset from 1st note (given by `note` parameter). Function will step through this list, & each element will be added to `note` to calculate which note to play. `pattern` parameter can be as simple or complex as you want, but a good starting point are some of chords introduced in Chap. 3. Some common arpeggiation patterns:

```
Major Triad [ 0, 4, 7 ]
Minor Triad [ 0, 3, 7 ]
Major Triad Up/Down [ 0, 4, 7, 12, 7, 4 ]
Minor Triad Up/Down [ 0, 3, 7, 12, 7, 3 ]
```

This function is a great example of using *nested* loops in Python, which just means: 1 loop is embedded inside of another. Outside loop gets set up on line 3 & repeats until variable `offset` > duration of note. Inner loop is set up on line 4 & steps through each value in `pattern` parameter. What all this means: inner for loop will get run at least once & possibly many times – as long as it takes to completely fill duration of note provided by `beats` parameter. Line 5 plays notes in arpeggio by adding `step` to base note s.t. it moves up or down pattern, & set duration of note to speed. Then on line 6, increment `offset` bookkeeping variable by duration of an arpeggiation step. Last 2 lines of function clean up timing to make sure playhead advances by correct amount.

`speed` parameter is a matter of taste & musical convention, but common to use 16th notes (0.25 beats) or 32nd notes (0.125 beats) for this value. 1 thing you might notice: this function will repeat arpeggiation chord pattern for as long as it takes to fill out entire note duration, but it might also play out beyond end of note. This extra holdover could be undesirable depending on music we're trying to make. To fix this, can insert 1 additional line right after line 6 inside for loop.

```
if offset >= beats: break
```

This has effect of exiting out of loop as soon as hit desired length, possibly short circuiting ARP pattern. Before look at an example of `ARPNote` function in action, add 1 more feature. It sounds a little heavy to have every note of arpeggio hit with same velocity. To add a more interesting rhythm & texture, could instead emphasize 1st note of arpeggio & then drop volume down for remaining notes. A variation of `ARPNote` function that creates this effect with sth like technique we used in `echoNote` example.

```
def ARPNote(note, beats = 1, velocity = 100, pattern = [0,4,7], speed = 0.125):
    offset = 0
    while offset < beats:
        volume = velocity # reset volume every iteration
        for step in pattern:
            playNote(note + step, speed, volume)
```

```
            volume = velocity * 0.25 # reduce volume after 1st note
            offset += speed
            if offset >= beats: break
        rewind(offset)
        fastForward(beats)
```

What's happening: create a `volume` variable that gets reset every iteration of while loop to full value of `velocity` parameter. Then, after play 1st note of arpeggio inside for loop, reduce `volume` to a quarter of `velocity` for remaining notes in arpeggio. When arpeggio is finished, `volume` is restored to its original value for next arpeggio.

This project puts everything together with an example from The Police. Try this code at https://tunepad.com/examples/arp.

When try this online, might notice: have added 1 more trick to `ARPNote` function on line 5. This uses `sustain` parameter of `playNote` to have 1st note of arpeggio ring out for entire duration of set, but at a reduced volume. This gives arpeggio a nice resonant quality.

- 8.6. Swing effect. Last effect in this chap creates a swing beat from a standard straight rhythm. Essentially this means adding a bounce or swing to a rhythm's timing so that we move from mechanical precision to more of a human feel. Most DAWs provide a variety of options for swinging a beat. Typically this involves altering a beat at either 9th or 16th note level so that odd-numbered notes are stretched out slightly in time, while even-numbered notes are compressed by same amount. Most DAWs let you select ratio between even- & odd-numbered notes by either offering a selection of fixed choices or a continuous dial. Creating sth similar in TunePad turns out to be fairly straightforward. Hardest part is figuring out whether we're on an even or an odd beat. To do this, can use built-in TunePad function called `getPlayHead()` that returns current position of playhead in beats (quarter notes). To figure out which 8th note we're on, can divide by an 8th note duration (0.5 on line 3). use Python's `round` function to make sure this is an integer value. Next step: ask whether we're on an even or an odd 8th note. Do that in line 4 using modulus operator `%`. Can think of this as returning remainder of a division operation. I.e., divide step variable by 2. If remainder is 1, it's an even 8th note, & need to push note forward a little by swing factor that we defined on line 2. Otherwise, it's an odd 8th note, & leave it in place. Together this has same effect as stretching out duration of odd-numbered notes.

```
def swingNote(note, beats=1, velocity=100):
    swing = 0.25
    step = round(getPlayhead() / 0.5)
    if step % 2 == 1:
        fastForward(beats * swing)
        playNote(note, beats, velocity)
        rewind(beats * swing)
    else:
        playNote(note, beats, velocity)
```

This example codes a simple rock beat that uses `swingNote`. Try this online at https://tunepad.com/examples/swing. Try adjusting value of swing factor on line 2 to get a feel for how it adds bounce to rhythm. Set swing down to 0.0 to get a straight beat.

- Interlude 8: How to Make a Drum Fill. In this interlude, explore 4 kinds of drum fills. A *drum fill* is a short phrase dropped into main groove of a drum track, usually every 9 or 16 bars. Fills add variety & support transition between sects of a song (e.g., verse to chorus). Can follow along online with this TunePad project https://tunepad.com/interlude/drum-fills.

  - **Step 1: Groove.** (Rãnh) Start by creating a 4-beat *groove*. For this tutorial, use a sparse rock-style drum pattern. Keep it simple because going to add decoration with various fills. Code below wraps drum pattern inside a function def so that it's easy to reuse in examples below. Define *groove* function inside of a *Code* cell in TunePad so that can import it into other cells.

```
def groove():
    playNote(0, beats = 0.5)
    playNote(4, beats = 0.5)
    playNote(2, beats = 0.5)
    playNote(4, beats = 0.5)
    playNote(0, beats = 0.5)
    playNote([0, 4], beats = 0.5) # double kick with hat
    playNote(2, beats = 0.5)
    playNote(4, beats = 0.5)
```

Set title of your code cell to "groove" & then note Python import statement that it generates for us directly below title Fig. 8.2: Code cell in TunePad showing import statement.

- **Pattern A: Tom runs.** Start with easiest drum fill. This fill takes up 1 measure (4 beats) & consists of 16th note runs on high tom, mid tom, low tom, & kick drum in order Fig. 8.3: Drum fill pattern A.

  Can code this with simple for loops to play each of drum sounds:

```
from groove import * # import our groove function
def fillA():
    for i in range(4):
        playNote(6, beats = 0.25) # high tom
    for i in range(4):
        playNote(7, beats = 0.25) # mid tom
    for i in range(4):
        playNote(8, beats = 0.25) # low tom
    for i in range(4):
        playNote(0, beats = 0.25) # kick drum
    playNote(9, beats = 0, sustain = 4) # crash cymbal
```

  Fill finishes with a crash cymbal hit on line 16. Use `sustain` (duy trì) parameter to let crash cymbal ring out & overlap with next drum measure. Code to play fill:

```
# let's try it!
groove()
groove()
fillA()
groove()
```

- **Pattern B: Triplets.** Our next pattern plays triplet notes that subivide each beat into 3 equal parts. Pattern repeats 4times in a row with HIGH TOM - LOW TOM - KICK DRUM combos Fig. 8.4: Drum fill pattern B.

  Can code this with just 1 for loop where set beat duration to 1/3.0.

```
from groove import *
def fillB():
    for i in range(4):
        playNote(6, beats = 1/3.0)
        playNote(8, beats = 1/3.0)
        playNote(0, beats = 1/3.0)
    playNote(9, beats = 0, sustain = 4)
```

  Use this code to play `fillB` with groove.

```
groove()
groove()
fillB()
groove()
```

- **Pattern C: Random 16ths.** Next pattern sprinkles random 16th note hits in to decorate an otherwise boring pattern.
  – Mẫu tiếp theo rắc ngẫu nhiên nốt nhạc móc đơn 16 để trang trí cho 1 mẫu nhàm chán.

  To do this, going to use Python's `choice` function which is part of built-in `random` module. This function works by picking an element from a list at random. Can think of it as drawing a random card from a deck. To use `choice`, have to add a line at top of our code to import function:

```
from random import choice
```

  Next secret ingredient: define a list of possible drum sounds that will get selected at random. 1 trick: pad this list with a few `None` values, which will result in random empty spaces in our pattern. Can experiment with different drum sounds in this list or with using a larger or smaller number of `Nones`. Might also notice: doubled up on number of 10 notes (claps) because like how they sound.

```
notes = [ 0, 2, 3, 4, 6, 7, 8, 10, 10, 11, None, None, None, None ]
```

  Once have defined our note array, function is simple to write. Just select & play 8 notes from our list at random.

```
from random import choice
from groove import *
notes = [ 0, 2, 3, 4, 6, 7, 8, 10, 10, 11, None, None, None, None ]
```

```
def fillC():
    for i in range(8):
        playNote(choice(notes), 0.25)
    rewind(2)

# play it with our groove
for i in range(10):
    groove()
    fillC()
    groove()
```

1 trick on line 9 where rewind playhead by 2 beats. Do this so that drum fill overlaps with beat instead of pausing it while fill plays.

- **Pattern D: Random triplets.** Our last pattern is a combination of Pattern B & C. use triplets but with random notes instead of a fixed pattern. Also sue `choice` function again, but with a smaller set of notes:

```
notes = [ 0, 6, 7, 8, 10 ]
```

Then can play 6 triplets (2 beats) at random followed by a crash.

```
def fillD():
    notes = [ 0, 6, 7, 8, 10 ]
    for i in range(6):
        playNote(choice(notes), 1/3.0)
    playNote(9, beats = 0, sustain = 4)
```

Code to play this full with our groove.

```
for i in range(10):
    groove()
    groove()
    fillD()
    groove()
```

- 9. Song composition & EarSketch. [Guest chap by Lauren McCall] Art of sampling & remixing has become a central practice in modern digital music production. Artists from Dr. Dre to Cardi B have used new technologies to blend short samples from existing music into entirely new compositions. This chap introduces sampling, remixing, & song composition using a free online platform called EarSketch. This platform is similar to TunePad in that it combines Python programming with music creation. But with EarSketch focus is more on creating full-length songs by combining & remixing pre-recorded samples instead of composing music from individual notes & percussion sounds. Cover basics of working with EarSketch & show how to structure musical samples into full-length compositions.

  - 9.1. Song structure. There's more to writing music than coming up with a beat, harmony, & melody. Nearly every popular song follows some kind of codified structure. In days before recorded audio, structure & repetition helped listeners develop relationships with music. Material would get introduced early in a song & then elaborated throughout. Structure of a Blues song is an excellent sample, but this is true across almost every genre of popular music from punk to hip-hop to country. Structure & repetition is equally important to music we listen to today in that it gives us an opportunity to notice contrasting elements, remember a melody, or sing along with chorus.

    Song structure – sometimes known as *musical form* – describes how musical ideas & material play out over a piece of music. Earlier in book introduced idea of musical phrase: a self-contained musical thought, like a sentence. Can group these phrases into larger musical sects. These sects can repeat to form even larger structures Fig. 9.1: Songs are composed of nested & repeating notes, phrases, & sects. Notes ⊂ Phrase ⊂ Section ⊂ Song.

    Most common way for musicians to clarify & discuss different parts of a song: label them with letters. Every time there's a new sect, it gets next letter in alphabet. If a sect reappears, reuse letter we had already applied to that sect. If a sect reappears but is slightly different – maybe with different lyrics – can give it a number. This works as a timeline or map of a song. In popular music, *verse-chorus form* is most common song structure. At its simplest, this form is built on 2 repeating sect called `verse, chorus`. There are endless variations on this basic structure, but form tends to include: Intro → A1 Verse → B Chorus → A2 Verse → B Chorus → C bridge → B Chorus → Outro.

    Some songs will feature an introduction that builds up energy & introduces musical material. Verse is a repeated sect that helps tell story of a song. Generally, melody of verse is same each time, but lyrics are different. Chorus, or hook, usually contains an important musical or lyrical motif to song. This is usually most memorable part of a song, & lyrics are almost always same for each repetition. Bridge provides contrast with rest of song, often using sth special like a change in harmony

or tempo. Not every song has a bridge; some have another repetition of verse. Final sect, outro, is a way to finish song. This might include sth like a slow fade out of chorus.

Take a song like Carrie Underwood's hit single, *Before He Cheats* (2005). This follows exactly structure in table above. There's a short intro followed by 1st verse ("Right now, he's probably slow dancin' ..."), followed by chorus ("I dug my key into side of his pretty little souped-up 4-wheel drive ..."). A 2nd verse & chorus is then followed by a bridge ("I might have saved a little trouble for the next girl"), 1 last chorus, & outro ("Oh, maybe next time he'll think before he cheats"). Like many of concepts discussed in this book, these rules are more of rough guidelines than hard rules. Being aware of song form can make writing music much easier, & can thin in larger sects. Using variations of common forms gives listeners an entry point as to what to expect from your music.

- 9.2. Sampling. Sampling involves taking sects from existing pieces of music & repurposing or remixing them in order to make a new creation. A sample might consist of a specific musical element of original song e.g. a bassline, drum break, vocals, or even a snippet of speech. Samples often have creative effects applied, e.g. reverb, chorus, or filters, & they might be looped, pitch shifted, or even reversed. Examples includes songs like Kayne West's *Good Morning* (2007), which sampled Elton John's *Someone Saved My Life Tonight* (1975) or *Naughty* By Nature's O.P.P. (1991), which sampled multiple elements from Jackson 5's *ABC* (1970). Sampling often occurs in dialogue with or homage to original work. An artist might include a reference to a work that inspired them by incorporating signature chord progressions, instrumentation, or melodies.

  Availability of new recording technology in 20th century gave artists opportunity to engage with prior generations of music in innovative ways. Although sampling is common to many genres of music, it's 1 of defining characteristics of hip-hop music. Roots of hip-hop were in live performance – DJs manipulated existing records on turntables to create completely new soundscapes. Techniques from live performance then carried over into original works driven by advances in recording technology & equipment specifically designed for sampling. Advanced sampling tools are now built into digital audio workstation (DAW) software in which vast majority of modern music is created.

  When combining samples in your composition, be mindful of how different musical elements work with 1 another. If your song is in E major & add a sample that's in D minor, it might clash harmonically. If sample is in a different tempo, it might not line up rhythmically with rest of your song. Samples get manipulated in pitch or tempo or a variety of other ways to make them work stylistically with rest of a song.

  Remark 2 (Copyright law)**.** *Be aware of copyright issues around music that you sample, especially if you're thinking about sharing it online or licensing it to make money. Most of music on TunePad & EarSketch websites is licensed so that you can use them however you want, but important to get permission or a license when sampling other artists' music.*

- 9.3. Introduction to EarSketch. EarSketch Fig. 9.2: Main EarSketch interface features a large library of samples (left), an interactive timeline (middle top), a code editor (center), & extensive documentation & curriculum (right). is a free online platform for creating music with code that was developed by researchers at Georgia Institute of Technology in Atlanta. EarSketch works with a few different programming languages including Python & JavaScript. All of Python concepts you've been learning about (lists, loops, variables, functions, & parameters) still apply, although music-making functions are different. E.g., EarSketch doesn't have a `playNote` or `rest` function. Instead, it has a `fitMedia` function that places musical samples on timeline of a DAW.

  Remark 3 (JavaScript)**.** *JavaScript is another important & widely used programming language. It's often called* language of web *because web sites use JavaScript to add logic, user interaction, & dynamic effects. Every time click on a button on a web page, it's almost certainly running JavaScript code. If were to look under hood at TunePad & EarSketch, you would see: they both use JavaScript to generate & play music. If interested in trying it out, EarSketch has excellent resources for learning JavaScript.*

  Once log into main EarSketch site at https://earsketch.gatech.edu, can write programs in *Code Editor*. Note: both EarSketch & TunePad sometimes use term *script* to refer to a program. A script is just another word for a short computer program that performs smaller tasks like putting together a song. Creating a new script in EarSketch generates a small amount of *boilerplate* code to set up your project. In computer programming "boilerplate" means standard code that you use across multiple projects. EarSketch boilerplate:

```
from earsketch import *
init()
setTempo(120)
finish()
```

  1st line imports core EarSketch module. In TunePad have used Python's import functionality as well, but can read more about how it works in Python appendix at end of book. 2nd line `init()`, sets up DAW. Next, `setTempo` specifies project's tempo in beats per minute (bpm). Code you write for your project should be added in between `setTempo, finish()` function calls.

  1 of best parts about EarSketch is its extensive sound library. This library has nearly 4000 premade audio clips created by producers & musicians that you can use in your projects for free. Can browse through sample library on main EarSketch page, filtering by musical genre, artist, & instrument type. Can add samples to your project with code using their predefined variable names & `fitMedia` function.

  - 9.3.1. `fitMedia` function. p. 163+++

- 10. Modular synthesis. [Guest chap by Dillon Hal] Modular synthesis is a set of tools & techniques for electronically or digitally "synthesizing" musical sounds. *Modular* means "made up of smaller pieces that can be taken apart & rearranged". *Synthesis* means "creating sth new from existing parts or ideas". Modular synthesizers were 1st introduced in 1950s & 1960s with devices like Moog System 55 Fig. 10.1: Moog System 55 modular synthesizer., which were sometimes size of entire rooms. These synthesizers were *analog*, meaning they generated sounds using electrical circuits instead of starting with some kind of physical vibration (like a guitar string). By chaining together electronically generated waveforms with filters, effects, & envelopes, they could not only approximate traditional acoustic instruments – like bass, piano, brass, woodwinds, & drums – but also create entirely new sounds altogether. Today, can make these same sounds on a computer using *digital signal processing* techniques. While process has changed, basic ideas are same: take several different *modules* & combine them together into a *patch*. A module is a single device with a single purpose e.g. creating, transforming, or controlling sound. A patch is formed by chaining many modules together to combine sounds & layer effects. This chap provides a high-level overview of modular synthesis concepts using an interface built into TunePad.
  - 10.1. Signals. In modular synthesis, a *signal* is a collection of values that varies over time & can carry information. In physical world, when sound waves collide with membrane of a microphone, they cause fluctuations in voltage levels of an electric circuit. Microphone has transformed physical sound waves into an electric signal that consists of fluctuating voltage levels. These recorded voltage levels are just 1 example of a signal Fig. 10.2: Audio signal from a microphone & an audio signal from an electric circuit.

    – Tín hiệu. Trong tổng hợp mô-đun, *tín hiệu* là tập hợp các giá trị thay đổi theo thời gian & có thể mang thông tin. Trong thế giới vật lý, khi sóng âm va chạm với màng của micrô, chúng gây ra sự dao động ở mức điện áp của mạch điện. Micrô đã biến đổi sóng âm vật lý thành tín hiệu điện bao gồm các mức điện áp dao động. Các mức điện áp được ghi lại này chỉ là 1 ví dụ về tín hiệu Hình 10.2: Tín hiệu âm thanh từ micrô & tín hiệu âm thanh từ mạch điện.

    For analog synthesizers, this process works in reverse. Audio signals are generated synthetically using simple electronic components – e.g. resistors, capacitors, & transistors – rather than a microphone. By varying voltage input levels, can change frequency at which circuit oscillates or "vibrates". In digital world, instead of using oscillator circuits, computers generate streams of numbers that simulate voltage output of original electronic components.

    – Đối với bộ tổng hợp tương tự, quá trình này hoạt động ngược lại. Tín hiệu âm thanh được tạo ra tổng hợp bằng cách sử dụng các thành phần điện tử đơn giản – ví dụ như điện trở, tụ điện, & bóng bán dẫn – thay vì micrô. Bằng cách thay đổi mức điện áp đầu vào, có thể thay đổi tần số mà mạch dao động hoặc "rung". Trong thế giới kỹ thuật số, thay vì sử dụng mạch dao động, máy tính tạo ra các luồng số mô phỏng điện áp đầu ra của các thành phần điện tử gốc.

    For modular synthesis, there are 2 basic kinds of signals: *audio signals & control signals*. Audio signals oscillate in range of human hearing, i.e., they can create musical notes. Faster they move, higher their *frequency* & higher pitch we hear. Control signals, on other hand, tend to vibrate more slowly, below range of human hearing. Instead of sending them directly to our speakers, use them to change parameters of audio signals. A good example of this is *vibrato*. A violinist draws a bow across a string to generate a high-frequency sound that falls in range of human hearing. As bow is drawn, violinist quickly rocks a finger on fretboard to modulate pitch. Violinist's finger operates like a control signal that modifies audio signal of violin string.

    – Đối với tổng hợp mô-đun, có 2 loại tín hiệu cơ bản: *tín hiệu âm thanh & tín hiệu điều khiển*. Tín hiệu âm thanh dao động trong phạm vi thính giác của con người, tức là chúng có thể tạo ra các nốt nhạc. Chúng di chuyển càng nhanh, *tần số* của chúng & cao độ mà chúng ta nghe được càng cao. Mặt khác, tín hiệu điều khiển có xu hướng rung chậm hơn, thấp hơn phạm vi thính giác của con người. Thay vì gửi chúng trực tiếp đến loa của chúng ta, hãy sử dụng chúng để thay đổi các thông số của tín hiệu âm thanh. 1 ví dụ điển hình về điều này là *rung*. 1 nghệ sĩ vĩ cầm kéo 1 cây vĩ trên 1 dây đàn để tạo ra âm thanh tần số cao nằm trong phạm vi thính giác của con người. Khi kéo vĩ, nghệ sĩ vĩ cầm nhanh chóng lắc ngón tay trên cần đàn để điều chỉnh cao độ. Ngón tay của nghệ sĩ vĩ cầm hoạt động giống như 1 tín hiệu điều khiển để điều chỉnh tín hiệu âm thanh của dây đàn vĩ cầm.
  - 10.2. Modules. 1 way to think about modules is as physical pieces of equipment with 1 or more sockets for inputs & an output signal. Inputs may be audio signals or control signals. Outputs may feed into other modules or be sent to our speakers. In simple TunePad patch below, sine wave module generates an audio signal that gets plugged into Output module & sent to speakers Fig. 10.3: A simple Modular Synthesis patch created in TunePad.
    - 10.2.1. Source modules. A source module is a type of module that generates a signal. Sources can also have inputs to control parameters like pitch & amplitude.
      – Mô-đun nguồn. Mô-đun nguồn là loại mô-đun tạo ra tín hiệu. Các nguồn cũng có thể có đầu vào để điều khiển các thông số như cao độ & biên độ.
      p. 180+++
    - 10.2.2. Control modules. Control modules generate signals that aren't typically audible to humans. Control modules can feed into input sockets of other modules.
    - 10.2.3. Creating patches.
- 11. History of music & computing. [Guest chap by Jason Freeman] Throughout this book, you've seen that music & computing are connected. Musicians think about things like notes, meter, & phrases just like programmers think about things like variables, functions, & loops. Code can help us understand inner logic of how music is created. Writing music with code can help us create interesting music & express ourselves in new ways.

These connections between music & computing are no accident. Since earliest computers, musicians have been inventing ways to use computers in music. Today, when listen to your favorite song, computers have likely been involved in many different ways. E.g.:

- Songwriter may have used music notation or audio recording software to capture their initial inspiration for song on a laptop, tablet, or mobile phone.
- Musicians in band may have performed on digital keyboards, drum machines, or control surfaces. Guitarists & bass players may have routed their audio through digital effects pedals. Each of these devices makes or transforms sound through an embedded computer.
- In recording studio, a producer & audio engineer probably recorded, edited, & mixed song using digital audio workstation software on a computer. They may have even used additional software plugins for special tasks like eliminating background noise or fixing notes that were out of tune.
- A music streaming service had to store music track & information about it in a cloud-based database. Service also developed an app to stream track to your device. In addition, service probably developed ML algorithms to generate personalized playlists or radio stations for you.

In this chap, explore some of key moments in history of computer music. This history will help us understand how computing is continuing to revolutionize creation, performance, & distribution of music today.

- 11.1. What is computer music? Computer music refers to any process of creating, performing, analyzing, or distributing music that involves a computer. This includes a wide range of activities, from writing code to creating music (like in EarSketch & TunePad) to producing new music with an app to watching your favorite music videos online.

  Because computer music is such a broad field, it helps to ask a few basic questions as we look at developments in computer music history or think about things we want to do with computers ourselves:

  What is role of computer? Often think of computers as tools that help us get a job done, like writing a paper or reading a message. Computers can be musical tools too. They can be musical instruments that create sound in response to our actions, like a digital keyboard that plays a note every time we press a key. Most of instruments in TunePad, e.g., work in this way. Press keys on computer keyboard & immediately hear corresponding instrument sound from TunePad.

  Computers can also act more like a musician who performs a piece, taking each note in a score & turning it into sound. Much of code we write in TunePad works this way. Specify a series of notes to be played. Then TunePad displays notes in a grid & plays them back for us. Each instrument in a TunePad dropbook acts like a single musician. When play them all back together, computer becomes like a band of musicians.

  Many computer musicians also explore how computers can act like intelligent musicians. These intelligent musicians don't simply follow instructions in a score. An artificially intelligent computer composer may create its own musical scores that mimic style of a human musician. An artificially intelligent computer performer may listen to human musicians & improvise along with them.

  How does computer represent music? Human musicians have many different ways of representing music: create notated scores, create shorthand lead sheets, talk about it in words, & teach it by oral tradition. Computer musicians also have many different ways of representing music. 2 approaches are most common. Symbolic representations describe music as a series of musical notes, each with a specific pitch, loudness, start time, & duration. Audio representations describe music in terms of actual sound that we hear.

  Have already seen these 2 approaches with TunePad & EarSketch. TunePad's `playNote()` function, e.g., is a symbolic approach that defines properties of a single musical note. Music in TunePad consists of a bunch of these notes defined by `playNote()` & other TunePad functions.

  In contrast, EarSketch's `fitMedia()` function adds an audio file into your song. That audio file can be anything: it can be a single musical note, it can be an entire musical phrase, or it can be a sound effect that might not even seem like music. A song in EarSketch is made up of a bunch of these audio files placed on a multi-track timeline.

  There are advantages & disadvantages to each approach. With a symbolic representation, computer can easily represent & manipulate each individual note. TunePad can easily draw music on a grid that shows each note, & if want to change a single note in your song, easy to find corresponding line of code & update it. With an audio representation, those kinds of visualizations & edits are often not possible. But there is more flexibility to capture, generate, & edit any kind of sound.

  Does computer create music in real time? Early computers often took hours to generate a few secs of sound, & even today, computer music applications that use ML & AI can take days to analyze large datasets. Many computer music applications, like musical instruments, require immediate, real-time responses, while with other applications, like composition, real-time operation may not always be as important.

  Consider these 3 questions as look at some of earliest examples of computer music.

- 11.2. Computer music on mainframe computers. In 1950s, Bell Labs in New Jersey was 1 of most famous research & development labs in world. Building from legacy of Alexander Graham Bell's invention of telephone, this division of AT&T had created innovative new technologies that sill impact how transmit radio signals, synthesize speech, encrypt communications, & build computers.

  Max Matthews was an electrical engineer & violinist working at Bell Labs. He was working with giant mainframe computers like IBM 7094, which cost millions of dollars, occupied an entire room, & relied on punch cards & magnetic tapes to read &

write data. These computers were used for many purposes, including during some of early NASA space missions Fig. 11.1: IBM 7094 computer at NASA. Public domain. Available at Wikipedia.

Matthews wanted to make music with these mainframe computers. He created a programming language called MUSIC, then later MUSIC II, MUSIC III, & so on. Today call his languages MUSIC-N (N is like a variable representing specific version number of MUSIC).

In MUSIC-N languages, programmers created an "orchestra" of musical instruments. Orchestra defined a set of instruments & configured how each instrument created sound. An example of an instrument definition in CSound, a music programming language inspired by MUSIC-N that can still use today: [Codes]

Even though syntax of programming language may look strange, elements are actually familiar. 1st line defines instrument, which is similar to defining your own function in TunePad or EarSketch. 3rd line outputs sound, which is similar to a return statement in your EarSketch or TunePad function def. 2nd line calls a unit generator (oscil), which is similar to calling a function, & passes 3 arguments to it: `p4, p5, 1`. Oscil is an oscillator that synthesizes a simple waveform, like a sine wave or square wave. Arguments configure things like how low or high sound is (frequency) & how loud it is (amplitude).

– Mặc dù cú pháp của ngôn ngữ lập trình có vẻ lạ, nhưng các phần tử thực sự quen thuộc. Dòng thứ nhất định nghĩa nhạc cụ, tương tự như định nghĩa hàm của riêng bạn trong TunePad hoặc EarSketch. Dòng thứ ba xuất ra âm thanh, tương tự như câu lệnh return trong hàm EarSketch hoặc TunePad của bạn def. Dòng thứ hai gọi 1 trình tạo đơn vị (oscil), tương tự như gọi 1 hàm, & truyền 3 đối số cho nó: `p4, p5, 1`. Oscil là 1 bộ dao động tổng hợp 1 dạng sóng đơn giản, như sóng sin hoặc sóng vuông. Các đối số cấu hình những thứ như âm thanh thấp hay cao (tần số) & âm lượng lớn như thế nào (biên độ).

p. 197+++

- 11.3. Digital synthesizers & personal computers.

- Appendix A: Python ref.
- Appendix B: TunePad programming ref.
- Appendix C: Music ref.

1.4 [KD06]. Anssi Klapuri, Manuel Davy. Signal Processing Methods for Music Transcription. 2006

- Preface. Signal processing techniques, & information technology in general, have undergone several scientific advances which permit us to address very complex problem of automatic music transcription (AMT). During last 10 years, interest in AMT has increased rapidly, & time has come for a book-length overview of this subject.

  – Kỹ thuật xử lý tín hiệu, & công nghệ thông tin nói chung, đã trải qua nhiều tiến bộ khoa học, cho phép chúng ta giải quyết vấn đề rất phức tạp về phiên âm nhạc tự động (AMT). Trong 10 năm qua, sự quan tâm đến AMT đã tăng lên nhanh chóng, & đã đến lúc cần có 1 cuốn sách tổng quan về chủ đề này.

  Purpose of this book: present signal processing algorithms dedicated to various aspects of music transcription. AMT is a multifaceted problem, comprising several subtasks: rhythm analysis, multiple fundamental frequency analysis, sound source separation, musical instrument classification, & integration of all these into entire systems. AMT is, in addition, deeply rooted in fundamental signal processing, which this book also covers. As field is quite wide, have focused mainly on signal processing methods & Western polyphonic music. An extensive presentation of work in musicology & music perception is beyond scope of this book.

  – Mục đích của cuốn sách này: trình bày các thuật toán xử lý tín hiệu chuyên sâu cho nhiều khía cạnh khác nhau của việc phiên âm nhạc. AMT là 1 vấn đề đa diện, bao gồm nhiều nhiệm vụ phụ: phân tích nhịp điệu, phân tích đa tần số cơ bản, tách nguồn âm thanh, phân loại nhạc cụ, & tích hợp tất cả những yếu tố này vào toàn bộ hệ thống. AMT, ngoài ra, còn có nền tảng sâu sắc trong xử lý tín hiệu cơ bản, mà cuốn sách này cũng đề cập đến. Vì lĩnh vực này khá rộng, chúng tôi chủ yếu tập trung vào các phương pháp xử lý tín hiệu & âm nhạc đa âm phương Tây. Việc trình bày chi tiết về các công trình trong âm nhạc học & cảm thụ âm nhạc nằm ngoài phạm vi của cuốn sách này.

  This book is mainly intended for researchers & graduate students in signal processing, computer science, acoustics, & music. Hope book will make field easier to approach, providing a good starting point for newcomers, but also a comprehensive reference source for those already working in field. Book is also suitable for use as a textbook for advanced courses in music signal processing. Chaps are mostly self-contained, & readers may want to read them in any order or jump from 1 to another at will. Whenever an element from another chap is needed, an explicit reference is made to relevant chap. Chaps. 1–2 provide some background of AMT & signal processing for entire book, resp. Otherwise, only a basic knowledge of signal processing is assumed.

  – Cuốn sách này chủ yếu dành cho các nhà nghiên cứu & sinh viên sau đại học về xử lý tín hiệu, khoa học máy tính, âm học, & âm nhạc. Hy vọng cuốn sách sẽ giúp lĩnh vực này dễ tiếp cận hơn, cung cấp 1 điểm khởi đầu tốt cho người mới bắt đầu, nhưng cũng là 1 nguồn tham khảo toàn diện cho những người đã làm việc trong lĩnh vực này. Cuốn sách cũng phù hợp để sử dụng làm giáo trình cho các khóa học nâng cao về xử lý tín hiệu âm nhạc. Các chương hầu hết là độc lập, & người đọc có thể muốn đọc chúng theo bất kỳ thứ tự nào hoặc chuyển từ chương này sang chương khác tùy ý. Bất cứ khi nào cần 1 phần tử từ

chương khác, 1 tham chiếu rõ ràng sẽ được thực hiện đến chương liên quan. Các chương 1–2 cung cấp 1 số kiến thức cơ bản về AMT & xử lý tín hiệu cho toàn bộ cuốn sách. Ngoài ra, chỉ cần có kiến thức cơ bản về xử lý tín hiệu là được.

PART I: FOUNDATIONS.

- 1. Introduction to Music Transcription. Music transcription refers to analysis of an acoustic musical signal so as to write down pitch, onset time, duration, & source of each sound that occurs in it. In Western tradition, written music uses note symbols to indicate these parameters in a piece of music. Fig. 1.1: An acoustic musical signal & its time-frequency domain representation. Fig. 1.2: Musical notation corresponding to signal in Fig. 1.1. Upper staff lines show notation for pitched musical instruments & lower staff lines show notation for percussion instruments. show notation of an example music signal. Omitting details, main conventions are that time flows from left to right & pitch of notes is indicated by their vertical position on staff lines. In case of drums & percussions, vertical position indicates instrument & stroke type. Loudness (& applied instrument in case of pitched instruments) is normally not specified for individual notes but is determined for larger parts.

  – Phiên âm nhạc đề cập đến việc phân tích tín hiệu âm nhạc âm thanh để ghi lại cao độ, thời gian bắt đầu, thời lượng, & nguồn của mỗi âm thanh phát ra trong đó. Trong truyền thống phương Tây, âm nhạc viết sử dụng các ký hiệu nốt nhạc để chỉ các thông số này trong 1 bản nhạc. Hình 1.1: 1 tín hiệu âm nhạc âm thanh & biểu diễn miền thời gian-tần số của nó. Hình 1.2: Ký hiệu âm nhạc tương ứng với tín hiệu trong Hình 1.1. Các dòng khuông nhạc trên cùng hiển thị ký hiệu cho các nhạc cụ có cao độ & các dòng khuông nhạc dưới cùng hiển thị ký hiệu cho các nhạc cụ gõ. hiển thị ký hiệu của 1 tín hiệu âm nhạc ví dụ. Bỏ qua các chi tiết, các quy ước chính là thời gian chảy từ trái sang phải & cao độ của các nốt nhạc được chỉ ra bằng vị trí dọc của chúng trên các dòng khuông nhạc. Trong trường hợp trống & bộ gõ, vị trí dọc biểu thị nhạc cụ & loại nét. Độ to (& nhạc cụ được sử dụng trong trường hợp nhạc cụ có cao độ) thường không được chỉ định cho các nốt riêng lẻ nhưng được xác định cho các phần lớn hơn.

  Besides common musical notation, transcription can take many other forms, too. E.g., a guitar player may find it convenient to read chord symbols which characterize note combinations to be played in a more general manner. In a computational transcription system, a MIDI file [Musical Instrument Digital Interface (MIDI) is a standard for exchanging performance data & parameters between electronic musical devices] is often an appropriate format for musical notations Fig. 1.3: A 'piano-roll' illustration of a MIDI file which corresponds to pitched instruments in signal in Fig. 11. Different notes are arranged on vertical axis & time flows from left to right. Common to all these representations: they capture musically meaningful parameters that can be used in performing or synthesizing piece of music in question. From this point of view, music transcription can be seen as discovering 'recipe', or reverse-engineering 'source code' of a music signal.

  – Bên cạnh ký hiệu âm nhạc phổ biến, bản ghi chép cũng có thể có nhiều dạng khác. E.g., 1 người chơi guitar có thể thấy thuận tiện khi đọc các ký hiệu hợp âm đặc trưng cho các tổ hợp nốt nhạc được chơi theo cách tổng quát hơn. Trong hệ thống ghi chép tính toán, tệp MIDI [Giao diện kỹ thuật số nhạc cụ (MIDI) là 1 tiêu chuẩn để trao đổi dữ liệu biểu diễn & các tham số giữa các thiết bị âm nhạc điện tử] thường là định dạng phù hợp cho các ký hiệu âm nhạc Hình 1.3: Hình minh họa 'piano-roll' của tệp MIDI tương ứng với các nhạc cụ có cao độ trong tín hiệu ở Hình 11. Các nốt khác nhau được sắp xếp trên trục dọc & thời gian chảy từ trái sang phải. Điểm chung của tất cả các biểu diễn này: chúng nắm bắt các tham số có ý nghĩa về mặt âm nhạc có thể được sử dụng khi biểu diễn hoặc tổng hợp 1 bản nhạc đang được đề cập. Theo quan điểm này, bản ghi chép âm nhạc có thể được coi là khám phá ra 'công thức" hoặc kỹ thuật đảo ngược 'mã nguồn" của 1 tín hiệu âm nhạc.

  A complete transcription would require: pitch, timing, & instrument of all sound events be resolved. As this can be very hard or even theoretically impossible in some cases, goal is usually redefined as being either to notate as many of constituent sounds as possible (complete transcription) or to transcribe only some well-defined part of music signal, e.g. dominant melody or most prominent drum sounds (partial transcription). Both of these goals are relevant & are discussed in this book.

  – 1 bản chép nhạc hoàn chỉnh sẽ yêu cầu: cao độ, nhịp điệu, & nhạc cụ của tất cả các sự kiện âm thanh phải được giải quyết. Vì điều này có thể rất khó hoặc thậm chí là bất khả thi về mặt lý thuyết trong 1 số trường hợp, mục tiêu thường được định nghĩa lại là ghi chép càng nhiều âm thanh cấu thành càng tốt (bản chép nhạc hoàn chỉnh) hoặc chỉ chép 1 phần tín hiệu âm nhạc được xác định rõ ràng, ví dụ như giai điệu chủ đạo hoặc âm thanh trống nổi bật nhất (bản chép nhạc 1 phần). Cả hai mục tiêu này đều có liên quan & được thảo luận trong cuốn sách này.

  Music transcription is closely related to structured audio coding. A musical notation or a MIDI file is an extremely compact representation that retains characteristics of a piece of music to an important degree. Another related area of study: music perception [144]. Detecting & recognizing individual sounds in music is a big part of its perception, although it should be emphasized: musical notation is primarily designed to serve sound production & not to model hearing. Do not hear music in terms of note symbols but, as described by Bregman [49, pp. 457–460], music often 'fools' auditory system so that we perceive simultaneous sounds as a single entity.

  – Phiên âm nhạc có liên quan chặt chẽ đến mã hóa âm thanh có cấu trúc. Ký hiệu âm nhạc hoặc tệp MIDI là 1 biểu diễn cực kỳ cô đọng, giữ lại các đặc điểm của 1 bản nhạc ở mức độ đáng kể. 1 lĩnh vực nghiên cứu liên quan khác: nhận thức âm nhạc [144]. Việc phát hiện & nhận dạng các âm thanh riêng lẻ trong âm nhạc là 1 phần quan trọng trong nhận thức của nó, mặc dù cần nhấn mạnh rằng: ký hiệu âm nhạc chủ yếu được thiết kế để phục vụ cho việc tạo ra âm thanh & không phải để mô phỏng thính giác. Đừng nghe nhạc theo ký hiệu nốt nhạc, nhưng, như Bregman [49, tr. 457–460] mô tả, âm nhạc thường "đánh lừa" hệ thống thính giác để chúng ta cảm nhận các âm thanh đồng thời như 1 thực thể duy nhất.

  In addition to audio coding, applications of music transcription comprise.

  - Music information retrieval based on melody of a piece, e.g.

- Music processing, e.g. changing instrumentation, arrangement, or loudness of different parts before resynthesizing a piece from its score.
- Human-computer interaction in various applications, including score type-setting programs & musically oriented computer games. Singing transcription is of particular importance here.
- Music-related equipment, ranging from music-synchronous light effects to highly sophisticated interactive music systems which generate an accompaniment for a soloist.
- Musicological analysis of improvised & ethnic music for which musical notations do not exist.
- Transcription tools for amateur musicians who wish to play along with their favorite music.

– Ngoài mã hóa âm thanh, các ứng dụng của phiên âm nhạc bao gồm.

- Truy xuất thông tin âm nhạc dựa trên giai điệu của 1 bản nhạc, ví dụ:
- Xử lý âm nhạc, ví dụ: thay đổi nhạc cụ, cách sắp xếp hoặc âm lượng của các phần khác nhau trước khi tổng hợp lại 1 bản nhạc từ bản nhạc của nó.
- Tương tác giữa người & máy tính trong nhiều ứng dụng khác nhau, bao gồm các chương trình thiết lập kiểu bản nhạc & trò chơi máy tính định hướng âm nhạc. Phiên âm giọng hát có tầm quan trọng đặc biệt ở đây.
- Thiết bị liên quan đến âm nhạc, từ hiệu ứng ánh sáng đồng bộ với âm nhạc đến các hệ thống âm nhạc tương tác cực kỳ tinh vi tạo ra phần đệm cho nghệ sĩ độc tấu.
- Phân tích âm nhạc học về nhạc ứng tác & nhạc dân tộc mà không có ký hiệu âm nhạc.
- Công cụ phiên âm dành cho các nhạc sĩ nghiệp dư muốn chơi theo bản nhạc yêu thích của họ.

Purpose of this book: describe algorithms & models for different subtopics of music transcriptions, including pitch analysis, meter analysis, percussion transcription, musical instrument classification, & music structure analysis. Main emphasis is laid on low-level signal analysis where sound events are dtected & their parameters are estimated, & not so much on subsequent processing of note data to obtain larger musical structures. Theoretical background of different signal analysis methods is presented & their application to transcription problem is discussed.

– Mục đích của cuốn sách này: mô tả các thuật toán & mô hình cho các chủ đề phụ khác nhau của phiên âm nhạc, bao gồm phân tích cao độ, phân tích nhịp điệu, phiên âm bộ gõ, phân loại nhạc cụ, & phân tích cấu trúc âm nhạc. Trọng tâm chính được đặt vào phân tích tín hiệu mức thấp, trong đó các sự kiện âm thanh được phát hiện & ước tính các tham số của chúng, & không quá chú trọng vào việc xử lý dữ liệu nốt nhạc tiếp theo để thu được các cấu trúc âm nhạc lớn hơn. Cơ sở lý thuyết của các phương pháp phân tích tín hiệu khác nhau được trình bày & thảo luận về ứng dụng của chúng vào vấn đề phiên âm.

Primary target material considered in this book is complex music signals where several sounds are played simultaneously. These are referred to as polyphonic signals, in contrast to monophonic signals where at most 1 note is sounding at a time. For practical reasons, scope is limited to Western music, although not to any particular genre. Many of analysis methods make no assumptions about larger-scale structure of signal & are thus applicable to analysis of music from other cultures as well.

– Tài liệu chính được xem xét trong cuốn sách này là các tín hiệu âm nhạc phức tạp, trong đó nhiều âm thanh được phát đồng thời. Chúng được gọi là tín hiệu đa âm, trái ngược với tín hiệu đơn âm, trong đó chỉ có tối đa 1 nốt nhạc vang lên tại 1 thời điểm. Vì lý do thực tế, phạm vi nghiên cứu chỉ giới hạn ở âm nhạc phương Tây, mặc dù không áp dụng cho bất kỳ thể loại cụ thể nào. Nhiều phương pháp phân tích không đưa ra giả định về cấu trúc tín hiệu ở quy mô lớn hơn & do đó cũng có thể áp dụng để phân tích âm nhạc từ các nền văn hóa khác.

To give a reasonable estimate of achievable goals in automatic music transcription, it is instructive to study what human listeners are able to do in this task. An average listener perceives a lot of musically relevant information in complex audio signals. He or she can tap along with rhythm, hum melody (more or less correctly), recognize musical instruments, & locate structural parts of piece, e.g. chorus & verse in popular music. Harmonic changes & various details are perceived less consciously. Similarly to natural language, however, reading & writing music requires education. Not only notation needs to be studied, but recognizing different pitch intervals & timing relationships is an ability that has to be learned – these have to be encoded into a symbolic form in one's mind before writing them down. Moreover, an untrained listener is typically not able to hear inner lines in music (sub-melodies other than dominant one), so musical ear training is needed to develop an analytic mode of listening where these can be distinguished. Richer polyphonic complexity of a musical composition, more its transcription requires musical ear training & knowledge of particular musical style & of playing techniques of instruments involved.

– Để đưa ra ước tính hợp lý về các mục tiêu có thể đạt được trong việc phiên âm nhạc tự động, việc nghiên cứu những gì người nghe có thể làm trong nhiệm vụ này là rất hữu ích. 1 người nghe trung bình cảm nhận được rất nhiều thông tin liên quan đến âm nhạc trong các tín hiệu âm thanh phức tạp. Họ có thể gõ theo nhịp điệu, ngân nga giai điệu (ít nhiều chính xác), nhận biết nhạc cụ, & xác định các phần cấu trúc của bản nhạc, ví dụ như điệp khúc & câu trong nhạc đại chúng. Những thay đổi về hòa âm & các chi tiết khác nhau được cảm nhận ít có ý thức hơn. Tuy nhiên, tương tự như ngôn ngữ tự nhiên, việc đọc & viết nhạc đòi hỏi sự giáo dục. Không chỉ cần học ký hiệu, mà việc nhận biết các khoảng cao độ khác nhau & mối quan hệ về nhịp điệu cũng là 1 khả năng cần phải học – những điều này phải được mã hóa thành 1 dạng ký hiệu trong tâm trí trước khi viết chúng ra. Hơn nữa, 1 người nghe chưa được đào tạo thường không thể nghe được các dòng bên trong trong âm nhạc (các giai điệu phụ khác ngoài giai điệu chủ đạo), vì vậy cần phải rèn luyện thính giác âm nhạc để phát triển 1 phương thức nghe phân tích, trong đó có thể phân biệt được những điều này. 1 bản nhạc có độ phức tạp đa âm càng cao thì việc chuyển soạn

càng đòi hỏi phải rèn luyện khả năng cảm thụ âm nhạc & kiến thức về phong cách âm nhạc cụ thể & kỹ thuật chơi các nhạc cụ liên quan.

1st attempts towards automatic transcription of polyphonic music were made in 1970s, when Moorer proposed a system for transcribing 2-voice compositions. His work was followed by Chafe, Piszczalski, & Maher in 1980s. In all these early systems, number of concurrent voices was limited to 2 & pitch relationships of simultaneous sounds were restricted in various ways. On rhythm analysis side, 1st algorithm for beat tracking [Beat tracking refers to estimation of a rhythmic pulse which corresponds to tempo of a piece & (loosely) to foot-tapping rate of human listeners.] in general audio signals was proposed by Goto & Muraoka in 1990s, although this was preceded by a considerable amount of work for tracking beat in parametric note data & by beat-tracking algorithm of Schloss for percussive audio tracks. 1st attempts to transcribe percussive instruments were made in mid-1980s by Schloss & later by Bilmes, both of whom classified different types of conga strikes in continuous recordings. Transcription of polyphonic percussion tracks was later addressed by Goto & Muraoka. A more extensive description of early stages of music transcription has been given by Tanguiane.

– Những nỗ lực đầu tiên hướng tới việc phiên âm tự động nhạc đa âm được thực hiện vào những năm 1970, khi Moorer đề xuất 1 hệ thống phiên âm các tác phẩm 2 giọng. Công trình của ông được tiếp nối bởi Chafe, Piszczalski, & Maher vào những năm 1980. Trong tất cả các hệ thống ban đầu này, số lượng giọng nói đồng thời bị giới hạn ở 2 & mối quan hệ cao độ của các âm thanh đồng thời bị hạn chế theo nhiều cách khác nhau. Về phía phân tích nhịp điệu, thuật toán đầu tiên để theo dõi nhịp [Theo dõi nhịp đề cập đến việc ước tính xung nhịp điệu tương ứng với nhịp độ của 1 bản nhạc & (một cách lỏng lẻo) với tốc độ gõ chân của người nghe.] nói chung, tín hiệu âm thanh đã được Goto & Muraoka đề xuất vào những năm 1990, mặc dù trước đó đã có 1 lượng công việc đáng kể để theo dõi nhịp trong dữ liệu nốt nhạc tham số & bằng thuật toán theo dõi nhịp của Schloss cho các bản nhạc gõ. Những nỗ lực đầu tiên trong việc phiên âm nhạc cụ gõ được thực hiện vào giữa những năm 1980 bởi Schloss & sau đó là Bilmes, cả hai đều phân loại các loại phách conga khác nhau trong các bản ghi âm liên tục. Việc phiên âm các bản nhạc gõ đa âm sau đó được Goto & Muraoka đề cập. Tanguiane đã đưa ra 1 mô tả chi tiết hơn về các giai đoạn đầu của quá trình phiên âm nhạc.

Since beginning of 1990s, interest in music transcription has grown rapidly & not possible to make a complete account of work here. However, certain general trends & successful approaches can be discerned. 1 of these has been use of *statistical methods*. To mention a few examples, [...] proposed statistical methods for pitch analysis of polyphonic music; in beat tracking, statistical methods were employed by [...]; & in percussive instrument transcription by [...]. In musical instrument classification, statistical pattern recognition methods prevail. Another trend has been increasing utilization of *computational models of human auditory system*. These were 1st used for music transcription by Martin, & auditorily motivated methods have since been proposed for polyphonic pitch analysis by Karjalainen & Tolonen & Klapuri, & for beat tracking by Scheirer. Another prominent approach has been to model human *auditory scene analysis* (ASA) ability. Term ASA refers to way in which humans organize spectral components to their respective sound sources & recognize simultaneously occurring sounds. Principles of ASA were brought to pitch analysis of polyphonic music signals by Mellinger & Kashino, & later by Godsmark & Brown & Sterian. Most recently, several unsupervised learning methods have been proposed where a minimal number of prior assumptions are made about analyzed signal. Methods based on independent component analysis were introduced to music transcription by Casey, & various other methods were later proposed by [...]. Of course, there are also methods that do not represent any of above-mentioned trends, & a more comprehensive review of literature is presented in coming chaps.

– Kể từ đầu những năm 1990, sự quan tâm đến việc phiên âm nhạc đã tăng nhanh chóng & không thể đưa ra 1 báo cáo đầy đủ về công việc ở đây. Tuy nhiên, 1 số xu hướng chung & các phương pháp tiếp cận thành công có thể được nhận ra. 1 trong số đó là việc sử dụng *phương pháp thống kê*. Để đề cập đến 1 vài ví dụ, [...] đã đề xuất các phương pháp thống kê để phân tích cao độ của âm nhạc đa âm; trong việc theo dõi nhịp, các phương pháp thống kê đã được [...] sử dụng; & trong việc phiên âm nhạc cụ gõ bởi [...]. Trong phân loại nhạc cụ, các phương pháp nhận dạng mẫu thống kê chiếm ưu thế. 1 xu hướng khác là việc sử dụng ngày càng tăng các *mô hình tính toán của hệ thống thính giác của con người*. Những phương pháp này lần đầu tiên được Martin sử dụng để phiên âm nhạc, & các phương pháp lấy cảm hứng từ thính giác kể từ đó đã được Karjalainen & Tolonen & Klapuri đề xuất để phân tích cao độ đa âm, & để theo dõi nhịp bởi Scheirer. 1 phương pháp nổi bật khác là mô hình hóa khả năng *phân tích cảnh thính giác* của con người (ASA). Thuật ngữ ASA đề cập đến cách con người sắp xếp các thành phần phổ thành các nguồn âm thanh tương ứng & nhận biết các âm thanh phát ra đồng thời. Các nguyên lý của ASA đã được Mellinger & Kashino, & sau đó là Godsmark & Brown & Sterian đưa vào phân tích cao độ của các tín hiệu âm nhạc đa âm. Gần đây nhất, 1 số phương pháp học không giám sát đã được đề xuất, trong đó 1 số lượng tối thiểu các giả định trước đó được đưa ra về tín hiệu được phân tích. Các phương pháp dựa trên phân tích thành phần độc lập đã được Casey giới thiệu vào phiên âm nhạc, & nhiều phương pháp khác sau đó đã được [...] đề xuất. Tất nhiên, cũng có những phương pháp không đại diện cho bất kỳ xu hướng nào đã đề cập ở trên, & 1 bài tổng quan toàn diện hơn về tài liệu sẽ được trình bày trong các chương tiếp theo.

State-of-art music transcription systems are still clearly inferior to skilled human musicians in accuracy & flexibility. I.e., a reliable general-purpose transcription system does not exist at present time. However, some degree of success has been achieved for polyphonic music of limited complexity. In transcription of pitched instruments, typical restrictions are: number of concurrent sounds is limited, interference of drums & percussive sounds is not allowed, or only a specific instrument is considered. Some promising results for transcription of real-world music on CD recordings has been demonstrated by [...]. In percussion transcription, quite good accuracy has been achieved in transcription of percussive tracks which comprise a limited number of instruments (typically bass drum, snare, & hi-hat) & no pitched instruments. Also promising results have been reported for transcription of bass & snare drums on real-world recordings, but this is a more open problem. Beat tracking of complex real-world audio signals can be performed quite reliably with state-of-art methods, but difficulties remain especially

in analysis of classical music & rhythmically complex material. Comparative evaluations of beat-tracking systems can be found in [...]. Research on musical instrument classification has mostly concentrated on working with isolated sounds, although more recently this has been attempted in polyphonic audio signals, too.

– Các hệ thống phiên âm nhạc hiện đại vẫn rõ ràng kém hơn so với các nhạc công lành nghề về độ chính xác & tính linh hoạt. Tức là, hiện tại vẫn chưa có 1 hệ thống phiên âm đa năng đáng tin cậy. Tuy nhiên, 1 số thành công đã đạt được đối với nhạc đa âm có độ phức tạp hạn chế. Trong phiên âm nhạc cụ có cao độ, các hạn chế điển hình là: số lượng âm thanh đồng thời bị giới hạn, không được phép có sự can thiệp của trống & âm thanh gõ, hoặc chỉ xem xét 1 nhạc cụ cụ thể. [...] đã chứng minh 1 số kết quả đầy hứa hẹn cho việc phiên âm nhạc thực tế trên các bản ghi CD. Trong phiên âm bộ gõ, độ chính xác khá tốt đã đạt được trong phiên âm các bản nhạc gõ bao gồm 1 số lượng nhạc cụ hạn chế (thường là trống trầm, trống snare, & hi-hat) & không có nhạc cụ có cao độ. Cũng đã có những báo cáo về kết quả đầy hứa hẹn cho việc phiên âm trống trầm & âm snare trên các bản ghi thực tế, nhưng đây là 1 vấn đề còn bỏ ngỏ. Việc theo dõi nhịp của các tín hiệu âm thanh thực tế phức tạp có thể được thực hiện khá đáng tin cậy bằng các phương pháp hiện đại, nhưng vẫn còn nhiều khó khăn, đặc biệt là trong việc phân tích các tài liệu nhạc cổ điển & phức tạp về mặt nhịp điệu. Đánh giá so sánh các hệ thống theo dõi nhịp có thể được tìm thấy trong [...]. Nghiên cứu về phân loại nhạc cụ chủ yếu tập trung vào việc xử lý các âm thanh riêng lẻ, mặc dù gần đây điều này cũng đã được thử nghiệm trên các tín hiệu âm thanh đa âm.

- 1.1. Terminology & Concepts. Before turning to a more general discussion of music transcription problem & concepts of this book, necessary to introduce some basic terminology of auditory perception & music. To discuss music signals, 1st have to discuss perceptual attributes of sounds of which they consist. There are 4 subjective qualities that are particularly useful in characterizing sound events: pitch, loudness, duration, & timbre.

  – Trước khi đi sâu vào thảo luận chung hơn về vấn đề phiên âm âm nhạc & các khái niệm của cuốn sách này, cần giới thiệu 1 số thuật ngữ cơ bản về nhận thức thính giác & âm nhạc. Để thảo luận về tín hiệu âm nhạc, trước tiên cần thảo luận về các thuộc tính nhận thức của âm thanh mà chúng tạo thành. Có 4 đặc điểm chủ quan đặc biệt hữu ích trong việc mô tả các sự kiện âm thanh: cao độ, độ to, trường độ, & âm sắc.

  Pitch is a perceptual attribute which allows ordering of sounds on a frequency-related scale extending from low to high. More exactly, pitch is defined as frequency of a sine wave that is matched to target sound by human listeners. *Fundamental frequency* (F0) is corresponding physical term & is defined for periodic or nearly periodic sounds only. For these classes of sounds, F0 is defined as inverse of period & is closely related to pitch. In ambiguous situations, period corresponding to perceived pitch is chosen.

  – Cao độ là 1 thuộc tính nhận thức cho phép sắp xếp âm thanh theo thang tần số từ thấp đến cao. Chính xác hơn, cao độ được định nghĩa là tần số của sóng sin được người nghe khớp với âm thanh mục tiêu. *Tần số cơ bản* (F0) là 1 thuật ngữ vật lý tương ứng & chỉ được định nghĩa cho các âm thanh tuần hoàn hoặc gần tuần hoàn. Đối với các loại âm thanh này, F0 được định nghĩa là nghịch đảo của chu kỳ & có liên quan chặt chẽ đến cao độ. Trong các tình huống mơ hồ, chu kỳ tương ứng với cao độ cảm nhận được sẽ được chọn.

  Perceived loudness of an acoustic signal has a nontrivial connection to its physical properties, & computational models of loudness perception constitute a fundamental part of psychoacoustics. [Psychoacoustics is science that deals with perception of sound. In a psychoacoustic experiment, relationships between an acoustic stimuluus & resulting subjective sensation are studied by presenting specific tasks or questions to human listeners.] In music processing, however, often more convenient to express level of sounds with their mean-square power & to apply a logarithmic (decibel) scale to deal with wide dynamic range involved. Perceived *duration* of sound has more or less 1-1 mapping to its physical duration in cases where this can be unambiguously determined.

  – Độ lớn cảm nhận được của tín hiệu âm thanh có mối liên hệ không hề tầm thường với các đặc tính vật lý của nó, & các mô hình tính toán về nhận thức độ lớn tạo nên 1 phần cơ bản của tâm lý âm học. [Tâm lý âm học là khoa học nghiên cứu về nhận thức âm thanh. Trong 1 thí nghiệm tâm lý âm học, mối quan hệ giữa kích thích âm thanh & cảm giác chủ quan được nghiên cứu bằng cách đưa ra các nhiệm vụ hoặc câu hỏi cụ thể cho người nghe.] Tuy nhiên, trong xử lý âm nhạc, thường thuận tiện hơn khi biểu thị mức độ âm thanh bằng lũy thừa trung bình bình phương của chúng & áp dụng thang logarit (decibel) để xử lý dải động rộng liên quan. *Thời lượng* âm thanh cảm nhận được có ánh xạ ít nhiều 1-1 với thời lượng vật lý của nó trong những trường hợp có thể xác định rõ ràng.

  Timbre is sometimes referred to as sound 'color' & is closely related to recognition of sound sources. E.g., sounds of violin & flute may be identical in their pitch, loudness, & duration, but are still easily distinguished by their timbre. Concept is not explained by any simple acoustic property but depends mainly on coarse spectral energy distribution of a sound, & time evolution of this. Whereas pitch, loudness, & duration can be quite naturally encoded into a single scalar value, timbre is essentially a multidimensional concept & is typically represented with a feature vector in musical signal analysis tasks.

  – Âm sắc đôi khi được gọi là 'màu sắc' âm thanh & có liên quan chặt chẽ đến việc nhận dạng nguồn âm thanh. E.g., âm thanh của vĩ cầm & sáo có thể giống hệt nhau về cao độ, độ to, & trường độ, nhưng vẫn dễ dàng phân biệt bằng âm sắc. Khái niệm này không được giải thích bằng bất kỳ đặc tính âm học đơn giản nào mà chủ yếu phụ thuộc vào sự phân bố năng lượng phổ thô của âm thanh, & sự tiến triển theo thời gian của nó. Trong khi cao độ, độ to, & trường độ có thể được mã hóa 1 cách khá tự nhiên thành 1 giá trị vô hướng duy nhất, thì âm sắc về cơ bản là 1 khái niệm đa chiều & thường được biểu diễn bằng 1 vectơ đặc trưng trong các tác vụ phân tích tín hiệu âm nhạc.

  Musical information is generally encoded into relationships between individual sound events & between larger entities composed of these. Pitch relationships are utilized to make up melodies & chords. Timbre & loudness relationships are used to create musical form especially in percussive music, where pitched musical instruments are not necessarily employed at all.

Inter-onset interval (IOI) relationships, in turn, largely define rhythmic characteristics of a melody or a percussive sound sequence (term IOI refers to time interval between beginnings of 2 sound events). Although durations of sounds play a role too, IOIs are more crucial in determining perceived rhythm. Indeed, many rhythmically important instruments, e.g. drums & percussions, produce exponentially decaying wave shapes that do not even have a uniquely defined duration. In case of sustained musical sounds, however, durations are used to control articulation. 2 extremes here are 'staccato', where notes are cut very short, & 'legato', where no perceptible gaps are left between successive notes.

– Thông tin âm nhạc thường được mã hóa thành các mối quan hệ giữa các sự kiện âm thanh riêng lẻ & giữa các thực thể lớn hơn được tạo thành từ những sự kiện này. Mối quan hệ cao độ được sử dụng để tạo nên giai điệu & hợp âm. Mối quan hệ âm sắc & độ to được sử dụng để tạo ra hình thức âm nhạc, đặc biệt là trong âm nhạc gõ, trong đó các nhạc cụ có cao độ không nhất thiết phải được sử dụng. Ngược lại, các mối quan hệ khoảng cách giữa các lần bắt đầu (IOI) phần lớn xác định các đặc điểm nhịp điệu của 1 giai điệu hoặc 1 chuỗi âm thanh gõ (thuật ngữ IOI dùng để chỉ khoảng thời gian giữa các lần bắt đầu của 2 sự kiện âm thanh). Mặc dù thời lượng của âm thanh cũng đóng 1 vai trò, nhưng IOI quan trọng hơn trong việc xác định nhịp điệu được cảm nhận. Thật vậy, nhiều nhạc cụ quan trọng về mặt nhịp điệu, ví dụ như trống & bộ gõ, tạo ra các dạng sóng suy giảm theo cấp số nhân mà thậm chí không có thời lượng được xác định duy nhất. Tuy nhiên, trong trường hợp âm thanh âm nhạc kéo dài, thời lượng được sử dụng để kiểm soát cách phát âm. Hai thái cực ở đây là 'staccato', trong đó các nốt nhạc được cắt rất ngắn, & 'legato', trong đó không có khoảng cách nhận biết nào được để lại giữa các nốt nhạc liên tiếp.

A melody is a series of pitched sounds with musically meaningful pitch & IOI relationships. In written music, this corresponds to a sequence of single notes. A chord is a combination of ≥ 2 simultaneous notes. A chord can be harmonious or dissonant, subjective attributes related to specific relationships between component pitches & their overtone partials. Harmony refers to part of music theory which studies formation & relationships of chords.

– Giai điệu là 1 chuỗi các âm thanh có cao độ với các mối quan hệ cao độ & IOI có ý nghĩa về mặt âm nhạc. Trong âm nhạc viết, điều này tương ứng với 1 chuỗi các nốt đơn. Hợp âm là sự kết hợp của ≥ 2 nốt nhạc đồng thời. 1 hợp âm có thể là hòa âm hoặc bất hòa, các thuộc tính chủ quan liên quan đến mối quan hệ cụ thể giữa các cao độ thành phần & các phần âm bội của chúng. Hòa âm đề cập đến 1 phần của lý thuyết âm nhạc, nghiên cứu sự hình thành & mối quan hệ của hợp âm.

p 9 (19)+++

- 2. An Introduction to Statistical Signal Processing & Spectrum Estimation.
- 3. Sparse Adaptive Representations for Musical Signals.

  PART II: RHYTHM & TIMBRE ANALYSIS.
- 4. Beat Tracking & Music Metre Analysis.
- 5. Unpitched Percussion Transcription.
- 6. Automatic Classification of Pitched Musical Instrument Sounds.

  PART III: MULTIPLE FUNDAMENTAL FREQUENCY ANALYSIS.
- 7. Multiple Fundamental Frequency Estimation Based on Generative Models.
- 8. Auditory Model-Based Methods for Multiple Fundamental Frequency Estimation.
- 9. Unsupervised Learning Methods for Source Separation in Monaural Music Signals.

  PART IV: ENTIRE SYSTEMS, ACOUSTIC & MUSICOLOGICAL MODELING.
- 10. Auditory Scene Analysis in Music Signals.
- 11. Music Scene Description.
- 12. Singing Transcription.

1.5 Mengshan Li. Design & Implementation of Piano Audio Automatic Music Transcription Algorithm Based on CNN

- Abstract. Present design & implementation of an AMT algorithm for piano audio, utilizing an optimized CNN with optimal parameters. In this study, adopt cepstral coefficient derived from cochlear filters, a method commonly used in speech signal processing, for extracting features from transformed musical audio. Conventional CNNs often rely on a universally shared convolutional kernel when processing piano audio, but this approach fails to account for variations in information across different frequency bands. To address this, select 24 Mel filters, each featuring a distinct center frequency ranging from 105 to 19093 Hz, which aligns with 44100 Hz sampling rate of converted music. This setup enables system to effectively capture key characteristics of piano audio signals across a wide frequency range, providing a solid frequency-domain foundation for subsequent music transcription algorithms.

  – Tóm tắt. Trình bày thiết kế & triển khai thuật toán AMT cho âm thanh piano, sử dụng CNN được tối ưu hóa với các tham số tối ưu. Trong nghiên cứu này, áp dụng hệ số cepstral có nguồn gốc từ bộ lọc ốc tai, 1 phương pháp thường được sử dụng trong xử lý tín hiệu giọng nói, để trích xuất các đặc điểm từ âm thanh nhạc đã chuyển đổi. Các CNN thông thường thường

dựa vào 1 hạt nhân tích chập được chia sẻ chung khi xử lý âm thanh piano, nhưng cách tiếp cận này không tính đến các biến thể thông tin trên các dải tần số khác nhau. Để giải quyết vấn đề này, hãy chọn 24 bộ lọc Mel, mỗi bộ lọc có tần số trung tâm riêng biệt trong khoảng từ 105 đến 19093 Hz, phù hợp với tốc độ lấy mẫu 44100 Hz của nhạc đã chuyển đổi. Thiết lập này cho phép hệ thống nắm bắt hiệu quả các đặc điểm chính của tín hiệu âm thanh piano trên 1 dải tần số rộng, cung cấp nền tảng miền tần số vững chắc cho các thuật toán phiên âm nhạc tiếp theo.

Keywords. CNN, piano audio, design of AMT algorithm, filtering.

- 1. Introduction. Automatic classification method of music transcription is a technology that uses computer technology to identify & classify converted music automatically. It combines multidisciplinary knowledge e.g. piano audio signal processing, computer science, & ML, & it aims to realize task of automatic classification of converted music works [1, 2]. Compared with traditional manual classification method, AMT technology can improve classification efficiency, especially in managing large-scale converted music libraries & information retrieval of music transcription [3, 4]. As ML & AI technologies advance, automatic subclassification of converted music is continually evolving & optimizing, offering new solutions & opportunities for managing & servicing translation industry [5]. While this study focuses on piano audio due to its unique spectral characteristics & availability of well-annotated datasets, proposed frequency-based local shared optimized convolution kernel design holds promise for generalization to other musical instruments [6, 7]. E.g., string or wind instruments, which exhibit distinct timbral & spectral features, could benefit from similar frequency-domain partitioning & kernel optimization tailored to their specific acoustic properties. Additionally, exploring integration of other filter banks, e.g. bark-scale filters or gammatone filters, which mimic different aspects of human auditory processing, could further enhance feature representation for diverse instrument classifications [8, 9].

  – Phương pháp phân loại tự động bản ghi âm nhạc là 1 công nghệ sử dụng công nghệ máy tính để xác định & phân loại nhạc đã chuyển đổi 1 cách tự động. Nó kết hợp kiến thức đa ngành, ví dụ như xử lý tín hiệu âm thanh piano, khoa học máy tính, & ML, & nó nhằm mục đích hiện thực hóa nhiệm vụ phân loại tự động các tác phẩm âm nhạc đã chuyển đổi [1, 2]. So với phương pháp phân loại thủ công truyền thống, công nghệ AMT có thể cải thiện hiệu quả phân loại, đặc biệt là trong việc quản lý các thư viện nhạc đã chuyển đổi quy mô lớn & truy xuất thông tin về bản ghi âm nhạc [3, 4]. Khi công nghệ ML & AI phát triển, việc phân loại phụ tự động của nhạc đã chuyển đổi liên tục phát triển & tối ưu hóa, cung cấp các giải pháp mới & cơ hội để quản lý & phục vụ ngành công nghiệp dịch thuật [5]. Mặc dù nghiên cứu này tập trung vào âm thanh piano do các đặc điểm phổ độc đáo & tính khả dụng của các tập dữ liệu được chú thích tốt, thiết kế hạt nhân tích chập cục bộ được tối ưu hóa chia sẻ dựa trên tần số được đề xuất hứa hẹn sẽ được khái quát hóa cho các nhạc cụ khác [6, 7]. Ví dụ, nhạc cụ dây hoặc nhạc cụ hơi, thể hiện các đặc điểm âm sắc & phổ riêng biệt, có thể được hưởng lợi từ phân vùng miền tần số tương tự & tối ưu hóa hạt nhân được điều chỉnh theo các đặc tính âm học cụ thể của chúng. Ngoài ra, việc khám phá sự tích hợp của các ngân hàng bộ lọc khác, ví dụ như bộ lọc quy mô vỏ cây hoặc bộ lọc gammatone, mô phỏng các khía cạnh khác nhau của quá trình xử lý thính giác của con người, có thể cải thiện hơn nữa khả năng biểu diễn tính năng cho các phân loại nhạc cụ đa dạng [8, 9].

  In this paper, propose a novel technique known as frequency-based local shared optimized convolution kernel design. Method divides piano audio signal into multiple frequency domains, with each segment processed by an independently optimized convolution kernel, thereby capitalizing on distinct local features of piano signal within each frequency band. This approach enables more accurate tracking of musical transcription features across varying frequency ranges, leading to a substantial improvement in both precision & overall performance of music transcription [10]. Conventional CNN models face notable challenges in subclassifying converted music, as they tend to neglect spectral conversion traits intrinsic to piano audio signals [11, 12]. These potential extensions highlight methodology's adaptability & open new avenues for cross-instrument music information processing [13, 14], this approach redesigns network architecture & enhances feature extraction process by integrating both frequency & time-domain attributes of piano audio signals. Proposed model divides piano audio signal into separate frequency regions, based on unique characteristics of converted music frequencies, ensuring better adaptability & performance in complex music classification tasks. In each of these regions, specialized optimized convolution kernels are applied locally, which improves precision & efficiency of fine classification [15, 16]. This new convolution neural network model, optimized for spectral transformation, effectively captures essential features of spectral conversion music signals, enabling more accurate differentiation of various types & styles of converted music. By combining expertise in DL & transcription music, this method provides new technical paths & solutions for transcription music information processing [17, 18]. It promotes development & application of automated processing technology for transcription music. Optimized convolution optimal parameters based on spectral conversion music fine classification method shows great potential & advantages in solving problems of spectral conversion music fine classification & retrieval [19].

  – Trong bài báo này, chúng tôi đề xuất 1 kỹ thuật mới được gọi là thiết kế hạt nhân tích chập tối ưu hóa cục bộ chia sẻ dựa trên tần số. Phương pháp này chia tín hiệu âm thanh piano thành nhiều miền tần số, với mỗi phân đoạn được xử lý bởi 1 hạt nhân tích chập tối ưu hóa độc lập, do đó tận dụng các đặc điểm cục bộ riêng biệt của tín hiệu piano trong mỗi dải tần. Phương pháp này cho phép theo dõi chính xác hơn các đặc điểm phiên âm nhạc trên các dải tần số khác nhau, dẫn đến cải thiện đáng kể cả độ chính xác & hiệu suất tổng thể của phiên âm nhạc [10]. Các mô hình CNN thông thường gặp phải những thách thức đáng kể trong việc phân loại phụ âm nhạc đã chuyển đổi, vì chúng có xu hướng bỏ qua các đặc điểm chuyển đổi phổ vốn có trong tín hiệu âm thanh piano [11, 12]. Những mở rộng tiềm năng này làm nổi bật khả năng thích ứng của phương pháp & mở ra những hướng đi mới cho xử lý thông tin âm nhạc giữa các nhạc cụ [13, 14], phương pháp này thiết kế lại kiến trúc mạng & cải thiện quy trình trích xuất đặc điểm bằng cách tích hợp cả thuộc tính tần số & miền thời gian của tín hiệu âm thanh piano. Mô hình đề xuất chia tín hiệu âm thanh piano thành các vùng tần số riêng biệt, dựa trên các đặc điểm riêng biệt của tần số âm nhạc đã chuyển đổi, đảm bảo khả năng thích ứng tốt hơn & hiệu suất trong các tác vụ phân loại âm nhạc

phức tạp. Trong mỗi vùng này, các hạt nhân tích chập được tối ưu hóa chuyên biệt được áp dụng cục bộ, giúp cải thiện độ chính xác & hiệu quả của phân loại chi tiết [15, 16]. Mô hình mạng nơ-ron tích chập mới này, được tối ưu hóa cho phép biến đổi phổ, nắm bắt hiệu quả các đặc điểm thiết yếu của tín hiệu âm nhạc chuyển đổi phổ, cho phép phân biệt chính xác hơn các loại & phong cách âm nhạc đã chuyển đổi khác nhau. Bằng cách kết hợp chuyên môn về DL & âm nhạc phiên âm, phương pháp này cung cấp các giải pháp kỹ thuật mới & cho việc xử lý thông tin âm nhạc phiên âm [17, 18]. Nó thúc đẩy sự phát triển & ứng dụng công nghệ xử lý tự động cho âm nhạc phiên âm. Các tham số tối ưu tích chập được tối ưu hóa dựa trên phương pháp phân loại chi tiết âm nhạc chuyển đổi phổ cho thấy tiềm năng & lợi thế lớn trong việc giải quyết các vấn đề phân loại chi tiết âm nhạc chuyển đổi phổ & truy xuất [19].

- 2. Related works.
  - 2.1. CNN analysis. Optimized convolution optimal parameter-neural network based on spectral conversion characteristics is proposed & implemented as a fine classification method of spectral conversion music. Traditional optimized convolution optimal parameters-neural networks are significantly limited when processing piano audio: optimized convolution kernel parameters are shared globally. As shown in (1)–(2), $s(t)$ is convolution value contribution function, & $s(i,j)$ is piano audio feature function, W: advanced feature value extracted from deep convolutional layer, M: vanishing gradient value, m: residual connection value, i.e., no matter which frequency region piano audio feature is, network will process it in same way, thus ignoring difference of frequency domain information.

$$s(t) = x(t) * w(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau),$$

$$s(i,j) = \sum_{m=0}^{M}\sum_{n=0}^{N}(w_{m,n}x_{i+m} + w_b).$$

NQBH: where j??? or w_b or w_j? Mathematical formulas in this paper are unreliable!!!

– Phân tích CNN. Mạng nơ-ron tham số tối ưu tích chập được tối ưu hóa dựa trên các đặc điểm chuyển đổi phổ được đề xuất & triển khai như 1 phương pháp phân loại tốt cho âm nhạc chuyển đổi phổ. Các mạng nơ-ron tham số tối ưu tích chập truyền thống bị hạn chế đáng kể khi xử lý âm thanh piano: các tham số hạt nhân tích chập được tối ưu hóa được chia sẻ trên toàn cục. Như thể hiện trong (1)–(2), $s(t)$ là hàm đóng góp giá trị tích chập, & $s(i,j)$ là hàm đặc trưng âm thanh piano, W: giá trị đặc trưng nâng cao được trích xuất từ lớp tích chập sâu, M: giá trị gradient biến mất, m: giá trị kết nối dư, tức là, bất kể đặc trưng âm thanh piano ở vùng tần số nào, mạng sẽ xử lý nó theo cùng 1 cách, do đó bỏ qua sự khác biệt về thông tin miền tần số.

$$s(t) = x(t) * w(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau),$$

$$s(i,j) = \sum_{m=0}^{M}\sum_{n=0}^{N}(w_{m,n}x_{i+m} + w_b).$$

Even if same piano audio feature occurs in different frequency regions, its meaning & role may differ. To overcome this limitation, as shown in (3), L is optimal parameter value of piano audio processing, N: local feature value, x: audio processing value, & a: sequence data value. This paper proposes a novel method: optimized convolutional optimal parameter-neural network based on spectral conversion characteristics. This method devides time-frequency features of converted music into different regions according to their frequencies & only shares optimized convolution kernel in each specific area. (3)

$$L = \frac{1}{2n}\sum_{x} y(x) - a^{L(x)^2}.$$

I.e., optimization convolution kernel within each frequency region can specifically learn & adapt to unique piano audio features within area, as shown in (4), where L is frequency region value, & w is optimization factor coefficient, without being disturbed or confused by features of other frequency regions. Working mode of human transsepctral system provides a theoretical basis for this method. (4)

$$\partial_w L = (a-y)\sigma'(z)x.$$

– Ngay cả khi cùng 1 đặc điểm âm thanh piano xuất hiện ở các vùng tần số khác nhau, ý nghĩa & vai trò của nó có thể khác nhau. Để khắc phục hạn chế này, như được thể hiện trong (3), L là giá trị tham số tối ưu của quá trình xử lý âm thanh piano, N: giá trị đặc điểm cục bộ, x: giá trị xử lý âm thanh, & a: giá trị dữ liệu chuỗi. Bài báo này đề xuất 1 phương pháp mới: mạng nơ-ron tham số tối ưu tích chập được tối ưu hóa dựa trên các đặc điểm chuyển đổi phổ. Phương pháp này chia các đặc điểm thời gian-tần số của âm nhạc đã chuyển đổi thành các vùng khác nhau theo tần số của chúng & chỉ chia sẻ hạt nhân tích chập được tối ưu hóa trong mỗi vùng cụ thể. (3)

$$L = \frac{1}{2n}\sum_{x} y(x) - a^{L(x)^2}.$$

Tức là, hạt nhân tích chập tối ưu hóa trong mỗi vùng tần số có thể học & thích ứng cụ thể với các đặc điểm âm thanh piano riêng biệt trong vùng, như thể hiện trong (4), trong đó L là giá trị vùng tần số, & w là hệ số hệ số tối ưu hóa, mà

không bị nhiễu hoặc nhầm lẫn bởi các đặc điểm của các vùng tần số khác. Chế độ làm việc của hệ thống xuyên vách ngăn của con người cung cấp cơ sở lý thuyết cho phương pháp này. (4)

$$\partial_w L = (a - y)\sigma'(z)x.$$

Human ear can distinguish different sound characteristics according to sound frequency, as shown in (5)–(6), y is different timbre characteristic values, & a is an audible estimation coefficient to understand & process sound signals more effectively. Similarly, optimized convolutional optimal parameter-neural network based on spectral conversion characteristics simulates this spectral conversion processing process. It enhances sensitivity & accuracy of frequency domain information of piano audio signals through frequency partition. (5)–(6)

– Tai người có thể phân biệt các đặc điểm âm thanh khác nhau theo tần số âm thanh, như thể hiện trong (5)–(6), y là các giá trị đặc điểm âm sắc khác nhau, & a là hệ số ước lượng âm thanh để hiểu & xử lý tín hiệu âm thanh hiệu quả hơn. Tương tự, mạng nơ-ron tham số tối ưu tích chập được tối ưu hóa dựa trên các đặc điểm chuyển đổi phổ mô phỏng quá trình xử lý chuyển đổi phổ này. Nó tăng cường độ nhạy & độ chính xác của thông tin miền tần số của tín hiệu âm thanh piano thông qua phân vùng tần số.

$$L = \frac{1}{n}\sum_x [y \ln a + (1 - y)\ln(1 - a)],$$
$$\partial_w L = x(\sigma(z) - y).$$

1stly, necessary to preprocess converted music signal, including framing & windowing, to convert continuous piano audio signal into discrete time-frequency features. As shown in (7)–(8), g: framing parameter value, n: windowing times, & each time-frequency frame is input into optimized convolution optimal parameter-neural network designed based on spectral transformation characteristics. Network's structure & parameter setting should consider distribution characteristics of converted music signal in frequency domain & characteristics differences in different frequency regions. (7)–(8)

– Đầu tiên, cần phải xử lý trước tín hiệu âm nhạc đã chuyển đổi, bao gồm cả đóng khung & tạo cửa sổ, để chuyển đổi tín hiệu âm thanh piano liên tục thành các đặc trưng thời gian-tần số rời rạc. Như được thể hiện trong (7)–(8), g: giá trị tham số đóng khung, n: thời gian cửa sổ, & mỗi khung thời gian-tần số được đưa vào mạng nơ-ron tham số-tối ưu tích chập được tối ưu hóa, được thiết kế dựa trên các đặc trưng biến đổi phổ. Cấu trúc & thiết lập tham số của mạng nên xem xét các đặc trưng phân bố của tín hiệu âm nhạc đã chuyển đổi trong miền tần số & sự khác biệt về đặc trưng ở các vùng tần số khác nhau. (7)–(8)

$$\hat{g} = \frac{1}{n}\sum_i \nabla_\theta L(f(x^i;\theta), y^i),$$
$$\theta = \theta - \varepsilon\hat{g},$$

Through training & optimization, network can gradually learn & extract most representative features of converted music in different frequency regions to realize accurate classification & recognition of converted music. As shown in (9)–(10), y (or *lambda*?) is eigenvalue of music transcription, & x: substitution value of convolutional optimal parameter. This optimized convolutional optimal parameter-neural network method based on music transcription characteristics can improve accuracy & effect of subclassification of music transcription music & bring new technological breakthroughs & application prospects to field of music transcription music information processing. (9)–(10)

– Thông qua huấn luyện & tối ưu hóa, mạng có thể dần dần học & trích xuất các đặc điểm tiêu biểu nhất của âm nhạc đã chuyển đổi ở các vùng tần số khác nhau để thực hiện phân loại & nhận dạng chính xác âm nhạc đã chuyển đổi. Như được thể hiện trong (9)–(10), y (hay *lambda*?) là giá trị riêng của bản sao âm nhạc, & x: giá trị thay thế của tham số tối ưu tích chập. Phương pháp mạng nơ-ron tham số tối ưu tích chập được tối ưu hóa dựa trên các đặc điểm của bản sao âm nhạc này có thể cải thiện độ chính xác & hiệu quả của việc phân loại phụ bản sao âm nhạc & mang lại những đột phá công nghệ mới & triển vọng ứng dụng cho lĩnh vực xử lý thông tin âm nhạc. (9)–(10)

$$v = av - \varepsilon\hat{g},$$
$$C(i,t) = 1 - \frac{1}{1 + e^{(H_{it} - \theta)\lambda}}.$$

It provides powerful tools & methods for automatic music classification, information retrieval, intelligent music recommendation systems, etc., & promotes technology's progress & application innovation. As shown in (11)–(12), $K()$ is intelligent converted music function, & $E()$ is parameter estimation function. Optimized convolution optimal parameter-neural network based on music transcription characteristics has important theoretical significance & practical application value in dealing with subclassification of music transcription music.

– Nó cung cấp các công cụ mạnh mẽ & phương pháp cho phân loại nhạc tự động, truy xuất thông tin, hệ thống đề xuất nhạc thông minh, v.v., & thúc đẩy sự tiến bộ của công nghệ & đổi mới ứng dụng. Như được thể hiện trong (11)–(12), $K()$ là hàm âm nhạc được chuyển đổi thông minh, & $E()$ là hàm ước lượng tham số. Mạng nơ-ron tham số tối ưu tích chập dựa trên các đặc điểm phiên âm nhạc có ý nghĩa lý thuyết quan trọng & giá trị ứng dụng thực tế trong việc phân loại phụ các bản phiên âm nhạc. (11)–(12)

$$K(m,n) = S(m,n) + \min\{K(m,n-1), K(m-1,n)\},$$

$$E(x) = \sum_{k \in \left[n-\frac{d}{2}, n+\frac{d}{2}\right]} |x(k)|^2.$$

- 2.2. Piano audio frame level dataset. Fourier transform is a powerful tool that can convert complex time-domain stationary signals to frequency domains, so this study can analyze signals using frequency domain characteristics. As shown in (13), CMAP is time-domain stationary signal value, & various frequency components contained in signal can be clearly understood. However, Fourier transform has a limitation that it cannot handle unstable signals. (13)

  – Bộ dữ liệu mức khung âm thanh Piano. Biến đổi Fourier là 1 công cụ mạnh mẽ có thể chuyển đổi tín hiệu dừng phức tạp trong miền thời gian sang miền tần số, do đó nghiên cứu này có thể phân tích tín hiệu bằng cách sử dụng các đặc tính miền tần số. Như được thể hiện trong (13), CMAP là giá trị tín hiệu dừng trong miền thời gian, & các thành phần tần số khác nhau chứa trong tín hiệu có thể được hiểu rõ ràng. Tuy nhiên, biến đổi Fourier có 1 hạn chế là không thể xử lý tín hiệu không ổn định.

$$\hat{c}_{\mathrm{MAP}} = \arg\max_{c \in C} P(cx) = \arg\max_{c \in C} \frac{P(cx)P(c)}{P(x)}.$$

  Short-time Fourier transform came into being. Short-time Fourier transform divides signal into short-time approximately stationary segments using a window function, then performs a fast Fourier transform on each segment to obtain spectrograms at different times. As shown in (14), CML is frequency information value of short-time Fourier transform to know each frequency information appearing at a specific time. (14)

  – Biến đổi Fourier thời gian ngắn ra đời. Biến đổi Fourier thời gian ngắn chia tín hiệu thành các đoạn xấp xỉ dừng ngắn hạn bằng hàm cửa sổ, sau đó thực hiện biến đổi Fourier nhanh trên mỗi đoạn để thu được phổ đồ tại các thời điểm khác nhau. Như thể hiện trong (14), CML là giá trị thông tin tần số của biến đổi Fourier thời gian ngắn để biết từng thông tin tần số xuất hiện tại 1 thời điểm cụ thể. (14)

$$\hat{c}_{\mathrm{ML}} = \arg\max_{c \in C} P(cx).$$

  There are also some problems with short-time Fourier transform. Its window size & shape are fixed, & it lacks adaptability. If window is selected too small, it may lead to sufficient signal length in window, accurate frequency analysis & low-frequency resolution. However, choosing a smaller window may contain less information in time. As shown in (15)–(16), w is window selection coefficient, & $filter$ is filtering frequency coefficient, which leads to decrease of time resolution & impossibility of fine time domain analysis of signal. (15)–(16)

  – Biến đổi Fourier thời gian ngắn cũng có 1 số vấn đề. Kích thước cửa sổ & hình dạng của nó là cố định, & thiếu khả năng thích ứng. Nếu chọn cửa sổ quá nhỏ, nó có thể dẫn đến độ dài tín hiệu đủ lớn trong cửa sổ, phân tích tần số chính xác & độ phân giải tần số thấp. Tuy nhiên, chọn cửa sổ nhỏ hơn có thể chứa ít thông tin hơn theo thời gian. Như thể hiện trong (15)–(16), w là hệ số lựa chọn cửa sổ, & $filter$ là hệ số lọc tần số, điều này dẫn đến giảm độ phân giải thời gian & không thể phân tích tín hiệu trong miền thời gian chính xác. (15)–(16)

$$\begin{aligned} w_{i,\sin} &= (\sin 2\pi f_i t_1, \sin 2\pi f_i t_2, \ldots, \sin 2\pi f_i t_s), \\ filter_i &= (w_{i,\sin}^{\top} x_t)^2 + (w_{i,\cos}^{\top} x_t)^2. \end{aligned}$$

  To overcome defect of fixed window of short-time Fourier transform, wavelet transform inherits & develops localization idea of short-time Fourier transform. As shown in (17)–(18), P is fixed wavelet coefficient, & F is adaptive adjustment coefficient, TP: window width, & FP: optimization parameter. Wavelet transform introduces a "time-frequency" window that can change with frequency, which can be adaptively adjusted according to frequency characteristics of signal. In high-frequency part, window can subdivide time & improve temporal resolution. (17)–(18)

  – Để khắc phục nhược điểm cửa sổ cố định của phép biến đổi Fourier thời gian ngắn, phép biến đổi wavelet kế thừa & phát triển ý tưởng định vị của phép biến đổi Fourier thời gian ngắn. Như thể hiện trong (17)–(18), P là hệ số wavelet cố định, F là hệ số điều chỉnh thích nghi, TP: độ rộng cửa sổ, FP: tham số tối ưu. Phép biến đổi wavelet giới thiệu 1 cửa sổ "thời gian-tần số" có thể thay đổi theo tần số, có thể được điều chỉnh thích nghi theo đặc tính tần số của tín hiệu. Ở phần tần số cao, cửa sổ có thể chia nhỏ thời gian & cải thiện độ phân giải thời gian. (17)–(18)

$$P = \frac{U_{TP}}{U_{TP} + U_{FP}}, \; F_1 = 2\frac{PR}{P+R}.$$

  Window can subdivide frequency & improve frequency resolution in low-frequency part. This makes wavelet transform an ideal tool for signal time–frequency analysis, which can capture signal changes in time & frequency more accurately. As shown in (19)–(20), R is wavelet selection coefficient, Q is signal analysis coefficient, F is signal energy density, P is width of filtering window, b is filter order, & k is spectral density. Fourier transform, short-time Fourier transform, & wavelet transform are all important tools in signal analysis. (19)–(20)

  – Cửa sổ có thể chia nhỏ tần số & cải thiện độ phân giải tần số trong phần tần số thấp. Điều này làm cho biến đổi wavelet trở thành 1 công cụ lý tưởng cho phân tích thời gian - tần số tín hiệu, có thể nắm bắt các thay đổi tín hiệu theo thời gian & tần số chính xác hơn. Như được thể hiện trong (19)–(20), R là hệ số lựa chọn wavelet, Q là hệ số phân tích tín hiệu, F là mật độ năng lượng tín hiệu, P là độ rộng cửa sổ lọc, b là bậc lọc, & k là mật độ phổ. Biến đổi Fourier, biến đổi Fourier thời gian ngắn, & biến đổi wavelet đều là những công cụ quan trọng trong phân tích tín hiệu. (19)–(20)

$$F_\beta = (1+\beta^2)\frac{PR}{\beta^2 P + R}, \; Q = \frac{f_k}{\delta_{f_k}} = \frac{f_k}{f_{k+1} - f_k} = \left(2^{\frac{1}{b}} - 1\right)^{-1}.$$

Each method has its own unique advantages & applicable scenarios. As shown in (21), N: number of wavelet transforms. By introducing a variable time–frequency window, wavelet transforms effectively make up for deficiency of fixed window of short-time Fourier transform & provide a more flexible & accurate solution for time-frequency analysis of signals. (21)

– Mỗi phương pháp đều có những ưu điểm riêng & các trường hợp áp dụng. Như được thể hiện trong (21), N: số phép biến đổi wavelet. Bằng cách đưa vào 1 cửa sổ thời gian-tần số biến đổi, phép biến đổi wavelet bù đắp hiệu quả cho sự thiếu hụt của cửa sổ cố định của phép biến đổi Fourier thời gian ngắn & cung cấp 1 giải pháp linh hoạt hơn & chính xác hơn cho việc phân tích thời gian-tần số của tín hiệu. (21)

$$N_k = \frac{f_s}{\delta_{f_k}} = \frac{f_s Q}{f_k}.$$

- 3. Piano audio AMT algorithm based on time-frequency cepstrum coefficient optimization.
  - 3.1. Piano audio AMT algorithm based on application of Mel frequency cepstrum coefficient. Formants play a crucial role in distinguishing different sounds within a spectrogram, making accurate extraction of formants essential for capturing features of converted music. In formant extraction process, important to locate their positions & capture variations between formants, which together form music's spectral envelope [20, 21]. Beyond envelope information, input signal's spectrogram also contains intricate spectral details. Therefore, effectively separating envelope from these finer details is critical for obtaining precise envelope information. This separation can be achieved through various signal processing techniques, e.g. wavelet transform, Hilbert transform, among others [22, 23]. These methods allow for isolation of envelope & detailed information within spectrogram, enabling a more accurate analysis & identification of formant characteristics in converted music signal [24, 25]. Fig. 1: Schematic diagram of basic architecture of convolutional neural network. depicts fundamental structure of a convolutional neural network. In particular, wavelet transform emerges as a highly effective method for extracting spectral envelope, providing notable benefits in this application. It can adjust size & shape of analysis window according to time-frequency characteristics of signal, thus capturing changes in spectral envelope more accurately [26, 27]. On other hand, Hilbert transform can decompose complex signal into its envelope & phase components, further helping to distinguish envelope information from detail information in spectrogram [28, 29].

  – Thuật toán AMT âm thanh piano dựa trên ứng dụng hệ số cepstrum tần số Mel. Các formant đóng vai trò quan trọng trong việc phân biệt các âm thanh khác nhau trong 1 phổ đồ, khiến việc trích xuất chính xác các formant trở nên cần thiết để nắm bắt các đặc điểm của âm nhạc đã chuyển đổi. Trong quá trình trích xuất formant, việc xác định vị trí của chúng & nắm bắt các biến thể giữa các formant, cùng nhau tạo thành đường bao phổ của âm nhạc [20, 21]. Ngoài thông tin đường bao, phổ đồ của tín hiệu đầu vào còn chứa các chi tiết phổ phức tạp. Do đó, việc tách hiệu quả đường bao khỏi các chi tiết nhỏ hơn này là rất quan trọng để có được thông tin đường bao chính xác. Sự tách biệt này có thể đạt được thông qua nhiều kỹ thuật xử lý tín hiệu khác nhau, ví dụ như biến đổi wavelet, biến đổi Hilbert, v.v. [22, 23]. Các phương pháp này cho phép cô lập thông tin đường bao & chi tiết trong phổ đồ, cho phép phân tích chính xác hơn & xác định các đặc điểm formant trong tín hiệu âm nhạc đã chuyển đổi [24, 25]. Hình 1: Sơ đồ kiến trúc cơ bản của mạng nơ-ron tích chập. mô tả cấu trúc cơ bản của 1 mạng nơ-ron tích chập. Đặc biệt, biến đổi wavelet nổi lên như 1 phương pháp hiệu quả cao để trích xuất đường bao phổ, mang lại những lợi ích đáng kể trong ứng dụng này. Nó có thể điều chỉnh kích thước & hình dạng của cửa sổ phân tích theo đặc điểm thời gian-tần số của tín hiệu, do đó nắm bắt những thay đổi trong đường bao phổ chính xác hơn [26, 27]. Mặt khác, biến đổi Hilbert có thể phân tích tín hiệu phức tạp thành các thành phần đường bao & pha của nó, giúp phân biệt thông tin đường bao với thông tin chi tiết trong phổ đồ [28, 29].

  Automatic piano transcription has historically relied on a mix of traditional signal processing techniques & ML methods. Early approaches often utilized methods e.g.:

  1. **Fourier transform & spectral analysis.** These techniques analyze frequency-domain information to identify individual notes in an audio signal. While effective for monophonic or simple polyphonic recordings, their performance deteriorates in complex polyphonic contexts. Hidden Markov Models (HMMs): HMMs have been used to model temporal & sequential dependencies in music. They require significant domain-specific feature engineering & struggle with high polyphony levels.
  2. **Matrix factorization techniques.** Nonnegative matrix factorization (NMF) has been employed to decompose audio signals into constituent components, with each component ideally corresponding to a piano note. However, these methods often require manual tuning & lack robustness in real-world scenarios.
  3. **Deep neural networks (DNNs).** More recently, DL techniques, including fully connected networks & RNNs, have shown significant promise. These methods leverage large datasets to learn complex audio features, often outperforming traditional approaches.
  4. **CNNs.** CNNs have emerged as a powerful tool for automatic piano transcription due to their ability to capture local patterns in spectrogram representations of audio signals. By convolving kernels over input features, CNNs extract meaningful information that aids in identifying pitch, duration, & intensity of notes. As a DL approach, CNNs exhibit powerful representation & learning capabilities for piano audio subclassification. To optimize convolutional parameters & neural network structure, Fig. 2: Block diagram of piano audio feature extraction & matching algorithm. illustrates block diagram of piano audio feature extraction & matching algorithm. In this paper, while designing optimized CNN for spectral characteristic optimization, kernel size & depth are fine-tuned to better match spatial distribution & spectral properties of input features. In particular, study focuses on frequency partitioning & optimized convolution kernel sharing, incorporating techniques from music transcription systems to allow network to more effectively capture & utilize piano audio features across different frequency ranges. MFCC is a technique that extracts features by simulating nonlinear response of human auditory system to frequency. Calculation process involves signal framing, fast Fourier transform, Mel filter

bank processing, logarithmic operation, & discrete cosine transform (DCT). CFCC is based on cochlear filtering model, which further enhances spectral resolution & is more in line with human auditory characteristics, making it particularly suitable for capturing frequency features of complex audio signals.

– Việc chuyển soạn piano tự động trước đây dựa trên sự kết hợp giữa các kỹ thuật xử lý tín hiệu truyền thống & phương pháp ML. Các phương pháp tiếp cận ban đầu thường sử dụng các phương pháp như:

1. **Biến đổi Fourier & phân tích phổ.** Các kỹ thuật này phân tích thông tin miền tần số để xác định các nốt riêng lẻ trong tín hiệu âm thanh. Mặc dù hiệu quả đối với các bản ghi đơn âm hoặc đa âm đơn giản, hiệu suất của chúng giảm sút trong các bối cảnh đa âm phức tạp. Mô hình Markov Ẩn (HMM): HMM đã được sử dụng để mô hình hóa các phụ thuộc thời gian & tuần tự trong âm nhạc. Chúng đòi hỏi kỹ thuật đặc trưng miền cụ thể đáng kể & gặp khó khăn với mức độ đa âm cao.
2. **Kỹ thuật phân tích ma trận.** Phân tích ma trận không âm (NMF) đã được sử dụng để phân tích tín hiệu âm thanh thành các thành phần cấu thành, với mỗi thành phần lý tưởng tương ứng với 1 nốt piano. Tuy nhiên, các phương pháp này thường yêu cầu điều chỉnh thủ công & thiếu tính mạnh mẽ trong các tình huống thực tế.
3. **Mạng nơ-ron sâu (DNN).** Gần đây hơn, các kỹ thuật DL, bao gồm mạng kết nối hoàn toàn & RNN, đã cho thấy nhiều hứa hẹn. Các phương pháp này tận dụng các tập dữ liệu lớn để học các đặc điểm âm thanh phức tạp, thường vượt trội hơn các phương pháp truyền thống.
4. **CNN.** CNN đã nổi lên như 1 công cụ mạnh mẽ để sao chép tự động piano nhờ khả năng nắm bắt các mẫu cục bộ trong biểu diễn phổ của tín hiệu âm thanh. Bằng cách tích chập các hạt nhân trên các đặc điểm đầu vào, CNN trích xuất thông tin có ý nghĩa giúp xác định cao độ, trường độ, & cường độ của các nốt nhạc. Là 1 phương pháp DL, CNN thể hiện khả năng biểu diễn & học mạnh mẽ cho phân loại phụ âm thanh piano. Để tối ưu hóa các tham số tích chập & cấu trúc mạng nơ-ron, Hình 2: Sơ đồ khối của thuật toán trích xuất đặc điểm âm thanh piano & thuật toán khớp. minh họa sơ đồ khối của thuật toán trích xuất đặc điểm âm thanh piano & thuật toán khớp. Trong bài báo này, khi thiết kế CNN được tối ưu hóa để tối ưu hóa đặc tính phổ, kích thước & độ sâu của hạt nhân được tinh chỉnh để phù hợp hơn với phân bố không gian & các đặc tính phổ của các đặc điểm đầu vào. Nghiên cứu tập trung vào phân vùng tần số & tối ưu hóa việc chia sẻ hạt nhân tích chập, kết hợp các kỹ thuật từ hệ thống phiên âm nhạc để cho phép mạng lưới thu thập hiệu quả hơn & sử dụng các đặc điểm âm thanh piano trên các dải tần số khác nhau. MFCC là 1 kỹ thuật trích xuất các đặc điểm bằng cách mô phỏng phản ứng phi tuyến tính của hệ thống thính giác con người đối với tần số. Quá trình tính toán bao gồm đóng khung tín hiệu, biến đổi Fourier nhanh, xử lý ngân hàng bộ lọc Mel, phép toán logarit, & biến đổi cosin rời rạc (DCT). CFCC dựa trên mô hình lọc ốc tai, giúp tăng cường hơn nữa độ phân giải phổ & phù hợp hơn với đặc điểm thính giác của con người, khiến nó đặc biệt phù hợp để thu thập các đặc điểm tần số của tín hiệu âm thanh phức tạp.

- 3.2. Optimization algorithm of network model & design of AMT algorithm. Proposed method distinguishes itself from existing approaches by introducing a frequency-based locally optimized convolutional kernel, which captures unique spectral characteristics of piano audio across varying frequency bands. Unlike conventional CNN models that utilize globally shared convolutional kernels, this design emphasizes variability of piano signals across different frequency regions. Prior works, e.g. those using RNNs or simple Mel-frequency cepstral coefficients (MFCCs) as input features, have largely overlooked intricate frequency-specific information in piano audio. Inclusion of cochlear filter cepstral coefficients (CFCCs) further enhances feature extraction process, mimicking human auditory system's sensitivity to frequency variations. This dual-feature extraction mechanism – combining MFCCs & CFCCs – provides a richer representation of piano audio signals. Furthermore, this study integrates spectral envelope analysis through techniques like wavelet & Hilbert transforms, allowing for precise separation of formant features from fine spectral details. This approach builds on foundational work of previous researchers while addressing their limitations in fine-grained spectral analysis. By tailoring convolutional kernels to distinct frequency regions, proposed method offers a novel perspective on automatic transcription task. These techniques not only enhance network's performance in subclassifying converted music but also increase its effectiveness in handling complex piano audio data. In design of optimized CNN based on spectral conversion characteristics, further refinements & adjustments were made to traditional AlexNet architecture to improve its capacity to process spectral features of music signals more effectively. Fig. 3: CNN model training loss assessment diagram. illustrates training loss evaluation of CNN model. This approach not only enhances theoretical understanding of applying convolutional networks in piano audio processing but also offers innovative insights & methods for advancing music information processing & intelligent music recommendation systems. With ongoing technological advancements & improved techniques, optimized CNN based on spectral conversion characteristics is poised to demonstrate a wider range of applications & potential in fields of spectral music conversion & audio signal processing.

– Thuật toán tối ưu hóa mô hình mạng & thiết kế thuật toán AMT. Phương pháp đề xuất khác biệt so với các phương pháp hiện có nhờ việc giới thiệu 1 hạt nhân tích chập được tối ưu hóa cục bộ dựa trên tần số, giúp nắm bắt các đặc điểm phổ độc đáo của âm thanh piano trên các dải tần số khác nhau. Không giống như các mô hình CNN thông thường sử dụng các hạt nhân tích chập được chia sẻ toàn cục, thiết kế này nhấn mạnh vào tính biến thiên của tín hiệu piano trên các vùng tần số khác nhau. Các nghiên cứu trước đây, ví dụ như những nghiên cứu sử dụng mạng nơ-ron nhân tạo (RNN) hoặc các hệ số cepstral tần số Mel (MFCC) đơn giản làm đặc trưng đầu vào, phần lớn đã bỏ qua thông tin tần số cụ thể phức tạp trong âm thanh piano. Việc đưa vào các hệ số cepstral bộ lọc ốc tai (CFCC) giúp tăng cường hơn nữa quá trình trích xuất đặc trưng, mô phỏng độ nhạy của hệ thống thính giác con người đối với các biến đổi tần số. Cơ chế trích xuất đặc trưng kép này – kết hợp MFCC & CFCC – mang lại khả năng biểu diễn tín hiệu âm thanh piano phong phú hơn. Hơn nữa, nghiên cứu này tích hợp phân tích bao phổ thông qua các kỹ thuật như biến đổi wavelet & Hilbert, cho phép tách chính xác các đặc trưng formant khỏi các chi tiết phổ tinh tế. Phương pháp này dựa trên nền tảng công trình của các nhà nghiên cứu

trước đây, đồng thời giải quyết những hạn chế của họ trong phân tích phổ chi tiết. Bằng cách điều chỉnh các hạt nhân tích chập theo các vùng tần số riêng biệt, phương pháp được đề xuất mang đến 1 góc nhìn mới về tác vụ phiên mã tự động. Các kỹ thuật này không chỉ nâng cao hiệu suất của mạng trong việc phân loại phụ âm nhạc đã chuyển đổi mà còn tăng hiệu quả của nó trong việc xử lý dữ liệu âm thanh piano phức tạp. Trong quá trình thiết kế CNN được tối ưu hóa dựa trên các đặc điểm chuyển đổi phổ, kiến trúc AlexNet truyền thống đã được tinh chỉnh & điều chỉnh thêm để cải thiện khả năng xử lý các đặc điểm phổ của tín hiệu âm nhạc hiệu quả hơn. Hình 3: Sơ đồ đánh giá tổn thất huấn luyện mô hình CNN. minh họa việc đánh giá tổn thất huấn luyện của mô hình CNN. Phương pháp này không chỉ nâng cao hiểu biết lý thuyết về việc áp dụng mạng tích chập trong xử lý âm thanh piano mà còn cung cấp những hiểu biết sâu sắc & phương pháp cải tiến để thúc đẩy xử lý thông tin âm nhạc & hệ thống đề xuất âm nhạc thông minh. Với những tiến bộ công nghệ liên tục & các kỹ thuật được cải tiến, CNN được tối ưu hóa dựa trên các đặc điểm chuyển đổi phổ sẵn sàng thể hiện phạm vi ứng dụng rộng hơn & tiềm năng trong các lĩnh vực chuyển đổi âm nhạc phổ & xử lý tín hiệu âm thanh.

Proposed method distinguishes itself from existing approaches by introducing a frequency-based locally optimized convolutional kernel, which captures unique spectral characteristics of piano audio across varying frequency bands. Unlike conventional CNN models that utilize globally shared convolutional kernels, this design emphasizes variability of piano signals across different frequency regions. Prior works, e.g. those using RNNs or simple Mel-frequency cepstral coefficients (MFCCs) as input features, have largely overlooked intricate frequency-specific information in piano audio. Inclusion of cochlear filter cepstral coefficients (CFCCs) further enhances feature extraction process, mimicking human auditory system's sensitivity to frequency variations. This dual-feature extraction mechanism – combining MFCCs & CFCCs – provides a richer representation of piano audio signals. Furthermore, this study integrates spectral envelope analysis through techniques like wavelet & Hilbert transforms, allowing for precise separation of formant features from fine spectral details. This approach builds on foundational work of previous researchers while addressing their limitations in fine-grained spectral analysis. By tailoring convolutional kernels to distinct frequency regions, proposed method offers a novel perspective on automatic transcription task. By incorporating this novel design, network enhances its ability to analyze & understand complex nature of spectral conversion in music signals. This approach not only broadens theoretical application of optimized CNNs in piano audio processing but also opens new avenues for further research & practical application in areas e.g. converted music information processing & intelligent music recommendation systems. Fig. 4: Piano audio music characteristic evaluation diagram presents a characteristic evaluation diagram of piano audio music. As technology advances & methods are refined, optimized CNN, designed based on spectral conversion characteristics, demonstrates promising potential for applications in spectral conversion music & audio processing.

– Phương pháp đề xuất khác biệt so với các phương pháp hiện có nhờ việc giới thiệu 1 hạt nhân tích chập được tối ưu hóa cục bộ dựa trên tần số, giúp nắm bắt các đặc điểm phổ độc đáo của âm thanh piano trên các dải tần số khác nhau. Không giống như các mô hình CNN thông thường sử dụng các hạt nhân tích chập được chia sẻ toàn cục, thiết kế này nhấn mạnh vào tính biến thiên của tín hiệu piano trên các vùng tần số khác nhau. Các nghiên cứu trước đây, ví dụ như những nghiên cứu sử dụng mạng nơ-ron nhân tạo (RNN) hoặc các hệ số cepstral tần số Mel đơn giản (MFCC) làm đặc trưng đầu vào, phần lớn đã bỏ qua thông tin tần số cụ thể phức tạp trong âm thanh piano. Việc đưa vào các hệ số cepstral bộ lọc ốc tai (CFCC) giúp tăng cường hơn nữa quá trình trích xuất đặc trưng, mô phỏng độ nhạy của hệ thống thính giác con người đối với các biến đổi tần số. Cơ chế trích xuất đặc trưng kép này – kết hợp MFCC & CFCC – mang lại khả năng biểu diễn tín hiệu âm thanh piano phong phú hơn. Hơn nữa, nghiên cứu này tích hợp phân tích bao phổ thông qua các kỹ thuật như biến đổi wavelet & Hilbert, cho phép tách chính xác các đặc trưng formant khỏi các chi tiết phổ tinh tế. Phương pháp này dựa trên công trình nền tảng của các nhà nghiên cứu trước đây, đồng thời giải quyết những hạn chế của họ trong phân tích phổ chi tiết. Bằng cách điều chỉnh các hạt nhân tích chập theo các vùng tần số riêng biệt, phương pháp được đề xuất mang đến 1 góc nhìn mới về tác vụ phiên mã tự động. Bằng cách kết hợp thiết kế mới này, mạng lưới nâng cao khả năng phân tích & hiểu bản chất phức tạp của chuyển đổi phổ trong tín hiệu âm nhạc. Cách tiếp cận này không chỉ mở rộng ứng dụng lý thuyết của CNN tối ưu hóa trong xử lý âm thanh piano mà còn mở ra những hướng nghiên cứu mới & ứng dụng thực tế trong các lĩnh vực như xử lý thông tin âm nhạc đã chuyển đổi & hệ thống đề xuất âm nhạc thông minh. Hình 4: Sơ đồ đánh giá đặc tính âm nhạc piano trình bày sơ đồ đánh giá đặc tính của âm nhạc piano. Khi công nghệ tiến bộ & các phương pháp được tinh chỉnh, CNN tối ưu hóa, được thiết kế dựa trên các đặc tính chuyển đổi phổ, cho thấy tiềm năng đầy hứa hẹn cho các ứng dụng trong âm nhạc chuyển đổi phổ & xử lý âm thanh.

- 4. Design & implementation of piano audio AMT algorithm based on CNN. Dataset employed in this study comprises a carefully curated collection of high-quality piano recordings spanning various genres, tempos, & playing styles. Recordings were sourced from publicly available datasets, including MAPs dataset & portions of MusicNet database, supplemented by newly recorded samples to ensure comprehensive coverage of modern piano music. p. 7+++
- 5. Experimental analysis.
- 6. Conclusion.

1.6 [Mül15; Mül21]. Meinard Müller. Fundamentals of Music Processing: Using Python & Jupyter Notebooks. 2e

[91 citations]

- Amazon review.

- Preface to 2e. When writing 1e of this book, my motivation: provide a textbook on emerging fields of music processing & music information retrieval (MIR) with a focus on audio signal processing. Using well-established music analysis & retrieval topics as motivating application scenarios, book introduces fundamental techniques & algorithms relevant for general courses in various fields, including CS, multimedia engineering, information science, & digital humanities. Book is intended for Master & advanced Bachelor students in these fields as well as any reader interested in delving into field of music processing (& not being frightened by some mathematics). While providing profound technological knowledge as well as a comprehensive treatment of music processing applications, book also includes numerous examples & illustrations to convey main ideas in an intuitive fashion. In recent years, suitably designed software packages & freely accessible web-based frameworks have made education in CS & signal processing more interactive. Such novel technology allows for designing courses that aid students in moving from recalling & reciting theoretical concepts towards comprehension & application.

  – Khi viết 1e của cuốn sách này, động lực của tôi: cung cấp 1 giáo trình về các lĩnh vực mới nổi của xử lý âm nhạc & truy xuất thông tin âm nhạc (MIR) tập trung vào xử lý tín hiệu âm thanh. Sử dụng các chủ đề phân tích âm nhạc & truy xuất đã được thiết lập tốt làm các tình huống ứng dụng thúc đẩy, cuốn sách giới thiệu các kỹ thuật cơ bản & thuật toán có liên quan đến các khóa học chung trong nhiều lĩnh vực, bao gồm CS, kỹ thuật đa phương tiện, khoa học thông tin, & nhân văn kỹ thuật số. Cuốn sách dành cho sinh viên Thạc sĩ & Cử nhân nâng cao trong các lĩnh vực này cũng như bất kỳ độc giả nào quan tâm đến việc nghiên cứu sâu về lĩnh vực xử lý âm nhạc (& không sợ 1 số môn toán). Trong khi cung cấp kiến thức công nghệ sâu sắc cũng như cách xử lý toàn diện các ứng dụng xử lý âm nhạc, cuốn sách cũng bao gồm nhiều ví dụ & minh họa để truyền đạt các ý chính theo cách trực quan. Trong những năm gần đây, các gói phần mềm được thiết kế phù hợp & các khuôn khổ dựa trên web có thể truy cập miễn phí đã làm cho giáo dục về CS & xử lý tín hiệu trở nên tương tác hơn. Công nghệ mới lạ như vậy cho phép thiết kế các khóa học hỗ trợ sinh viên chuyển từ việc nhớ lại & đọc thuộc lòng các khái niệm lý thuyết sang hiểu & ứng dụng.

  These new developments are precisely motivation for 2e of this book. It extends 1e by providing additional material (called *FMP Notebooks*), yielding an interactive foundation for teaching & learning fundamentals of music processing (FMP). FMP notebooks are built upon Jupyter notebook framework, which has became a standard in educational settings. This opensource web application allows users to create documents that contain executable code, text-based information, mathematical formulas, plots, images, sound examples, & videos. By leveraging Jupyter framework, FMP notebooks bridge gap between theory & practice by interleaving technical concepts, mathematical details, code examples, illustrations, & sound examples within a unifying setting. FMP notebooks closely follow 8 chaps of textbook &, as such, provide an explicit link between structured educational environments & current professional practices, in line with current curricular recommendations for CS.

  – Những phát triển mới này chính xác là động lực cho phiên bản 2 của cuốn sách này. Phiên bản này mở rộng phiên bản 1 bằng cách cung cấp thêm tài liệu (gọi là *FMP Notebooks*), tạo ra nền tảng tương tác để giảng dạy & học các nguyên tắc cơ bản của xử lý âm nhạc (FMP). Sổ tay FMP được xây dựng dựa trên khuôn khổ sổ tay Jupyter, vốn đã trở thành tiêu chuẩn trong các thiết lập giáo dục. Ứng dụng web nguồn mở này cho phép người dùng tạo các tài liệu có chứa mã thực thi, thông tin dạng văn bản, công thức toán học, sơ đồ, hình ảnh, ví dụ âm thanh, & video. Bằng cách tận dụng khuôn khổ Jupyter, sổ tay FMP thu hẹp khoảng cách giữa lý thuyết & thực hành bằng cách đan xen các khái niệm kỹ thuật, chi tiết toán học, ví dụ mã, hình minh họa, & ví dụ âm thanh trong 1 thiết lập thống nhất. Sổ tay FMP bám sát 8 chương của sách giáo khoa &, do đó, cung cấp 1 liên kết rõ ràng giữa các môi trường giáo dục có cấu trúc & các hoạt động chuyên môn hiện tại, phù hợp với các khuyến nghị về chương trình giảng dạy hiện tại cho CS.

  1 primary purpose of FMP notebooks: provide audio-visual material & Python code examples that implement computational approaches step by step. Additionally, FMP notebooks yield an interactive framework that allows students to experiment with their music examples, explore effect of parameter settings, & understand computed results by suitable visualizations & sonifications. When teaching & learning music processing, essential to have a holistic view of MIR task at hand, algorithmic approach, & its practical implementation. Looking at all steps of processing pipeline sheds light on input data & its biases, possible violations of model assumptions, & shortcomings of quantitative evaluation measures. Only by an interactive examination of all these aspects will acquire students a deeper understanding of concepts, transitioning from merely understanding concepts to applying their music processing approaches both conceptually & in code.

  – 1 mục đích chính của sổ tay FMP: cung cấp tài liệu nghe nhìn & ví dụ mã Python triển khai các phương pháp tính toán từng bước. Ngoài ra, sổ tay FMP tạo ra 1 khuôn khổ tương tác cho phép học sinh thử nghiệm các ví dụ về âm nhạc của mình, khám phá hiệu ứng của các cài đặt tham số, & hiểu các kết quả tính toán bằng hình ảnh hóa phù hợp & âm thanh hóa. Khi dạy & học xử lý âm nhạc, điều cần thiết là phải có cái nhìn toàn diện về nhiệm vụ MIR trong tay, phương pháp tiếp cận thuật toán, & triển khai thực tế của nó. Xem xét tất cả các bước của quy trình xử lý sẽ làm sáng tỏ dữ liệu đầu vào & các thành kiến của nó, các vi phạm có thể xảy ra đối với các giả định của mô hình, & những thiếu sót của các biện pháp đánh giá định lượng. Chỉ bằng cách kiểm tra tương tác tất cả các khía cạnh này, học sinh mới có thể hiểu sâu hơn về các khái niệm, chuyển từ việc chỉ hiểu các khái niệm sang áp dụng các phương pháp xử lý âm nhạc của họ cả về mặt khái niệm & trong mã.

  Main body of FMP notebooks consists of 8 parts, structured along with 8 chaps of this textbook. In book's 2e, provide at end of each chap an additional sect titled *FMP Notebooks*. These sects serve 2 purposes. 1st, give a comprehensive guide by systematically describing content & purpose of all notebooks related to corresponding chap. As a 2nd objective, make concrete suggestions on using FMP notebooks to create an enriching, interactive, & interdisciplinary supplement in form of experiments & advanced studies in a music processing curriculum. Textbook's guide can be best appreciated & understood when FMP notebooks run in a browser simultaneously while reading.

  – Nội dung chính của sổ tay FMP bao gồm 8 phần, được cấu trúc cùng với 8 chương của sách giáo khoa này. Trong cuốn sách 2e, cung cấp ở cuối mỗi chương 1 phần bổ sung có tiêu đề *Sổ tay FMP*. Các phần này phục vụ 2 mục đích. 1, cung cấp hướng

dẫn toàn diện bằng cách mô tả 1 cách có hệ thống nội dung & mục đích của tất cả các sổ tay liên quan đến chương tương ứng. Với mục tiêu thứ 2, đưa ra các đề xuất cụ thể về việc sử dụng sổ tay FMP để tạo ra 1 phần bổ sung phong phú, tương tác, & liên ngành dưới dạng các thí nghiệm & các nghiên cứu nâng cao trong chương trình giảng dạy xử lý âm nhạc. Hướng dẫn của sách giáo khoa có thể được đánh giá cao & hiểu rõ nhất khi sổ tay FMP chạy trong trình duyệt đồng thời trong khi đọc.

FMP notebooks are publicly available under a Creative Commons license at https://www.audiolabs-erlangen.de/FMP in form of Jupyter notebooks as well as HTML exports, which can be accessed through a conventional web browser. Using static HTML version, all multimedia material, including music examples, audio files, video files, & mages, can be directly accessed without any specific technical requirements beyond a standard web browser. To run FMP notebook's code, one needs to install Python, Jupyter, & addition Python packages. All necessary steps for installing, running, & updating required software packages are described in a separate part (called Part B) of FMP notebooks. This part also contains short introductions to Python programming, Jupyter notebooks, multimedia integration, as well as data annotation, visualization, & sonification. Rather than being comprehensive, Part B gives instructive code examples that become relevant in other parts & documents how FMP notebooks were created.

– Sổ tay FMP được cung cấp công khai theo giấy phép Creative Commons tại https://www.audiolabs-erlangen.de/FMP dưới dạng sổ tay Jupyter cũng như xuất HTML, có thể truy cập thông qua trình duyệt web thông thường. Sử dụng phiên bản HTML tĩnh, tất cả tài liệu đa phương tiện, bao gồm các ví dụ về nhạc, tệp âm thanh, tệp video, & mages, có thể được truy cập trực tiếp mà không cần bất kỳ yêu cầu kỹ thuật cụ thể nào ngoài trình duyệt web tiêu chuẩn. Để chạy mã sổ tay FMP, người ta cần cài đặt Python, Jupyter, & các gói Python bổ sung. Tất cả các bước cần thiết để cài đặt, chạy, & cập nhật các gói phần mềm cần thiết đều được mô tả trong 1 phần riêng (gọi là Phần B) của sổ tay FMP. Phần này cũng bao gồm các phần giới thiệu ngắn về lập trình Python, sổ tay Jupyter, tích hợp đa phương tiện cũng như chú thích dữ liệu, trực quan hóa, & âm thanh hóa. Thay vì toàn diện, Phần B đưa ra các ví dụ mã hướng dẫn có liên quan trong các phần khác & ghi lại cách tạo sổ tay FMP.

Besides its substantial extensions through FMP notebooks, another major change in 2e: thorough revision of sects called *Summary & Further Readings* (previously called *Further Notes* in 1e). These sects have been streamlined, now containing more compact & focused summaries. Furthermore, references & thinks to literature for further readings have been revised & updated. Rather than providing an extensive literature review, have deliberately limited ourselves to citing only selected core literature & overview articles, where one can find further pointers to relevant & more advanced work. As in general multimedia processing, many recent advances in music processing have been driven by techniques based on DL. E.g., DL-based techniques have led to significant improvements for many tasks e.g. music source separation, music transcription, chord recognition, melody estimation, beat tracking, tempo estimation, & lyrics alignment, to name a few. In particular, major improvements could be achieved for music scenarios where sufficient training data is available. A particular strength of DL-based approaches is their ability to extract complex features directly from raw audio data, which can then be used to make predictions based on hidden structures & relations. Furthermore, powerful software packages allow for easily designing, implementing, & experimenting with ML algorithms based on deep neural networks (DNNs). Covering fast-growing & dynamic field of DL goes beyond scope of this textbook. Instead, focus on classical signal & music processing techniques, yielding fundamental insights into problem at hand & providing explicit baseline approaches one may (& should) compare against when exploring more powerful yet often difficult-to-interpret DNN-based learning approaches. For further readings, provided links to selected references that apply recent DNN-based techniques to music processing. Hope: these references help students & researchers transition from model-based approaches as introduced in this textbook to world of DL applied to specific music processing tasks. Our literature choice is undoubtedly subjective, & would like to apologize to all those whose work we have not mentioned or adequately appreciated.

– Bên cạnh những phần mở rộng đáng kể thông qua sổ tay FMP, 1 thay đổi lớn khác trong 2e: sửa đổi toàn diện các giáo phái được gọi là *Tóm tắt & Đọc thêm* (trước đây gọi là *Ghi chú thêm* trong 1e). Các giáo phái này đã được sắp xếp hợp lý, hiện chứa các bản tóm tắt & tập trung hơn. Hơn nữa, các tài liệu tham khảo & cho các bài đọc thêm đã được sửa đổi & cập nhật. Thay vì cung cấp 1 bài đánh giá tài liệu mở rộng, chúng tôi đã cố tình giới hạn bản thân chỉ trích dẫn 1 số bài viết tổng quan & tài liệu cốt lõi đã chọn, nơi người ta có thể tìm thấy thêm các gợi ý cho công việc & nâng cao hơn có liên quan. Giống như trong xử lý đa phương tiện nói chung, nhiều tiến bộ gần đây trong xử lý âm nhạc đã được thúc đẩy bởi các kỹ thuật dựa trên DL. E.g., các kỹ thuật dựa trên DL đã dẫn đến những cải tiến đáng kể cho nhiều tác vụ, ví dụ như tách nguồn nhạc, phiên âm nhạc, nhận dạng hợp âm, ước tính giai điệu, theo dõi nhịp, ước tính nhịp độ, & căn chỉnh lời bài hát, v.v. Đặc biệt, có thể đạt được những cải tiến lớn đối với các tình huống âm nhạc khi có đủ dữ liệu đào tạo. Điểm mạnh đặc biệt của các phương pháp tiếp cận dựa trên DL là khả năng trích xuất các tính năng phức tạp trực tiếp từ dữ liệu âm thanh thô, sau đó có thể được sử dụng để đưa ra dự đoán dựa trên các cấu trúc ẩn & quan hệ. Hơn nữa, các gói phần mềm mạnh mẽ cho phép dễ dàng thiết kế, triển khai, & thử nghiệm các thuật toán ML dựa trên mạng nơ-ron sâu (DNN). Việc bao phủ lĩnh vực DL đang phát triển nhanh & năng động vượt ra ngoài phạm vi của sách giáo khoa này. Thay vào đó, hãy tập trung vào các kỹ thuật xử lý tín hiệu & âm nhạc cổ điển, mang lại những hiểu biết cơ bản về vấn đề đang gặp phải & cung cấp các phương pháp tiếp cận cơ sở rõ ràng mà người ta có thể (& nên) so sánh khi khám phá các phương pháp tiếp cận học tập dựa trên DNN mạnh mẽ hơn nhưng thường khó diễn giải. Để đọc thêm, hãy cung cấp các liên kết đến các tài liệu tham khảo đã chọn áp dụng các kỹ thuật dựa trên DNN gần đây vào xử lý âm nhạc. Hy vọng: các tài liệu tham khảo này giúp sinh viên & các nhà nghiên cứu chuyển đổi từ các phương pháp tiếp cận dựa trên mô hình như được giới thiệu trong sách giáo khoa này sang thế giới DL được áp dụng cho các tác vụ xử lý âm nhạc cụ thể. Lựa chọn tài liệu của chúng tôi chắc chắn là chủ quan, & muốn xin lỗi tất cả những người mà chúng tôi chưa đề cập hoặc đánh giá đầy đủ về công trình của họ.

- Preface to 1e. Music is a ubiquitous & vital part of lives of billions of people worldwide. Musical creations & performances are

amongst most complex & intricate of our cultural artifacts, & emotional power of music can touch us in surprising & profound ways. Music spans an enormous range of forms & styles, from simple, unaccompanied folk songs, to popular & jazz music, to symphonies for full orchestras. Digital revolution in music distribution & storage has simultaneously fueled tremendous interest in & attention to ways that information technology can be applied to this kind of content. From browsing personal collections, to discovering new artists, to managing & protecting rights of music creators, computers are now deeply involved in almost every aspect of music consumption, not to mention their vital role in much of today's music production.

– Âm nhạc là 1 phần thiết yếu & phổ biến trong cuộc sống của hàng tỷ người trên toàn thế giới. Các sáng tác âm nhạc & biểu diễn là 1 trong những hiện vật văn hóa phức tạp & tinh vi nhất của chúng ta, & sức mạnh cảm xúc của âm nhạc có thể chạm đến chúng ta theo những cách đáng ngạc nhiên & sâu sắc. Âm nhạc trải dài trên nhiều hình thức & phong cách, từ những bài hát dân gian đơn giản, không có nhạc đệm, đến nhạc & jazz phổ biến, đến các bản giao hưởng cho dàn nhạc đầy đủ. Cuộc cách mạng kỹ thuật số trong phân phối & lưu trữ âm nhạc đồng thời thúc đẩy sự quan tâm to lớn & chú ý đến những cách mà công nghệ thông tin có thể được áp dụng cho loại nội dung này. Từ việc duyệt qua các bộ sưu tập cá nhân, đến việc khám phá các nghệ sĩ mới, đến việc quản lý & bảo vệ quyền của những người sáng tạo âm nhạc, máy tính hiện đang tham gia sâu sắc vào hầu hết mọi khía cạnh của việc tiêu thụ âm nhạc, chưa kể đến vai trò quan trọng của chúng trong phần lớn hoạt động sản xuất âm nhạc ngày nay.

Despite importance of music, *music processing* is still a relatively young discipline compared with speech processing, a research field with a long tradition. A research community represented by International Society for Music Information Retrieval (ISMIR), which systematically deals with a wide range of computer-based music analysis, processing, & retrieval topics, was formed in 2000. Traditionally, computer-based music research has mostly been conducted on basis of symbolic representations using music notation or MIDI representations. Because of increasing availability of digitized audio material & an explosion of computing power, automated processing of waveform-based audio signals is now increasingly in focus of research efforts.

– Mặc dù âm nhạc có tầm quan trọng, *music processing* vẫn là 1 ngành tương đối trẻ so với xử lý giọng nói, 1 lĩnh vực nghiên cứu có truyền thống lâu đời. 1 cộng đồng nghiên cứu do Hiệp hội quốc tế về truy xuất thông tin âm nhạc (ISMIR) đại diện, chuyên xử lý 1 cách có hệ thống nhiều chủ đề phân tích, xử lý, & truy xuất âm nhạc dựa trên máy tính, đã được thành lập vào năm 2000. Theo truyền thống, nghiên cứu âm nhạc dựa trên máy tính chủ yếu được tiến hành trên cơ sở các biểu diễn ký hiệu sử dụng ký hiệu âm nhạc hoặc biểu diễn MIDI. Do tính khả dụng ngày càng tăng của tài liệu âm thanh được số hóa & sự bùng nổ của sức mạnh tính toán, việc xử lý tự động các tín hiệu âm thanh dựa trên dạng sóng hiện đang ngày càng trở thành trọng tâm của các nỗ lực nghiên cứu.

Many of these research efforts are directed towards development of technologies that allow users to access & explore music in all its different facets. E.g., audio fingerprinting techniques are nowadays integrated into commercial products that help users automatically identify songs they hear. Music processing techniques are used in extended audio players that highlight current measures within sheet music while playing back a recording of a symphony. On demand, additional information about melodic & harmonic progressions or rhythm & tempo is automatically represented to listener. Interactive music interfaces display structural parts of current piece of music & allow users to directly jump to any sect e.g. chorus, main musical theme, or a solo sect without tedious fast-forwarding & rewinding. Furthermore, listeners are equipped with Google-like search engines that enable them to explore large music collections in various ways. E.g., user may create a query by specifying a certain note constellation, or some harmonic or rhythmic pattern by whistling a melody or tapping a rhythm, or simply by selecting a short passage from an audio recording; system then provides user with a ranked list of available music excerpts from collection that are musically related to query. In music processing, 1 main objective: contribute concepts, models, algorithms, implementations, & evaluations for tackling such types of analysis & retrieval problems.

– Nhiều nỗ lực nghiên cứu này hướng đến phát triển các công nghệ cho phép người dùng truy cập & khám phá âm nhạc ở mọi khía cạnh khác nhau của nó. E.g., các kỹ thuật lấy dấu vân tay âm thanh hiện nay được tích hợp vào các sản phẩm thương mại giúp người dùng tự động xác định các bài hát họ nghe. Các kỹ thuật xử lý âm nhạc được sử dụng trong các trình phát âm thanh mở rộng làm nổi bật các nhịp điệu hiện tại trong bản nhạc trong khi phát lại bản ghi âm của 1 bản giao hưởng. Theo yêu cầu, thông tin bổ sung về tiến trình giai điệu & hòa âm hoặc nhịp điệu & nhịp độ sẽ tự động được thể hiện cho người nghe. Giao diện âm nhạc tương tác hiển thị các phần cấu trúc của bản nhạc hiện tại & cho phép người dùng trực tiếp chuyển đến bất kỳ phần nào ví dụ như điệp khúc, chủ đề âm nhạc chính hoặc phần độc tấu mà không cần tua nhanh & tua lại. Hơn nữa, người nghe được trang bị các công cụ tìm kiếm giống như Google cho phép họ khám phá các bộ sưu tập nhạc lớn theo nhiều cách khác nhau. Ví dụ: người dùng có thể tạo truy vấn bằng cách chỉ định 1 chòm sao nốt nhạc nhất định hoặc 1 số mẫu hòa âm hoặc nhịp điệu bằng cách huýt sáo 1 giai điệu hoặc gõ 1 nhịp điệu hoặc chỉ cần chọn 1 đoạn ngắn từ bản ghi âm; hệ thống sau đó cung cấp cho người dùng danh sách xếp hạng các trích đoạn nhạc có sẵn từ bộ sưu tập có liên quan đến truy vấn về mặt âm nhạc. Trong xử lý âm nhạc, 1 mục tiêu chính: đóng góp các khái niệm, mô hình, thuật toán, triển khai, & đánh giá để giải quyết các loại vấn đề phân tích & truy xuất như vậy.

This textbook is devoted to emerging fields of music processing & music information retrieval (MIR) – interdisciplinary research ares which are related to various disciplines including signal processing, information retrieval, ML, multimedia engineering, library science, musicology, & digital humanities. Main goal of this book: give an introduction to this vibrant & exciting new research area for a wide readership. Well-established topics in music analysis & retrieval have been selected to serve as motivating application scenarios. Within these scenarios, fundamental techniques & algorithms that are applicable to a wide range of analysis & retrieval problems are presented in depth.

– Sách giáo khoa này dành cho các lĩnh vực mới nổi về xử lý âm nhạc & truy xuất thông tin âm nhạc (MIR) – các lĩnh vực nghiên cứu liên ngành liên quan đến nhiều ngành khác nhau bao gồm xử lý tín hiệu, truy xuất thông tin, ML, kỹ thuật đa

phương tiện, khoa học thư viện, âm nhạc học, & nhân văn kỹ thuật số. Mục tiêu chính của cuốn sách này: giới thiệu về lĩnh vực nghiên cứu mới sôi động & thú vị này cho nhiều độc giả. Các chủ đề đã được xác lập rõ ràng trong phân tích âm nhạc & truy xuất đã được lựa chọn để làm các kịch bản ứng dụng thúc đẩy. Trong các kịch bản này, các kỹ thuật cơ bản & thuật toán có thể áp dụng cho nhiều vấn đề phân tích & truy xuất được trình bày chi tiết.

This book is meant to be a *textbook* that is suitable for courses at advanced undergraduate & beginning master level. By mixing theory & practice, book provides both deep technological knowledge as well as a comprehensive treatment of music processing applications. Furthermore, by including numerous examples, illustrations (book contains > 300 figures), & exercises, hope book provides interesting material for courses in various fields e.g. CS, multimedia engineering, information science, & digital humanities.

Subsequent sects of this preface contain further information on overall structure of book, interconnections between various topics & techniques, & suggestions on how this book may be used as a basis for different courses. 1st give an overview of book's content by quickly going through individual chaps. Then, explain various ways of reading & using book, each time focusing on a different aspect. Start with view of a lecturer who wants to use this textbook as a basis for an introductory course in music processing or music information retrieval. Then, show how book may be used for an introductory course on Fourier analysis & its applications. Finally, assume view of a computer scientist who wants to teach fundamental issues on data representations & algorithms, where music may serve as an underlying application domain. Describing these different views, try to work out dependencies between chaps as well as conceptual relationships between various music processing tasks.

- **Content.** This textbook consists of 8 chaps. 1st 2 chaps cover fundamental material on music representations & Fourier transform – concepts that are required throughout book. These 2 chaps make book self-contained to a great extent. In subsequent chaps, concrete music processing tasks serve as starting points for our investigations. Each of these chaps is organized in a similar fashion. A chap starts with a general description of music processing scenario at hand & integrates topic into a wider context. Motivated by scenario at hand, each chap discusses important techniques & algorithms that are generally applicable to a wide range of analysis, classification, & retrieval problems. All these techniques are treated in a mathematically rigorous way. At same time, techniques are immediately applied to a concrete music processing task. By mixing theory & practice, book's goal: convey both profound technological knowledge as well as a solid understanding of music processing applications. Each of chaps ends with a sect that includes links to research literature, hints for further reading, a list of references, & exercises. Before discuss how this textbook may be employed in a course or used for self-study, 1st give an overview of individual chaps & main topics.

  [Table: Chap: Music Processing Scenario: Notations, Techniques, & Algorithms.]

  * Chap. 1: Music Representations: Music notation, MIDI, audio signal, waveform, pitch, loudness, timbre
  * Chap. 2: Fourier Analysis of Signals: Discrete/analog signal, sinusoid, exponential, Fourier transform, Fourier representation, DFT, FFT, STFT
  * Chap. 3: Music Synchronization: Chroma feature, dynamic programming, dynamic time warping (DTW), alignment, user interface
  * Chap. 4: Music Structure Analysis: Similarity matrix, repetition, thumbnail, homogeneity, novelty, evaluation, precision, recall, F-measure, visualization, scape plot
  * Chap. 5: Chord Recognition: Harmony, music theory, chords, scales, templates, hidden Markov model (HMM), evaluation
  * Chap. 6: Tempo & Beat Tracking: Onset, novelty, tempo, tempogram, beat, periodicity, Fourier analysis, autocorrelation
  * Chap. 7: Content-Based Audio Retrieval: Identification, fingerprint, indexing, inverted list, matching, version, cover song
  * Chap. 8: Musically Informed Audio Decomposition: Harmonic/percussive component, signal reconstruction, instantaneous frequency, fundamental frequency (F0), trajectory, nonnegative matrix factorization (NMF)

  Musical information can be represented in many different ways. In Chap. 1, consider 3 widely used music representations: sheet music, symbolic, & audio representations. This 1st chap also introduces basic terminology used throughout book. In particular, discuss musical & acoustic properties of audio signals including aspects e.g. frequency, pitch, dynamics, & timbre.

  Important technical terminology is covered in Chap. 2. In particular, approach Fourier transform – which is perhaps most fundamental tool in signal processing – from various perspectives. For reader who is more interested in musical aspects of book, Sect. 2.1 provides a summary of most important facts on Fourier transform. In particular, notion of a spectrogram, which yields a time-frequency representation of an audio signal, is introduced. Remainder of chap treats Fourier transform in greater mathematical depth & also includes fast Fourier transform (FFT) – an algorithm of great beauty & high practical relevance.

  As a 1st music processing task, study in Chap. 3 problem of music synchronization. Objective: temporally align compatible representations of same piece of music. Considering this scenario, explain need for musically informed audio features. In particular, introduce concept of chroma-based music features, which capture properties that are related to harmony & melody. Furthermore, study an alignment technique known as *dynamic time warping* (DTW), a concept that is applicable for analysis of general time series. For its efficient computation, discuss an algorithm based on dynamic programming – a widely used method for solving a complex problem by breaking it down into a collection of simpler subproblems.

  In Chap. 4, address a central & well-researched area within MIR known as music structure analysis. Given a music recording, objective: identify important structural elements & to temporally segment recording according to these elements. Within this scenario, discuss fundamental segmentation principles based on repetitions, homogeneity, & novelty – principles that also apply to other types of multimedia beyond music. As an important technical tool, study in detail concept of self-similarity

matrices & discussed their structural properties. Finally, briefly touch topic of evaluation, introducing notions of precision, recall, & F-measure. These measures are used to compare computed results obtained by an automated procedure with so-called ground truth annotations that are typically generated manually by some domain expert.

In Chap. 5, consider problem of analyzing harmonic properties of a piece of music by determining a descriptive progression of chords from a given audio recording. Take this opportunity to 1st discuss some basic theory of harmony including concepts e.g. intervals, chords, & scales. Then, motivated by automated chord recognition scenario, introduce template-based matching procedures & hidden Markov models – a concept of central important for analysis of temporal patterns in time-dependent data streams including speech, gestures, & music.

Tempo & beat are further fundamental properties of music. In Chap. 6, introduce basic ideas on how to extract tempo-related information from audio recordings. In this scenario, a 1st challenge: locate note onset information – a task that requires methods for detecting changes in energy & spectral content. To derive tempo & beat information, note onset candidates are then analyzed with regard to quasiperiodic patterns. This leads us to study of general methods for local periodicity analysis of time series.

1 important topic in information retrieval is concerned with development of search engines that enable users to explore music collections in a flexible & intuitive way. In Chap. 7, discuss audio retrieval strategies that follow query-by-example paradigm: given an audio query, task: retrieve all documents that are somehow similar or related to query. Starting with audio identification, a technique used in many commercial applications e.g. *Shazam*, study various retrieval strategies to handle different degrees of similarity. Considering efficiency issues, discuss fundamental indexing techniques based on inverted lists – a concept originally used in text retrieval.

In final Chap. 8 on audio decomposition, present a challenging research direction that is closely related to source separation. Within this wide research area, consider 3 subproblems: harmonic-percussive separation, main melody extraction, & score-informed audio decomposition. Within these scenarios, discuss a number of key techniques including instantaneous frequency estimation, fundamental frequency (F0) estimation, spectrogram inversion, & nonnegative matrix factorization (NMF). Encounter a number of acoustic & musical properties of audio recordings that have been introduced & discussed in prev chaps, which rounds off book.

– Trong Chương cuối cùng. 8 về phân tích âm thanh, trình bày 1 hướng nghiên cứu đầy thách thức có liên quan chặt chẽ đến việc tách nguồn. Trong phạm vi nghiên cứu rộng này, hãy xem xét 3 vấn đề phụ: tách hài hòa-bộ gõ, trích xuất giai điệu chính, & phân tích âm thanh dựa trên điểm số. Trong các tình huống này, hãy thảo luận về 1 số kỹ thuật chính bao gồm ước tính tần số tức thời, ước tính tần số cơ bản (F0), đảo ngược phổ, & phân tích ma trận không âm (NMF). Gặp phải 1 số tính chất âm thanh & nhạc của bản ghi âm đã được giới thiệu & thảo luận trong các chương trước, giúp kết thúc cuốn sách.

- Target Readership. In last 15 years, music processing & music information retrieval (MIR) have developed into a vibrant & multidisciplinary area of research. Because of diversity & richness of music, this area brings together researchers & students from a multitude of fields including information science, audio engineering, CS, & musicology. This book's intention: offer interesting material for courses in these fields. Main target groups of this book are Master & advanced Bachelor students. Also hope: researchers who are interested in delving into field of music processing will benefit from this textbook. 8 chaps are organized in a modular fashion, thus offering lecturers & readers many ways to choose, rearrange, or supplement material. In this way, it should be possible to easily integrate selected chaps or individual sects into courses related to general multimedia, information science, signal processing, music informatics, or digital humanities.

  Of course, writing a textbook requires making some choices. Topic selected for this textbook play an important role in music processing & MIR, but they also reflect research areas of author – want to apologize to my colleagues for having ignored many other important topics. Focus of this textbook is not to give a comprehensive overview of music processing, but to provide a solid understanding of concepts introduced within a small number of important application scenarios. Layout, tempo of presentation, & pattern of figures have been kept consistent throughout textbook. Hope this helps lecturers & students to quickly get comfortable with style of presentation & to flexibly use material. In particular, great care has been taken with illustrations. 1 way to approach a new topic: 1st go through all figures of a sect or chap. Not only should this hone one's intuition, but also yield a 1st visual overview of concepts to be studied.

  In following, describe dependencies between chaps & sects by assuming different views on book. Each view focuses on different aspects & may serve as a basis for designing a 1-semester or even 2-semester course (with 2–4 hours weekly per semester plus exercises). Even though views are presented from perspective of a lecturer, hope: also helpful for a student or reader to gain a comprehensive overview & a better understanding of crosslinks between sects & chaps. A more abstract goal of describing different views: highlight general applicability of presented techniques & conceptual relationships between various music processing tasks.

  – Sau đây, hãy mô tả sự phụ thuộc giữa các chương & giáo phái bằng cách giả định các quan điểm khác nhau về cuốn sách. Mỗi quan điểm tập trung vào các khía cạnh khác nhau & có thể dùng làm cơ sở để thiết kế khóa học 1 học kỳ hoặc thậm chí 2 học kỳ (với 2-4 giờ mỗi tuần cho mỗi học kỳ cộng với các bài tập). Mặc dù các quan điểm được trình bày theo quan điểm của giảng viên, hy vọng: cũng hữu ích cho sinh viên hoặc người đọc để có được cái nhìn tổng quan toàn diện & hiểu rõ hơn về các liên kết chéo giữa các giáo phái & chương. 1 mục tiêu trừu tượng hơn là mô tả các quan điểm khác nhau: làm nổi bật khả năng áp dụng chung của các kỹ thuật được trình bày & mối quan hệ khái niệm giữa các tác vụ xử lý âm nhạc khác nhau.

- View: A 1st Course in Music Processing. Start with view of a lecturer who wants to use this textbook as a basis for an introductory course in music processing or music information retrieval. To lay foundation for such a course & to fix important

notions, recommend to begin with Chap. 1 on music representations. By going through Sect. 1.1, student should get an intuitive idea on various attributes of music e.g. notes, pitch, chroma, note length, dynamics, or time signature. Also hope students who are not familiar with Western music notation will benefit from this sect by gaining some intuitive understanding – intricacies of music notation are not required for subsequent chaps. Sect. 1.2 contains background information on symbolic representations. As with sheet music sect, an understanding of all details, e.g., concerning MIDI format or optical music recognition, is not required. These details, however, become important when working with this kind of data in practice. For most tasks & techniques presented in this book, piano-roll representation (Sect. 1.2.1) may serve as an intuitive substitute for sheet music or symbolic representations.

Material on audio representations (Sect. 1.3) is fundamental for a music processing course based on this book. Many notions e.g. waveform, sinusoid, frequency, phase, pitch, harmonic, partial, decibel, timbre, transient, or spectrogram are introduced in a more informal way – concepts that will be revisited in subsequent chaps in more detail.

To make this textbook self-contained & accessible to a wide audience, required tools from signal processing have been confined to a small number of key techniques. Basically all audio processing steps as presented in this book are derived from standard Fourier analysis. Fourier transform becomes main signal processing tool, & a good understanding of this transform is indispensable. In Sect. 2.1, most important facts on Fourier analysis are introduced in a mathematically rigorous, yet compact fashion. Omitting proofs, this sect aims to convey main ideas (using many illustrations & examples), while introducing required technical notions. This sect contains all material required to understand subsequent chaps. For a course with a focus on music processing, recommend to skip remaining sects of Chap. 2 (& to come back to them at a later stage if required). However, Sect. 2.1 should be covered in detail.

Motivated by music synchronization application, Chap. 3 introduces further basic concepts that run like a thread through this book. To make music data comparable & algorithmically accessible, 1st step in most music processing tasks: convert data into suitable feature representations that capture relevant aspects while suppressing irrelevant details. In Sect. 3.1, address issue of converting an audio signal into musically informed feature representations. As our main example, discuss construction of time-chroma representations, which are based on equal-tempered scale. Besides music synchronization, these features play an important role in many other applications including music structure analysis (Chap. 4), chord recognition (Chap. 5), & content-based audio retrieval (Chap. 7).

2nd important concept introduced in Chap. 3 is known as sequence alignment – a general technique for arranging 2 time-dependent sequences to identify regions of similarity. To compute an optimal alignment, there are efficient algorithms based on dynamic programming – a general paradigm for solving a complex problem by breaking it down into a collection of simpler subproblems. In Sect. 3.2, study an alignment technique referred to as dynamic time warping (DTW) as well as an efficient algorithm. In later chaps, encounter similar alignment techniques, e.g., in context of audio thumbnailing (Sect. 4.3), chord recognition (Sect. 5.3), beat tracking (Sect. 6.3), audio matching (Sect. 7.2), & version identification (Sect. 7.3).

While recommend covering fundamental material presented in Chap. 1, Sect. 2.1, Sect. 3.1, & Sect. 3.2 in a course on music processing, there is a lot of freedom on how to proceed afterwards. Remaining chaps are kept mostly independent, excluding a few exceptions that are suitably referenced. 1 possible continuation of a course: cover applications of music synchronization (Sect. 3.3) & then proceed with Chap. 4 on music structure analysis. As opposed to music synchronization, where one compares 2 given sequences, in music structure analysis a single sequence is compared with itself. This leads to notion of self-similarity matrices – a concept related to recurrence plots as used for analysis of general time series. Study of self-similarity matrices yields deep insights into structural properties of music representations as well as into properties of underlying feature representations. By suitably visualizing self-similarity matrices, these aspects can be conveyed in a nontechnical & intuitive fashion. On other hand, automated extraction of musically relevant structures from self-similarity matrices – even if they seem obvious for humans – is anything but a trivial problem. In Chap. 4, various challenges as well as algorithmic approaches are presented.

As an alternative, after having introduced chroma-based audio features (Sect. 3.1), one may directly jump to Chap. 5. Task of automated chord recognition yields a natural motivation for this type of feature. Reason: chroma features capture a signal's short-time tonal content, which is closely correlated to harmonic progression of underlying piece. For a more musically oriented course, Sect. 5.1 provides some background material on harmony theory including concepts e.g. intervals, chords, & scales. In a more technically oriented course, mot of this material may be skipped. One can then directly proceed with classification approaches based on templates (Sect. 5.2) & hidden Markov models (Sect. 5.3). In view of their great importance, Sect. 5.3 provides a detailed technical account on Markov chains & hidden Markov models using chord recognition as a motivating application. In particular, Viterbi algorithm (Sect. 5.3.3.2) & its close relation to DTW algorithm (Sect. 3.2) can be elaborated in a lecture & in homework problems.

Being of high practical relevance & widely known by smartphone users, topic of audio identification (Sect. 7.1) is well suited to delve into topic of content-based audio retrieval. Only requiring spectrogram representation as prerequisite, this sect may be covered directly after Sect. 2.1. Audio identification application provides a good opportunity for raising efficiency & indexing issues – a topic often neglected in music processing & MIR. Next 2 sects on audio matching (Sect. 7.2) & version identification (Sect. 7.3) deal with retrieved may reveal only a low degree of similarity. Requiring chroma-based audio features & alignment techniques, Sect. 7.2 & Sect. 7.3 form a nice continuation of Chaps. 3–4.

Along with Sect. 7.1, Chaps. 6 & 8 focus more on technical aspects. Requiring Fourier analysis of audio signals, this material may be used after covering Sects. 1.3 & 2.1. In Chap. 6, which deals with tempo & beat tracking, Fourier transform is used on 2 different levels. On 1st level, it is used to convert an audio signal into a novelty representation that indicates note onset candidates (Sect. 6.1). On 2nd level, Fourier analysis is applied as a means to detect locally periodic patterns in novelty function. This type of periodicity analysis not only yields a tempogram representation (Sect. 6.2.2), but also reveals locally

periodic pulse trains that can be used for beat tracking applications (Sect. 6.3.1). Having a close personal relation to rhythm & dance, many students are immediately receptive to topic of beat & tempo tracking. Therefore, also in my experience as a lecturer, this topic generates a lot of interest & inspiration.

Chap. 8 is also quite independent from prev chaps & can be studied after Sects. 1.3 & 2.1. Topic of harmonic-percussive separation (Sect. 8.1) is a direct application of spectrogram representation. Applying some simple median filtering & binary masking techniques allows for decomposing a music signal into a percussive component & a harmonic component. In this context, also cover issue of reconstructing time-domain signals from modified spectral representations – a topic fraught with unanticipated pitfalls (Sect. 8.1.2). Using melody extraction as a motivating music processing application, Sect. 8.2 details further important topics including fundamental & instantaneous frequency estimation. This scenario provides opportunity to have a closer look at phase information supplied by Fourier analysis – a rather technical yet important topic that is not easy to understand when studied for 1st time (Sect. 8.2.1).

In Sect. 8.3, touch on another central research field related to source separation. Within this area, a general concept known as nonnegative matrix factorization (NMF) has turned out to be a key technique. Among its many variants, discuss most basic NMF version in Sect. 8.3.1. This technique is then employed for decomposing a music signal into more elementary sound events. Doing so, one can highlight another general strategy that is widely applied in music processing to cope with complexity of music signals. In order to make certain problems tractable, current approaches often exploit musical knowledge in 1 way or another. In this chap, study several score-informed approaches that make use of availability of score representations in order to support an audio processing task. This strategy, in turn, requires note information aligned to audio signal to be processed, which brings us back to Chap. 3 on music synchronization.

– Trong Phần 8.3, đề cập đến 1 lĩnh vực nghiên cứu trung tâm khác liên quan đến việc tách nguồn. Trong lĩnh vực này, 1 khái niệm chung được gọi là phân tích ma trận không âm (NMF) đã trở thành 1 kỹ thuật chính. Trong số nhiều biến thể của nó, hãy thảo luận về phiên bản NMF cơ bản nhất trong Phần 8.3.1. Sau đó, kỹ thuật này được sử dụng để phân tích tín hiệu âm nhạc thành các sự kiện âm thanh cơ bản hơn. Khi làm như vậy, người ta có thể làm nổi bật 1 chiến lược chung khác được áp dụng rộng rãi trong xử lý âm nhạc để đối phó với sự phức tạp của tín hiệu âm nhạc. Để giải quyết 1 số vấn đề nhất định, các phương pháp tiếp cận hiện tại thường khai thác kiến thức âm nhạc theo cách này hay cách khác. Trong chương này, hãy nghiên cứu 1 số phương pháp tiếp cận dựa trên bản nhạc sử dụng tính khả dụng của các biểu diễn bản nhạc để hỗ trợ tác vụ xử lý âm thanh. Đến lượt mình, chiến lược này yêu cầu thông tin nốt nhạc được căn chỉnh với tín hiệu âm thanh để xử lý, điều này đưa chúng ta trở lại Chương 3 về đồng bộ hóa âm nhạc.

- View: Introduction to Fourier Analysis & Applications. Fourier transform is 1 of most important tools for a wide range of applications in engineering & CS. Due to a large number of variants & complex-valued formulation, students often have difficulties in understanding Fourier transform when encountering this concept for 1st time. Music domain offers a natural access to main ideas of Fourier analysis thanks to intuitive relations between abstract concepts & musical counterparts e.g. sinusoids & musical tones, frequency & pitch, magnitude & tone intensity, & so on. This textbook can be used as a basis for an introductory course on Fourier analysis. Starting with some basics on audio representations & their properties (Sect. 1.3), one can continue with Sect. 2.1 to introduce most important facts on Fourier analysis. This sect contains all material actually needed to understand subsequent chaps. For an in-depth treatment of signals, signal spaces, & Fourier analysis – including many of mathematical proofs – one may proceed with remaining sects of Chap. 2. 1 algorithm highlight is definitely fast Fourier transform (FFT), treated in Sect. 2.4.3.

  As example applications of Fourier transform & its short-time versions (STFT, spectrogram), one can then discuss log-frequency spectrograms & their relation to musical pitch (Sect. 3.1.1), spectrum-based novelty detection as used in note onset detection (Sect. 6.1.2), & spectral peak fingerprints applied to audio identification (Sect. 7.1). Using many concrete examples & illustrations provided by book, these applications can be treated in a nontechnical fashion without needing to go through all material of respective chap.

  Considering only magnitude information, phases of complex-valued Fourier coefficients are often neglected in many applications. With Sects. 6.1.3 & 8.2.1, book offers material to illustrate importance of phase & to approach this difficult topic. Using phase-based novelty detection & instantaneous frequency estimation as motivating applications, meaning of phase becomes evident when considering possible phase inconsistencies over subsequent frames. These applications also put STFT & its properties in a different light.

  To round off an introductory course on Fourier analysis, one may look into how to decompose time-frequency representations with applications to source separation. In particular, decomposition of audio signals into harmonic & percussive components by considering horizontal & vertical time-frequency patterns is a simple & very instructive application (Sect. 8.1.1). This scenario also offers a nice motivation for discussing important topics e.g. binary & soft spectral masking (Sect. 8.1.1.2), as well as Fourier inversion & signal reconstruction (Sect. 8.1.2). Finally, as another more advanced application, one may consider Sect. 8.3 on audio decomposition using a technique known as nonnegative matrix factorization (NMF). In this application, a music signal is decomposed into a set of notewise audio events, where each audio event is directly associated with a note of a given musical score.

  – Để kết thúc khóa học nhập môn về phân tích Fourier, người ta có thể xem xét cách phân tích biểu diễn thời gian-tần số bằng các ứng dụng để tách nguồn. Đặc biệt, phân tích tín hiệu âm thanh thành các thành phần hài & gõ bằng cách xem xét các mẫu thời gian-tần số theo chiều ngang & theo chiều dọc là 1 ứng dụng đơn giản & rất bổ ích (Phần 8.1.1). Kịch bản này cũng cung cấp động lực tốt để thảo luận về các chủ đề quan trọng, ví dụ như che lấp phổ mềm nhị phân & (Phần 8.1.1.2), cũng như nghịch đảo Fourier & tái tạo tín hiệu (Phần 8.1.2). Cuối cùng, như 1 ứng dụng nâng cao khác, người ta có thể xem xét Phần 8.3 về phân tích âm thanh bằng 1 kỹ thuật được gọi là phân tích ma trận không âm (NMF). Trong

ứng dụng này, tín hiệu âm nhạc được phân tích thành 1 tập hợp các sự kiện âm thanh theo từng nốt nhạc, trong đó mỗi sự kiện âm thanh được liên kết trực tiếp với 1 nốt nhạc của 1 bản nhạc nhất định.

- View: Data Representations & Algorithms. Finally want to assume view of a computer scientist who may be interested in making his or her basic course on data representations & algorithms a bit more "musical". As a multimedia domain, music offers a wide range of data types & formats including text, symbolic data, audio, image, & video. E.g., as discussed in Chap. 1, music can be represented as printed sheet music (image domain), encoded as MIDI or MusicXML files (symbolic domain), & played back as audio recordings (acoustic domain). Using music as an example, one can discuss fundamental issues of data representations including bitmap & vector graphic encodings for images, XML-like markup languages for symbolic music, communication protocols for electronic musical instruments e.g. MIDI, or audio file formats including WAV or MP3. Immediate relationships between different music representations yield a natural motivation for data conversion issues including image rendering, optical character/music recognition, sound synthesis, & so on (see Fig. 1.24: Illustration of 3 classes of music representation & their relations.).

  1st step in most computer-based analysis & classification applications consists in transforming input data into suitable feature representations, which capture relevant information while suppressing redundancies. Spectrogram representation (Sect. 2.1) & derived audio features (Sect. 3.1) can be seen as typical examples for such a transformation process. In many cases, feature extraction can be seen as a kind of dimensionality reduction. A prominent example are 12-dimensional chroma features, which capture tonal information of a music signal (Sect. 3.1.2).

  After introducing data representations, a CS course may continue with discussion of algorithms. This textbook offers a number of interesting algorithms that are relevant for a wide range of applications going far beyond music processing scenarios considered. Many of these algorithms are based on dynamic programming, which is a fundamental algorithmic paradigm for solving optimization problems. This method appears – in 1 form or another – in curriculum of basically any CS student. Idea of dynamic programming: break down a complex problem into smaller "overlapping" subproblems in some recursive manner. An optimal solution of global problem is obtained by efficiently assembling optimal solutions for subproblems. Dynamic programming is widely used for alignment tasks as occurring in bioinformatics (e.g., to determine similarity of DNA sequences) or in text processing (e.g., to compute distance between text strings). In this book, consider a variant of this technique referred to as dynamic time warping (DTW), which allows us to temporally align feature sequences extracted from music representations. Motivated by a music synchronization application, Sect. 3.2 covers DTW in detail including careful mathematical modeling of optimization problem, algorithm based on dynamic programming, & mathematical proofs. Furthermore, numerous illustrations, examples, & exercises are provided.

  Besides DTW, further algorithms based on dynamic programming are presented throughout book. E.g., subsequence variants of DTW are discussed in context of audio matching (Sect. 7.2) & version identification (Sect. 7.3). In our audio thumbnailing application (Sect. 4.3), dynamic programming is used to efficiently compute a fitness measure for audio segments. Furthermore, well-known Viterbi algorithm for finding an optimizing state sequence is based on dynamic programming – a concept that is applied in this book for estimating chord sequences (Sect. 5.3). Finally, a dynamic programming approach is introduced to derive an optimal beat sequence (Sect. 6.3). In all these problems, which are motivated by concrete applications, objective: find a sequence or an alignment between 2 sequences that is optimal in 1 or another way. By considering various scenarios, student should acquire a solid understanding of underlying principles of dynamic programming.

  There are a number of other important algorithms treated in this book, which may be integrated into a basic CS curriculum. 1st of all, Sect. 2.4.3 covers classic fast Fourier transform (FFT), which goes back to Carl Friedrich Gauss (1805, published posthumously in 1866). Being a typical example for a divide-&-conquer strategy, basic idea of FFT algorithm: divide discrete Fourier transform (DFT) into 2 pieces of half size. FFT algorithm can also be interpreted as a factorization of DFT matrix into a product of sparse matrices.

  In Sect. 8.3, study another matrix factorization technique known as nonnegative matrix factorization (NMF). This technique is studied within an audio decomposition scenario. General objective of NMF: factorize a given real-valued matrix with no negative elements into a product of 2 other matrices that also have no negative elements. Usually, 2 matrices in product have a much lower rank than original matrix. In this case, product can be thought of as a compressed & more structured version of original matrix. As a typical example for how to approach nonconvex optimization problems in ML, discuss an iterative procedure for learning an NMF decomposition (Sect. 8.3.1).

  Originally applied for speech recognition, hidden Markov models (HMMs) are now a standard tool for applications in temporal pattern recognition. Motivated by a chord recognition application, introduce this mathematical concept in Sect. 5.3 as a typical example for a statistical data model. A rigorous treatment of statistical data analysis goes beyond scope of this book. With Sect. 5.3.2, provide, at least, a glimpse into this important area. By considering HMMs, one can also show how alignment concepts e.g. DTW can be extended using a probabilistic framework.

  As a final fundamental topic that may be covered in an introductory course in CS, address issue of data indexing, where objective: speed up a retrieval process. Basic procedure is similar to what we do when using a traditional book index. When looking for a specific passage in a book, an index allows us to directly access page numbers where certain key words occur. In Sect. 7.1, study such techniques in context of an audio identification application. Here, key words correspond to audio fingerprints (e.g., spectral peaks or combinations thereof), while page numbers corresponds to time positions where these fingerprints appear.

  With these comments, hope to have convinced lecturers that music processing may serve as a beautiful & instructive application scenario for teaching basic concepts on data representations & algorithms. In experience as a lecturer in CS & engineering, starting a lecture with music processing applications, in particular playing music to students, opens them up

& raises their interest. This makes it much easier to get students engaged with mathematical theory & technical details. Mixing theory & practice by immediately applying algorithms to concrete music processing tasks helps to develop necessary intuition behind abstract concepts & awakens student's fascination & enthusiasm for topic.

- 1. Music Representations. Music can be represented in many different ways & formats. E.g., a composer may write down a composition in form of a musical score. In a score, musical symbols are used to visually encode notes & how these notes are to be played by a musician. Printed form of a musical score is also referred to as *sheet music*. Original medium of this representation is paper, although now accessible on computer screens through digital images. For electronic instruments & computers, music may be communicated by means of standard protocols e.g. widely used Musical Instrument Digital Interface (MIDI) protocol, where event messages specify pitches, velocities, & other parameters to generate intended sounds. In this book, use term *symbolic* to refer to any machine-readable data format that explicitly represents musical entities. These musical entities may range from timed note events, as is case of MIDI files, to graphical shapes with attached musical meaning, as is case of music engraving systems. Unlike symbolic representations, audio representations e.g. WAV or MP3 files do not explicitly specify musical events. These files encode acoustic waves, which are generated when a source (e.g., an instrument) creates a sound that travels to human ear as air pressure oscillations.

  – 1. Biểu diễn âm nhạc. Âm nhạc có thể được biểu diễn theo nhiều cách & định dạng khác nhau. E.g., 1 nhà soạn nhạc có thể viết ra 1 tác phẩm dưới dạng bản nhạc. Trong bản nhạc, các ký hiệu âm nhạc được sử dụng để mã hóa trực quan các nốt nhạc & cách các nhạc công chơi những nốt nhạc này. Bản in của bản nhạc cũng được gọi là *tờ nhạc*. Phương tiện ban đầu của biểu diễn này là giấy, mặc dù hiện nay có thể truy cập trên màn hình máy tính thông qua hình ảnh kỹ thuật số. Đối với các nhạc cụ điện tử & máy tính, âm nhạc có thể được truyền đạt bằng các giao thức chuẩn, ví dụ như giao thức Giao diện kỹ thuật số nhạc cụ (MIDI) được sử dụng rộng rãi, trong đó các thông báo sự kiện chỉ định cao độ, vận tốc, & các tham số khác để tạo ra âm thanh mong muốn. Trong cuốn sách này, hãy sử dụng thuật ngữ ký hiệu để chỉ bất kỳ định dạng dữ liệu nào có thể đọc được bằng máy, biểu diễn rõ ràng các thực thể âm nhạc. Các thực thể âm nhạc này có thể bao gồm từ các sự kiện nốt nhạc được định thời gian, như trường hợp của các tệp MIDI, đến các hình dạng đồ họa có ý nghĩa âm nhạc kèm theo, như trường hợp của các hệ thống khắc nhạc. Không giống như các biểu diễn ký hiệu, các biểu diễn âm thanh, ví dụ như các tệp WAV hoặc MP3 không chỉ định rõ ràng các sự kiện âm nhạc. Các tệp này mã hóa sóng âm, được tạo ra khi 1 nguồn (ví dụ: nhạc cụ) tạo ra âm thanh truyền đến tai người dưới dạng dao động áp suất không khí.

  In this book, distinguish between 3 main classes of music representations: sheet music, symbolic, & audio. To put it simply, term *sheet music* stands for visual representations of a score given in printed form or in form of digitized images. Term *symbolic* comprises any kind of score representation with an explicit encoding of notes or other musical events. Finally, term *audio* refers to representations of acoustic sound waves. Each of these representations reflects certain aspects of a musical object, but no single representation encompasses al its properties. In this sense, each representation can be considered a projection or a realization of what we generally refer to as a piece of music. In this introductory chap, discuss some basic properties of music by means of these different music representations. Start by describing basic elements of Western music notation as used in sheet music representations (Sect. 1.1). Even though exact specifications of music notation are not essential in this book, require basic notions of pitch, duration, & onset time of musical notes. Then, summarize basic properties of symbolic representations with a specific focus on MIDI, which is prevailing standard for controlling music synthesizers (Sect. 1.2). Finally, discuss audio representations, which at heart of this book. In particular, deal with aspects concerning properties of sound waves including frequency, dynamics, & timbre (Sect. 1.3).

  – Trong cuốn sách này, hãy phân biệt 3 lớp biểu diễn âm nhạc chính: bản nhạc, ký hiệu, & âm thanh. Nói 1 cách đơn giản, thuật ngữ *bản nhạc* dùng để chỉ các biểu diễn trực quan của bản nhạc được đưa ra dưới dạng in hoặc dưới dạng hình ảnh số hóa. Thuật ngữ *biểu tượng* bao gồm bất kỳ loại biểu diễn bản nhạc nào có mã hóa rõ ràng các nốt nhạc hoặc các sự kiện âm nhạc khác. Cuối cùng, thuật ngữ *âm thanh* dùng để chỉ các biểu diễn của sóng âm thanh. Mỗi biểu diễn này phản ánh các khía cạnh nhất định của 1 đối tượng âm nhạc, nhưng không có biểu diễn đơn lẻ nào bao gồm tất cả các thuộc tính của nó. Theo nghĩa này, mỗi biểu diễn có thể được coi là 1 hình chiếu hoặc hiện thực hóa những gì chúng ta thường gọi là 1 tác phẩm âm nhạc. Trong chương giới thiệu này, hãy thảo luận về 1 số thuộc tính cơ bản của âm nhạc thông qua các biểu diễn âm nhạc khác nhau này. Bắt đầu bằng cách mô tả các yếu tố cơ bản của ký hiệu âm nhạc phương Tây được sử dụng trong các biểu diễn bản nhạc (Phần 1.1). Mặc dù các thông số kỹ thuật chính xác của ký hiệu âm nhạc không phải là điều cần thiết trong cuốn sách này, nhưng vẫn yêu cầu các khái niệm cơ bản về cao độ, trường độ, & thời điểm bắt đầu của các nốt nhạc. Sau đó, tóm tắt các đặc tính cơ bản của biểu diễn ký hiệu với trọng tâm cụ thể là MIDI, đây là tiêu chuẩn phổ biến để điều khiển bộ tổng hợp nhạc (Phần 1.2). Cuối cùng, thảo luận về biểu diễn âm thanh, là trọng tâm của cuốn sách này. Đặc biệt, giải quyết các khía cạnh liên quan đến các đặc tính của sóng âm bao gồm tần số, động lực, & âm sắc (Phần 1.3).

  - 1.1. Sheet Music Representations. *Sheet music*, also referred to as *musical score*, provides a visual representation of what commonly refer to – in particular for Western classical music – as "piece of music". Sheet music describes a musical work using a formal language based on musical symbols & letters, which are depicted in a graphical–textual form. Reading sheet music, a musician can create a performance by following given instructions. Performing a piece from sheet music, however, not only requires a special form of literary, i.e., ability to understand music notation, but also involves a creative process. A musical score is rarely played mechanically. Musicians may shape flow of music by varying tempo, dynamics, & articulation, thus resulting in a personal interpretation of given musical score. In this sense, rather than giving rigid specifications, sheet music can be considered as a guide for performing a piece of music leaving room for different interpretations.

    – *Bản nhạc*, còn được gọi là *bản nhạc*, cung cấp 1 hình ảnh đại diện cho những gì thường được gọi – đặc biệt là đối với nhạc cổ điển phương Tây – là "một tác phẩm âm nhạc". Bản nhạc mô tả 1 tác phẩm âm nhạc bằng ngôn ngữ chính thức

dựa trên các ký hiệu âm nhạc & chữ cái, được mô tả dưới dạng đồ họa–văn bản. Khi đọc bản nhạc, 1 nhạc sĩ có thể tạo ra 1 buổi biểu diễn bằng cách làm theo các hướng dẫn được đưa ra. Tuy nhiên, việc biểu diễn 1 tác phẩm từ bản nhạc không chỉ đòi hỏi 1 hình thức văn học đặc biệt, tức là khả năng hiểu ký hiệu âm nhạc, mà còn liên quan đến 1 quá trình sáng tạo. 1 bản nhạc hiếm khi được chơi 1 cách máy móc. Các nhạc sĩ có thể định hình dòng chảy của âm nhạc bằng cách thay đổi nhịp độ, cường độ, & cách phát âm, do đó tạo ra 1 cách diễn giải cá nhân về bản nhạc nhất định. Theo nghĩa này, thay vì đưa ra các thông số kỹ thuật cứng nhắc, bản nhạc có thể được coi là 1 hướng dẫn để biểu diễn 1 tác phẩm âm nhạc, tạo chỗ cho các cách diễn giải khác nhau.

As a 1st example, consider Symphony No. 5 in C minor by Ludwig van Beethoven, which is 1 of most popular & best-known compositions in classical music. It begins with a short musical idea, famous "short-short-short-long" *motif*, which is commonly referred to as "fate motif" of Beethoven's 5th. Fig. 1.1: Sheet music representation of 1st 5 measures of Symphony No. 5 by Ludwig van Beethoven in a piano reduced version. shows a sheet music representation of 1st 5 measures in a piano reduced version. In following sects, explain meaning of musical symbols in more detail while introducing some music notations used throughout this book. Beethoven piece will serve as our running example.

∗ 1.1.1. Musical Notes & Pitches. In music, term *note* is often used in a rather loose way & may refer to both a musical symbol (when talking about score representations) as well as a pitched sound (when talking about audio representations). In this sect, employ term to refer to musical symbols used in Western music notation. Each note has several attributes that determine relative duration & pitch of a sound to be performed by a musician. E.g., in case of a piano, pitch of a note tells a musician which key is to be pressed on keyboard, & duration of note determines how long this key is to be held. Notion of *pitch* is not strict & refers to a perceptual property that allows a listener to order a sound on a frequency-related scale. As discuss in Sect. 1.3 in more detail, playing a note on an instrument results in a (more or less) periodic sound of a certain fundamental frequency. This fundamental frequency is closely related to what is meant by pitch of a note. In following discussion, use term "pitch" in an intuitive way. It allows us to order pitched sounds from "lower" to "higher" – similarly to keys of a piano keyboard ordered from left to right.

– Nốt nhạc & Cao độ. Trong âm nhạc, thuật ngữ *note* thường được sử dụng theo cách khá lỏng lẻo & có thể ám chỉ cả ký hiệu âm nhạc (khi nói về biểu diễn bản nhạc) cũng như âm thanh có cao độ (khi nói về biểu diễn âm thanh). Trong phần này, sử dụng thuật ngữ để ám chỉ các ký hiệu âm nhạc được sử dụng trong ký hiệu âm nhạc phương Tây. Mỗi nốt nhạc có 1 số thuộc tính xác định độ dài tương đối & cao độ của âm thanh do nhạc công biểu diễn. E.g., trong trường hợp của đàn piano, cao độ của 1 nốt nhạc cho nhạc công biết phím nào sẽ được nhấn trên bàn phím, & độ dài của nốt nhạc xác định phím này sẽ được giữ trong bao lâu. Khái niệm *pitch* không nghiêm ngặt & ám chỉ 1 thuộc tính nhận thức cho phép người nghe sắp xếp âm thanh theo thang âm liên quan đến tần số. Như đã thảo luận chi tiết hơn trong Phần 1.3, việc chơi 1 nốt nhạc trên 1 nhạc cụ sẽ tạo ra âm thanh (ít nhiều) tuần hoàn có tần số cơ bản nhất định. Tần số cơ bản này có liên quan chặt chẽ đến ý nghĩa của cao độ của 1 nốt nhạc. Trong phần thảo luận sau, hãy sử dụng thuật ngữ "cao độ" theo cách trực quan. Nó cho phép chúng ta sắp xếp các âm thanh có cao độ từ "thấp hơn" đến "cao hơn" – tương tự như các phím đàn piano được sắp xếp từ trái sang phải.

2 notes with fundamental frequencies in a ratio equal to any power of 2 (e.g., half, twice, or 4 times) are perceived as very similar. Because of that, all notes with this kind of relation can be grouped under same *pitch class*. This observation also leads to fundamental notion of an *octave*, which is defined to be interval between 1 musical note & another with half or double its fundamental frequency. Using this definition, a pitch class is a set of all pitches or notes that are an integer number of octaves apart.

– 2 nốt nhạc có tần số cơ bản theo tỷ lệ bằng bất kỳ lũy thừa nào của 2 (ví dụ, 1 nửa, hai lần hoặc bốn lần) được coi là rất giống nhau. Vì lý do đó, tất cả các nốt nhạc có mối quan hệ này có thể được nhóm lại dưới cùng 1 *pitch class*. Quan sát này cũng dẫn đến khái niệm cơ bản về 1 *octave*, được định nghĩa là khoảng cách giữa 1 nốt nhạc & 1 nốt nhạc khác có tần số cơ bản bằng 1 nửa hoặc gấp đôi. Sử dụng định nghĩa này, 1 pitch class là 1 tập hợp tất cả các cao độ hoặc nốt nhạc cách nhau 1 số nguyên quãng tám.

In order to describe music using a finite number of symbols, one needs to discretize space of all possible pitches. This leads to notion of a *musical scale*, which can be thought of as a finite set of representative pitches. Because of close octave relationship of pitches, scales are generally considered to span a single octave, with higher or lower octaves simply repeating pattern. A musical scale can then be specified by a division of octave space into a certain number of scale steps. Elements of a scale are often simply referred to as *notes* of scale & are ordered according to their respective pitches.

– Để mô tả âm nhạc bằng 1 số lượng hữu hạn các ký hiệu, người ta cần phải rời rạc hóa không gian của tất cả các cao độ có thể. Điều này dẫn đến khái niệm về 1 *âm giai*, có thể được coi là 1 tập hợp hữu hạn các cao độ đại diện. Do mối quan hệ quãng tám chặt chẽ của các cao độ, các thang âm thường được coi là trải dài trên 1 quãng tám duy nhất, với các quãng tám cao hơn hoặc thấp hơn chỉ đơn giản là lặp lại mô hình. Sau đó, 1 thang âm có thể được chỉ định bằng cách chia không gian quãng tám thành 1 số bước thang âm nhất định. Các thành phần của 1 thang âm thường được gọi đơn giản là *nốt* của thang âm & được sắp xếp theo cao độ tương ứng của chúng.

In music history, many different scales have been suggested & used, & there have been fierce discussions about suitability of specific scales. Appropriateness of a scale very much depends on kind of music to be described, instruments used, musical genre, or cultural background. A scale that is suited for representing Western piano music may not be suited for representing Indian sitar music. A scale used for Gregorian chant of 10th century may not be a good choice for describing experimental music of 20th century. There is no universally valid musical scale, & choice of a musical scale necessarily goes along with simplifications typically imposed by practical considerations.

– Trong lịch sử âm nhạc, nhiều thang âm khác nhau đã được đề xuất & sử dụng, & đã có những cuộc thảo luận gay gắt về tính phù hợp của các thang âm cụ thể. Tính phù hợp của 1 thang âm phụ thuộc rất nhiều vào loại nhạc cần mô tả,

nhạc cụ được sử dụng, thể loại âm nhạc hoặc bối cảnh văn hóa. 1 thang âm phù hợp để thể hiện nhạc piano phương Tây có thể không phù hợp để thể hiện nhạc sitar Ấn Độ. 1 thang âm được sử dụng cho thánh ca Gregorian của thế kỷ thứ 10 có thể không phải là lựa chọn tốt để mô tả nhạc thử nghiệm của thế kỷ 20. Không có thang âm nhạc nào có giá trị phổ quát, & việc lựa chọn thang âm nhạc nhất thiết phải đi kèm với những sự đơn giản hóa thường được áp đặt bởi những cân nhắc thực tế.

In this book, only consider case of *12-tone equal-tempered scale*, where an octave is subdivided into 12 scale steps. Fundamental frequencies of these steps are equally spaced on a logarithmic frequency axis (Sect. 1.3.2). Difference between fundamental frequencies of 2 subsequent scale steps is also called a *semitone*, which is smallest possible interval in this scale.

– Trong cuốn sách này, chỉ xem xét trường hợp của *12-tone equal-tempered scale*, trong đó 1 quãng tám được chia thành 12 bước thang âm. Các tần số cơ bản của các bước này được cách đều nhau trên trục tần số logarit (Phần 1.3.2). Sự khác biệt giữa các tần số cơ bản của 2 bước thang âm tiếp theo cũng được gọi là *semitone*, là khoảng cách nhỏ nhất có thể trong thang âm này.

In 12-tone equal-tempered scale, there are 12 pitch classes. In Western music notation, these pitch classes are denoted by combining a letter-name & accidentals. 7 of pitch classes (corresponding to C major) are denoted by letters C, D, E, F, G, A, B. These pitch classes correspond to white keys of a piano keyboard (Fig. 1.2: (a) Section of piano keyboard with keys ranging from C3 to C5. (b) Corresponding notes using Western music notation.). Remaining 5 pitch classes correspond to black keys of a piano keyboard & are denoted by a combination of a letter & an *accidental* (♯, ♭). A sharp ♯ raises a note by a semitone, & a flat ♭ lowers it by a semitone. Accidentals are written after note name. E.g., *D*♯ represents D-sharp & D♭ represents D-flat. In equal-tempered scale, remaining 5 pitches can be either denoted by C♯, D♯, F♯, G♯, A♯ or by D♭, E♭, G♭, A♭, B♭. E.g., C♯ & D♭ represent same pitch class [This phenomenon is also known as *enharmonic equivalence*.], even though from a musical point of view one distinguishes between these 2 concepts.

– Trong thang âm đều 12 cung, có 12 lớp cao độ. Trong ký hiệu âm nhạc phương Tây, các lớp cao độ này được biểu thị bằng cách kết hợp tên chữ cái & dấu hóa. 7 lớp cao độ (tương ứng với Đô trưởng) được biểu thị bằng các chữ cái C, D, E, F, G, A, B. Các lớp cao độ này tương ứng với các phím trắng của bàn phím piano (Hình 1.2: (a) Mặt cắt bàn phím piano với các phím từ C3 đến C5. (b) Các nốt tương ứng sử dụng ký hiệu âm nhạc phương Tây.). 5 lớp cao độ còn lại tương ứng với các phím đen của bàn phím piano & được biểu thị bằng sự kết hợp của 1 chữ cái & 1 dấu hóa (♯, ♭). 1 dấu thăng ♯ nâng 1 nốt lên nửa cung, & 1 dấu giáng ♭ hạ nốt xuống nửa cung. Các dấu hóa được viết sau tên nốt. Ví dụ: D♯ biểu thị Rê thăng & D♭ biểu thị Rê giáng. Trong thang âm đều, 5 cao độ còn lại có thể được biểu thị bằng C♯, D♯, F♯, G♯, A♯ hoặc bằng D♭, E♭, G♭, A♭, B♭. E.g., C♯ & D♭ biểu thị cùng 1 lớp cao độ [Hiện tượng này cũng được gọi là *tương đương enharmonic*.], mặc dù theo quan điểm âm nhạc, người ta phân biệt giữa 2 khái niệm này.

To name notes of 12-tone equal-tempered scale, in addition to indicating pitch class, one needs to provide an identifier for octave. Following *Scientific Pitch Notation*, each note is specified by pitch class name, followed by a number that indicates octave. Note A4 is determined to have a fundamental frequency of 440 Hz & serves as a reference. *Octave number* increases by one upon an ascension from a note with pitch class B to one with pitch class C. E.g., note B4 is followed by note C5. Similarly, octave number decreases by 1 upon a descent from a C to a B. Lowest note C0 in this notation has a fundamental frequency in region of 16 Hz, which is already below what a human can acoustically perceive. Fig. 1.2 shows notes from C3 to C5 along with corresponding keys of a piano keyboard.

– Để đặt tên cho các nốt nhạc của thang âm 12 cung đồng thanh, ngoài việc chỉ ra lớp cao độ, người ta cần cung cấp 1 mã định danh cho quãng tám. Theo sau *Ký hiệu cao độ khoa học*, mỗi nốt nhạc được chỉ định theo tên lớp cao độ, theo sau là 1 con số chỉ quãng tám. Nốt nhạc A4 được xác định có tần số cơ bản là 440 Hz & đóng vai trò là tham chiếu. *Số quãng tám* tăng thêm 1 khi tăng dần từ 1 nốt có lớp cao độ B lên 1 nốt có lớp cao độ C. E.g., nốt nhạc B4 theo sau nốt nhạc C5. Tương tự như vậy, số quãng tám giảm đi 1 khi giảm dần từ nốt C xuống nốt B. Nốt nhạc thấp nhất C0 trong ký hiệu này có tần số cơ bản trong vùng 16 Hz, thấp hơn mức mà con người có thể cảm nhận được về mặt âm học. Hình 1.2 hiển thị các nốt nhạc từ C3 đến C5 cùng với các phím tương ứng của bàn phím đàn piano.

Ordering all notes of equal-tempered scale according to their pitches, one obtains an equal-tempered *chromatic scale*, where all notes of scale are equally spaced. Term *chromatic* is derived from Greek word *chroma*, meaning color. In music context, term "chroma" closely relates to 12 different pitch classes. E.g., notes C2 & C5 both have same chroma value C. I.e., all notes that have same chroma value belong to same pitch class. Recall notes that belong to same pitch class (or have same chroma value) are perceived as similar in a certain way. In contrast, notes that belong to different pitch classes (or have different chroma values) are perceived as dissimilar. This justifies usage of term "chroma" in sense that notes with different chroma values have a different "sound color". Cyclic nature of chroma values is illustrated by *chromatic circle* as shown in Fig. 1.3. (a) Chromatic circle. Extending this notion, *Shepard's helix of pitch* represents linear pitch space as a helix wrapped around a cylinder so that octave-related pitches lie along a single vertical line [23]. Projection of cylinder onto horizontal plane yields chromatic circle. Factorization of a pitch into a chroma value & an octave number will play an important role in this book. Chroma components of pitches can be used to yield mid-level representations, which turn out to be a powerful tool for various music analysis & retrieval applications.

– Sắp xếp tất cả các nốt nhạc của thang âm đồng bậc theo cao độ của chúng, ta sẽ thu được 1 thang âm đồng bậc *chromatic scale*, trong đó tất cả các nốt nhạc của thang âm đều cách đều nhau. Thuật ngữ *chromatic* bắt nguồn từ tiếng Hy Lạp *chroma*, có nghĩa là màu sắc. Trong ngữ cảnh âm nhạc, thuật ngữ "chroma" liên quan chặt chẽ đến 12 lớp cao độ khác nhau. E.g., các nốt C2 & C5 đều có cùng giá trị sắc độ C. Tức là, tất cả các nốt nhạc có cùng giá trị sắc độ đều thuộc về cùng 1 lớp cao độ. Nhớ lại các nốt nhạc thuộc cùng 1 lớp cao độ (hoặc có cùng giá trị sắc độ) được coi là giống nhau theo 1 cách nào đó. Ngược lại, các nốt nhạc thuộc các lớp cao độ khác nhau (hoặc có giá trị sắc độ khác nhau) được

coi là không giống nhau. Điều này biện minh cho việc sử dụng thuật ngữ "chroma" theo nghĩa là các nốt nhạc có giá trị sắc độ khác nhau sẽ có "màu sắc âm thanh" khác nhau. Bản chất tuần hoàn của các giá trị sắc độ được minh họa bằng *vòng tròn sắc độ* như thể hiện trong Hình 1.3. (a) Vòng tròn sắc độ. Mở rộng khái niệm này, *xoắn ốc cao độ của Shepard* biểu diễn không gian cao độ tuyến tính như 1 xoắn ốc quấn quanh 1 hình trụ sao cho các cao độ liên quan đến quãng tám nằm dọc theo 1 đường thẳng đứng duy nhất [23]. Phép chiếu của hình trụ lên mặt phẳng ngang tạo ra vòng tròn sắc độ. Phân tích cao độ thành giá trị sắc độ & 1 số quãng tám sẽ đóng vai trò quan trọng trong cuốn sách này. Các thành phần sắc độ của cao độ có thể được sử dụng để tạo ra các biểu diễn ở mức trung bình, đây hóa ra là 1 công cụ mạnh mẽ cho nhiều ứng dụng phân tích & truy xuất âm nhạc.

∗ 1.1.2. Western Music Notation. Generally speaking, *music notation* refers to a system for graphically representing music through symbols. Standard Western music notation is based on a *staff*, which is a set of 5 horizontal lines & 4 spaces each representing a different musical pitch (Fig. 1.4(a): Staff). Appropriate music symbols, depending upon intended effect, are placed on staff according to their corresponding pitch or function. Pitch is shown by placement of note symbols on staff – sometimes modified by accidentals. Higher placement within a given staff, higher pitch of corresponding note. Furthermore, duration is indicated by shapes of note symbols as well as additional symbols e.g. dots & ties.

– Ký hiệu âm nhạc phương Tây. Nói chung, *music notation* đề cập đến 1 hệ thống biểu diễn đồ họa âm nhạc thông qua các ký hiệu. Ký hiệu âm nhạc phương Tây chuẩn dựa trên 1 *staff*, là 1 tập hợp gồm 5 dòng ngang & 4 khoảng trống, mỗi khoảng trống biểu diễn 1 cao độ âm nhạc khác nhau (Hình 1.4(a): Staff). Các ký hiệu âm nhạc thích hợp, tùy thuộc vào hiệu ứng mong muốn, được đặt trên khuông nhạc theo cao độ hoặc chức năng tương ứng của chúng. Cao độ được thể hiện bằng cách đặt các ký hiệu nốt nhạc trên khuông nhạc – đôi khi được thay đổi bằng các dấu hóa. Vị trí cao hơn trong 1 khuông nhạc nhất định, cao độ của nốt nhạc tương ứng sẽ cao hơn. Ngoài ra, trường độ được chỉ ra bằng hình dạng của các ký hiệu nốt nhạc cũng như các ký hiệu bổ sung, ví dụ như dấu chấm & dấu nối.

Notation is read from left to right. A staff generally begins with a *clef* symbol, which indicates position of 1 particular note on staff. E.g., by convention, *treble clef*, also known as G-clef, indicates 2nd line is pitch G4 (Fig. 1.4(b): Staff with G-clef). Similarly, *bass clef*, also known as F-clef, indicates 4th line is pitch F3 (Fig. 1.4(c): Staff with F-clef). There are also further clef symbols & clef positions. Details are not important in this book. However, one should keep in mind: clef symbol, along with its position, serves as a reference in relation to which meaning of notes positioned on any line or space of staff can be determined. Notes representing a pitch outside scope of 5-line staff can be described using *ledger lines*, which provide a single note with a additional lines & space (see, e.g., C4 in Fig. 1.5: (a) Musical score of a C-major scale starting with C4 & ending with C5. (b) Key signature consisting of 3 flats converting notes into a C-minor scale.)

– Ký hiệu được đọc từ trái sang phải. 1 khuông nhạc thường bắt đầu bằng ký hiệu *clef*, biểu thị vị trí của 1 nốt nhạc cụ thể trên khuông nhạc. E.g., theo quy ước, *clef*, còn được gọi là khóa Sol, biểu thị dòng thứ 2 là cao độ G4 (Hình 1.4(b): Khuông nhạc có khóa Sol). Tương tự, *clef*, còn được gọi là khóa Fa, biểu thị dòng thứ 4 là cao độ F3 (Hình 1.4(c): Khuông nhạc có khóa Fa). Ngoài ra còn có các ký hiệu khóa nhạc khác & vị trí khóa nhạc. Chi tiết không quan trọng trong cuốn sách này. Tuy nhiên, cần lưu ý: ký hiệu khóa nhạc, cùng với vị trí của nó, đóng vai trò là tham chiếu liên quan đến việc có thể xác định ý nghĩa của các nốt nhạc được định vị trên bất kỳ dòng hoặc khoảng trống nào của khuông nhạc. Các nốt nhạc biểu thị cao độ nằm ngoài phạm vi của khuông nhạc 5 dòng có thể được mô tả bằng cách sử dụng *ledger lines*, cung cấp 1 nốt nhạc duy nhất với các dòng bổ sung & khoảng cách (xem, ví dụ, C4 trong Hình 1.5: (a) Bản nhạc của thang âm C trưởng bắt đầu bằng C4 & kết thúc bằng C5. (b) Dấu hóa gồm 3 giáng chuyển các nốt nhạc thành thang âm C thứ.)

Following clef, *key signature* on a staff indicates key of piece by specifying that certain notes are flat or sharp throughout piece, unless otherwise specified. E.g., notes shown in Fig. 1.5a are C4, D4, E4, F4, G4, A4, B4, C5 thus forming a C-major scale. Using key signature consisting of 3 flats as shown in Fig. 1.5b, notes become C4, D4, E♭4, F4, G4, A♭4, B♭4, C5 thus forming a (natural) C-minor scale.

– Tiếp theo khóa nhạc, *dấu hóa* trên khuông nhạc chỉ ra khóa nhạc của bản nhạc bằng cách chỉ rõ rằng 1 số nốt là giáng hoặc thăng trong suốt bản nhạc, trừ khi được chỉ định khác. E.g., các nốt nhạc được hiển thị trong Hình 1.5a là C4, D4, E4, F4, G4, A4, B4, C5 do đó tạo thành thang âm C-trưởng. Sử dụng dấu hóa gồm 3 dấu giáng như được hiển thị trong Hình 1.5b, các nốt nhạc trở thành C4, D4, E♭4, F4, G4, A♭4, B♭4, C5 do đó tạo thành thang âm C-thứ (tự nhiên).

Music is typically organized into temporal units, referred to as *beats*. Repeating sequences of stressed & unstressed beats, in turn, form higher temporal patterns, which are related to what is called *rhythm* of music & is expressed in terms of musical *meter*. A *measure* (or *bar*) is a segment of time defined by a given number of beats. Dividing music into measures not only reflects it rhythmic nature, but also provides regular reference points within it. In music notation, temporal structure of a piece is indicated by *time signature*, which appears in a staff after key signature. Typically, a time signature consists of 2 numerals, one stacked above the other. Lower numerical indicates note duration that represents 1 beat (give as a fraction with regard to a whole note), while upper numeral indicates how many such beats are in a measure. E.g., time signaturel $\frac{6}{8}$ shown in Fig. 1.6. Notation of time signature. (a) 4 quarter notes per measur with upbeat. (b) 6 8th notes per measure. indicates: a measure consists of 6 beats, where a beat has duration of an 8th note. In sheet music, 2 subsequent measures are separated by a vertical line drawn through staff, which are referred to as *bar lines*.

– Âm nhạc thường được sắp xếp thành các đơn vị thời gian, được gọi là *beats*. Lặp lại các chuỗi nhịp nhấn & không nhấn, lần lượt tạo thành các mẫu thời gian cao hơn, liên quan đến cái được gọi là *rhythm* của âm nhạc & được thể hiện dưới dạng *nhịp* âm nhạc. 1 *ô nhịp* (hoặc *bar*) là 1 phân đoạn thời gian được xác định bởi 1 số nhịp nhất định. Chia nhạc thành các ô nhịp không chỉ phản ánh bản chất nhịp điệu của nó mà còn cung cấp các điểm tham chiếu đều đặn trong đó. Trong ký hiệu âm nhạc, cấu trúc thời gian của 1 tác phẩm được biểu thị bằng *nhịp*, xuất hiện trong 1 khuông nhạc sau dấu khóa. Thông thường, 1 dấu nhịp bao gồm 2 chữ số, 1 chữ số xếp chồng lên chữ số kia. Chữ số thấp hơn biểu thị độ dài của nốt nhạc biểu thị 1 nhịp (được đưa ra dưới dạng phân số đối với 1 nốt nhạc trọn vẹn), trong khi chữ số cao hơn biểu

thị có bao nhiêu nhịp như vậy trong 1 ô nhịp. Ví dụ: dấu nhịp l $\frac{6}{8}$ được hiển thị trong Hình 1.6. Ký hiệu dấu nhịp. (a) 4 nốt đen cho mỗi ô nhịp với phách mạnh. (b) 6 nốt móc đơn cho mỗi ô nhịp. chỉ ra: 1 ô nhịp gồm 6 phách, trong đó 1 phách có độ dài bằng nốt móc đơn. Trong bản nhạc, 2 ô nhịp tiếp theo được phân cách bằng 1 đường thẳng đứng kẻ qua khuông nhạc, được gọi là *vạch nhịp*.

After specifying clef, key signature, & time signature, which all reflect global characteristics of piece & hold for entire staff (if not redefined explicitly), actual notes are specified. As illustrated by Fig. 1.7(a) Parts of a note, each note is represented by a symbol that consists of a *note head* & possibly a *stem* & a *flag*. Sometimes several notes are combined by a *beam*. A note's pitch is indicated by its placement on staff & possibly by an accidental, where clef symbol serves as a reference pitch. Duration of a note is defined in a relative fashion by means of its *note value*, which is indicated by color or shape of note head, presence or absence of a stem, & presence or absence of flags (Fig.1.7(b): Notation for different durations of notes.). Whole note is reference value, & other notes are named in accordance. E.g., a *half note* has half length of a whole note, a *quarter note* has a quarter length of a whole note, & so on. For each note value, there also exists a *rest symbol* of equivalent duration, which expresses an interval of silence in a piece of music (Fig. 1.7(c) Notation for different durations of rests.).

– Sau khi chỉ định khóa nhạc, ký hiệu hóa, & nhịp điệu, tất cả đều phản ánh các đặc điểm chung của bản nhạc & giữ nguyên cho toàn bộ khuông nhạc (nếu không được định nghĩa lại 1 cách rõ ràng), các nốt nhạc thực tế được chỉ định. Như minh họa trong Hình 1.7(a) Các phần của 1 nốt nhạc, mỗi nốt nhạc được biểu diễn bằng 1 ký hiệu bao gồm *đầu nốt nhạc* & có thể là *thân nốt nhạc* & 1 *cờ*. Đôi khi, 1 số nốt nhạc được kết hợp bằng 1 *chùm tia*. Cao độ của 1 nốt nhạc được chỉ ra bằng vị trí của nó trên khuông nhạc & có thể bằng 1 dấu hóa, trong đó ký hiệu khóa nhạc đóng vai trò là cao độ tham chiếu. Độ dài của 1 nốt nhạc được xác định theo cách tương đối thông qua *giá trị nốt nhạc* của nó, được chỉ ra bằng màu sắc hoặc hình dạng của đầu nốt nhạc, sự có hay không có thân nốt nhạc, & sự có hay không có cờ (Hình.1.7(b): Ký hiệu cho các độ dài khác nhau của các nốt nhạc.). Toàn bộ nốt nhạc là giá trị tham chiếu, & các nốt nhạc khác được đặt tên theo giá trị đó. E.g., 1 nốt nửa nốt có độ dài bằng 1 nửa nốt tròn, 1 nốt đen nốt có độ dài bằng 1 phần tư nốt tròn, v.v. Đối với mỗi giá trị nốt, cũng tồn tại 1 *ký hiệu nghỉ* có độ dài tương đương, biểu thị khoảng lặng trong 1 bản nhạc (Hình 1.7(c) Ký hiệu cho các độ dài khác nhau của các quãng nghỉ.).

Musical onset times of notes are specified in a relative fashion & follow from horizontal formation of note symbols. Notes that are to be played at same time are given by vertically aligned musical symbols. In this case, different notes may share same stem & flag as illustrated by Fig. 1.1. Once physical duration of a beat is known, physical onset times of notes can be derived from relative timing. Duration of a beat is given by *tempo* indication specified in beats per minute (BPM). E.g., a specification of 120 BPM means that 120 beats are to be played within 1 minute. In case that a beat corresponds to a quarter note, 120 BPM implies duration of a quarter note is half a sec. Composers often suggest a tempo notated above 1st staff line of piece. E.g., in Fig. 1.1, suggested tempo is 108 BPM with a beat being a half note. However, when performing a piece, musicians often significantly deviate from suggested tempo.

– Thời điểm bắt đầu chơi nhạc của các nốt nhạc được chỉ định theo cách tương đối & theo sự hình thành theo chiều ngang của các ký hiệu nốt nhạc. Các nốt nhạc sẽ được chơi cùng lúc được chỉ định bằng các ký hiệu âm nhạc được căn chỉnh theo chiều dọc. Trong trường hợp này, các nốt nhạc khác nhau có thể chia sẻ cùng 1 gốc & cờ như minh họa bởi Hình 1.1. Khi đã biết được độ dài vật lý của 1 nhịp, có thể suy ra thời điểm bắt đầu chơi nhạc vật lý của các nốt nhạc từ thời gian tương đối. Độ dài của 1 nhịp được chỉ định bằng tempo chỉ định theo nhịp mỗi phút (BPM). Ví dụ: thông số kỹ thuật là 120 BPM có nghĩa là 120 nhịp sẽ được chơi trong vòng 1 phút. Trong trường hợp nhịp tương ứng với 1 nốt đen, thì 120 BPM ngụ ý độ dài của 1 nốt đen là nửa giây. Các nhà soạn nhạc thường gợi ý 1 nhịp độ được ký hiệu ở trên dòng khuông nhạc thứ nhất của bản nhạc. Ví dụ: trong Hình 1.1, nhịp độ được gợi ý là 108 BPM với 1 nhịp là nửa nốt nhạc. Tuy nhiên, khi biểu diễn 1 bản nhạc, các nhạc công thường đi chệch đáng kể so với nhịp độ được gợi ý.

To notate music that is played on a piano or is played by different musicians on various instruments, one often uses several staves to notate various musical voices. A single vertical line drawn to left of multiple staves creates a *staff system*, which indicates: music on all staves is to be played simultaneously. A bracket is an additional vertically aligned symbol joining staves. This symbol shows groupings of instruments that function as a unit, e.g. string section of an orchestra (Fig. 1.8(b): Staff system as used for strings). When music notated across different staves is intended to be played at once by a single performer (usually a keyboard instrument or harp), a *grand staff* is created by joining 2 staves by a brace. E.g., in case of piano music, one has 2 staves, where upper staff uses a treble clef & lower staff uses a bass clef (Fig. 1.8(a): Staff system (grand staff) as used for piano.). When playing piano, upper staff is normally played with right hand & lower staff with left hand. This is case with our Beethoven example shown in Fig. 1.1.

– Để ký hiệu bản nhạc được chơi trên đàn piano hoặc được chơi bởi nhiều nhạc công khác nhau trên nhiều nhạc cụ khác nhau, người ta thường sử dụng nhiều khuông nhạc để ký hiệu nhiều giọng nhạc khác nhau. 1 đường thẳng đứng đơn được vẽ ở bên trái của nhiều khuông nhạc tạo thành 1 *hệ thống khuông nhạc*, biểu thị: bản nhạc trên tất cả các khuông nhạc sẽ được chơi cùng lúc. Dấu ngoặc là 1 ký hiệu bổ sung được căn chỉnh theo chiều dọc nối các khuông nhạc. Ký hiệu này cho biết các nhóm nhạc cụ hoạt động như 1 đơn vị, ví dụ như phần dây của 1 dàn nhạc (Hình 1.8(b): Hệ thống khuông nhạc được sử dụng cho đàn dây). Khi bản nhạc được ký hiệu trên nhiều khuông nhạc khác nhau có ý định được chơi cùng 1 lúc bởi 1 nghệ sĩ biểu diễn duy nhất (thường là 1 nhạc cụ bàn phím hoặc đàn hạc), 1 *khuông nhạc lớn* được tạo ra bằng cách nối 2 khuông nhạc bằng 1 dấu ngoặc nhọn. E.g., trong trường hợp bản nhạc piano, người ta có 2 khuông nhạc, trong đó khuông nhạc trên sử dụng khóa Sol & khuông nhạc dưới sử dụng khóa Fa (Hình 1.8(a): Hệ thống khuông nhạc (khuôn nhạc lớn) được sử dụng cho piano.). Khi chơi piano, khuông nhạc trên thường được chơi bằng tay phải & khuông nhạc dưới bằng tay trái. Đây là trường hợp với ví dụ Beethoven của chúng tôi được thể hiện trong Hình 1.1.

Besides aforementioned attributes, music notation may contain many more instructions to musician regarding matters

e.g. tempo, dynamics, & expression. E.g., overall *tempo & style* of piece may be specified by textual notations e.g. *Allegro con brio* (fast with vigor & spirit) or *Andante con moto* (moderate tempo with motion). Other directions e.g. *accelerando* (gradually becoming faster) or *ritardando* (gradually becoming slower) refer to local tempo deviations. Similarly, *dynamics*, which refers to volume of a sound or note, may be described by terms e.g. *forte* (loud), *piano* (soft), *crescendo* (gradually becoming louder), or *diminuendo* (gradually becoming softer). For vocal music, *lyrics* may be written above or below staff lines. Other symbols e.g. *articulation* marks are used to indicate how certain notes are to be played. E.g., a *staccato* mark (a dot placed above or below a note) signifies that a note is to be played with shortened duration detached from subsequent note, whereas a *legato* mark (a curved line placed above or below a group of notes) indicates: musical notes are played smoothly & connected (Fig. 1.9: Musical score with various symbols used for indicating dynamics & articulation.).
– Bên cạnh các thuộc tính đã đề cập ở trên, ký hiệu âm nhạc có thể chứa nhiều hướng dẫn hơn nữa cho nhạc sĩ liên quan đến các vấn đề như nhịp độ, cường độ, & biểu cảm. E.g., nhịp độ chung của bản nhạc có thể được chỉ định bằng ký hiệu văn bản như *Allegro con brio* (nhanh với sức sống & tinh thần) hoặc *Andante con moto* (nhịp độ vừa phải với chuyển động). Các hướng khác như *accelerando* (dần dần trở nên nhanh hơn) hoặc *ritardando* (dần dần trở nên chậm hơn) đề cập đến độ lệch nhịp độ cục bộ. Tương tự như vậy, *cường độ*, đề cập đến âm lượng của âm thanh hoặc nốt nhạc, có thể được mô tả bằng các thuật ngữ như *forte* (to), *piano* (nhẹ), *crescendo* (dần dần trở nên to hơn) hoặc *diminuendo* (dần dần trở nên nhẹ hơn). Đối với nhạc có lời, *lời bài hát* có thể được viết ở trên hoặc dưới các dòng khuông nhạc. Các ký hiệu khác ví dụ như dấu *articulation* được sử dụng để chỉ cách chơi 1 số nốt nhạc nhất định. E.g., dấu *staccato* (một dấu chấm được đặt phía trên hoặc phía dưới 1 nốt nhạc) biểu thị rằng 1 nốt nhạc sẽ được chơi với thời lượng ngắn hơn tách biệt với nốt nhạc tiếp theo, trong khi dấu *legato* (một đường cong được đặt phía trên hoặc phía dưới 1 nhóm nốt nhạc) biểu thị: các nốt nhạc được chơi mượt mà & được kết nối (Hình 1.9: Bản nhạc với nhiều ký hiệu khác nhau được sử dụng để chỉ động lực & cách phát âm.).
Close this sect by coming back to our Beethoven example. In piano reduced version shown in Fig. 1.1, score shows a system with 2 staves, where upper staff for right hand starts with a G-clef & lower staff for left hand with an F-clef. Both staves are equipped with a key signature (3 flats ♭) & a time signature (2 quarter-note beats per measure $\frac{2}{4}$). Score reveals: 1st 5 measures consist of 2 "short-short-short-long" patterns, where 2nd face motif is played lower than 1st fate motif. Further instructions are given in form of additional symbols & textual notations. E.g., a fermata sign indicates: respective note duration should be prolonged. Pedaling information tells musician to hold sustain pedal, which can have a significant impact on sound. Overall tempo is indicated by "Allegro con brio" & a metronome specification (108 half notes per minute) (Exercise 1.1):
– Kết thúc phần này bằng cách quay lại ví dụ về Beethoven của chúng ta. Trong phiên bản piano rút gọn được hiển thị trong Hình 1.1, bản nhạc cho thấy 1 hệ thống có 2 khuông nhạc, trong đó khuông nhạc trên cho tay phải bắt đầu bằng khóa Sol & khuông nhạc dưới cho tay trái với khóa Fa. Cả hai khuông nhạc đều được trang bị 1 dấu hóa (3 giáng ♭) & 1 dấu nhịp (2 phách nốt đen trên mỗi ô nhịp $\frac{2}{4}$). Bản nhạc cho thấy: 5 ô nhịp đầu tiên bao gồm 2 mẫu "ngắn-ngắn-ngắn-dài", trong đó họa tiết mặt thứ 2 được chơi thấp hơn họa tiết số phận thứ nhất. Các hướng dẫn thêm được đưa ra dưới dạng các ký hiệu bổ sung & ký hiệu văn bản. Ví dụ: dấu fermata chỉ ra: thời lượng nốt tương ứng phải được kéo dài. Thông tin về bàn đạp cho nhạc công biết phải giữ bàn đạp duy trì, điều này có thể có tác động đáng kể đến âm thanh. Nhịp độ chung được biểu thị bằng "Allegro con brio" & thông số kỹ thuật của máy đếm nhịp (108 nốt đen mỗi phút) (Bài tập 1.1):

Problem 1. *Assume a pianist exactly follows specifications given in Beethoven example from Fig. 1.1. Determine duration (in millisecs) of a quarter note & a measure, resp.*

Symbol ff stands for "fortissimo" or "very loud" & indicates dynamics.
Now have a look at Fig. 1.10: Sheet music representation of full orchestral score of beginning of Beethoven's 5th Symphony (from Breitkopf & Härtel, Leipzig, 1862)., which shows full orchestral score of beginning of Beethoven's 5th as used by conductors to direct rehearsals & performances. Shown excerpt is scanned version of an edition published by Breitkop & Härtel in 1862. Various staves of system specify music according to different instruments lined up in a fixed order. From top to bottom, voices for woodwinds (flute, oboe, clarinet, bassoon), brass (French horn, trumpet) & percussion (timpani), & strings (violin, viola, cello, double bass) are listed. For certain instruments e.g. violin, there may be > 1 musical voice to be played, each specified by a separate staff (e.g., violin I & violin II).
– Bây giờ hãy xem Hình 1.10: Biểu diễn bản nhạc của bản nhạc giao hưởng đầy đủ của phần đầu Bản giao hưởng số 5 của Beethoven (từ Breitkopf & Härtel, Leipzig, 1862)., trong đó cho thấy bản nhạc giao hưởng đầy đủ của phần đầu Bản giao hưởng số 5 của Beethoven được các nhạc trưởng sử dụng để chỉ đạo các buổi tập & biểu diễn. Đoạn trích được hiển thị là phiên bản được quét của 1 ấn bản do Breitkop & Härtel xuất bản năm 1862. Nhiều khuông nhạc của hệ thống chỉ định âm nhạc theo các nhạc cụ khác nhau được xếp theo thứ tự cố định. Từ trên xuống dưới, các giọng hát cho nhạc cụ hơi (sáo, ô-boa, clarinet, bassoon), kèn đồng (kèn Pháp, kèn trumpet) & bộ gõ (timpani), & bộ dây (violin, viola, cello, double bass) được liệt kê. Đối với 1 số nhạc cụ nhất định, ví dụ: vĩ cầm, có thể có > 1 giọng nhạc cần chơi, mỗi giọng được chỉ định bởi 1 khuông nhạc riêng (ví dụ, vĩ cầm I & vĩ cầm II).
In this sect, have only scratched surface of Western music notation. Rather than giving a comprehensive overview, goal was to build up some intuition while introducing some basic terminology. Furthermore, wanted to indicate: music notation is far from being comprehensive. Many of symbols only give a vague description of how notes should be played leaving room for artistic freedom & creativity. Furthermore, as indicated by full score & piano transcript of Beethoven's 5th, there may exist different score versions of same piece of music. For most parts of this book, it suffices to have a rough understanding of musical concepts examined in this chap. Aspects of pitch & timing will be picked up again when discussing various kinds of derived music representations.

– Trong giáo phái này, chỉ có bề mặt sơ lược về ký hiệu âm nhạc phương Tây. Thay vì đưa ra cái nhìn tổng quan toàn diện, mục tiêu là xây dựng 1 số trực giác trong khi giới thiệu 1 số thuật ngữ cơ bản. Hơn nữa, muốn chỉ ra rằng: ký hiệu âm nhạc còn lâu mới toàn diện. Nhiều ký hiệu chỉ đưa ra mô tả mơ hồ về cách chơi các nốt nhạc, tạo chỗ cho sự tự do nghệ thuật & sáng tạo. Hơn nữa, như được chỉ ra bởi bản nhạc đầy đủ & bản chép lại piano của Beethoven số 5, có thể tồn tại các phiên bản bản nhạc khác nhau của cùng 1 tác phẩm. Đối với hầu hết các phần của cuốn sách này, chỉ cần hiểu sơ qua về các khái niệm âm nhạc được xem xét trong chương này là đủ. Các khía cạnh về cao độ & thời gian sẽ được đề cập lại khi thảo luận về các loại biểu diễn âm nhạc phái sinh khác nhau.

- 1.2. Symbolic Representations. Symbolic representations describe music by means of entities that have an explicit musical meaning &, given in some digital format, can be parsed by a computer. Any kind of digital data format may be regarded as "symbolic" since it is based on a finite alphabet of letters or symbols. e.g., pixels in a digital image file or samples in a digital audio file may be regarded as symbols or basic entities. However, considering these entities individually, no musical meaning can be inferred. Therefore, neither scanned images nor digitized music recordings are regarded as being symbolic music formats. Similarly, graphical shapes in vector graphics representations are not considered to be musical entities as long as no musically meaningful specification of shapes is given. Still, There is a wide range of what may be considered as symbolic music. In this sect, discuss some examples including piano-roll representations, MIDI representations, & other symbolic formats that encode sheet music. Furthermore, touch on optical music recognition (OMR), which is process of converting digital scans of printed sheet music into symbolic representations.

  – Biểu diễn tượng trưng. Biểu diễn tượng trưng mô tả âm nhạc bằng các thực thể có ý nghĩa âm nhạc rõ ràng &, được đưa ra ở 1 số định dạng kỹ thuật số, có thể được phân tích bằng máy tính. Bất kỳ loại định dạng dữ liệu kỹ thuật số nào cũng có thể được coi là "tượng trưng" vì nó dựa trên bảng chữ cái hữu hạn của các chữ cái hoặc ký hiệu. ví dụ, các điểm ảnh trong tệp hình ảnh kỹ thuật số hoặc các mẫu trong tệp âm thanh kỹ thuật số có thể được coi là ký hiệu hoặc thực thể cơ bản. Tuy nhiên, khi xem xét các thực thể này riêng lẻ, không thể suy ra ý nghĩa âm nhạc nào. Do đó, cả hình ảnh được quét & bản ghi âm nhạc được số hóa đều không được coi là định dạng âm nhạc tượng trưng. Tương tự như vậy, các hình dạng đồ họa trong biểu diễn đồ họa vector không được coi là thực thể âm nhạc miễn là không có thông số kỹ thuật có ý nghĩa về mặt âm nhạc nào được đưa ra. Tuy nhiên, vẫn có nhiều thứ có thể được coi là âm nhạc tượng trưng. Trong phần này, hãy thảo luận 1 số ví dụ bao gồm biểu diễn piano-roll, biểu diễn MIDI, & các định dạng tượng trưng khác mã hóa bản nhạc. Ngoài ra, hãy đề cập đến nhận dạng âm nhạc quang học (OMR), đây là quá trình chuyển đổi các bản quét kỹ thuật số của bản nhạc đã in thành các biểu diễn tượng trưng.

  * 1.2.1. Piano-Roll Representations. Start with a symbolic representation having a history of > 100 years. In late 19th & early 20th century, self-playing pianos, so-called *player pianos* (Fig. 1.11(a): Player piano.), became quite popular with a peak in 1924, before being replaced by phonograph recordings. Player pianos contained pneumatic mechanisms to automatically operate key & pedal movements according to instructions specified by a prestored piano-roll medium. A *piano roll* is a continuous roll of paper with perforations (holes) punched into it. Perforations represent note control data (Fig. 1.11(b): Piano roll.). Roll moves over a reading system known as a *tracker bar*, & playing cycle for each musical note is triggered when a perforation crosses bar & is read. Rolls for player pianos were generally made from recorded performances of musicians. This way, playing of many famous pianists & composers including *Gustav Mahler, Edvard Grieg, Scott Joplin, George Gershwin* is preserved on piano rolls. Typically, a pianist would sit at a specially designed player piano, & pitch & duration of sustain & soft pedal. Player pianos can also recreate dynamics of a pianist's performance by means of specially encoded control perforations placed towards edges of a music roll.

    – Biểu diễn Piano-Roll. Bắt đầu bằng 1 biểu diễn tượng trưng có lịch sử > 100 năm. Vào cuối thế kỷ 19 & đầu thế kỷ 20, đàn piano tự chơi, còn gọi là *đàn piano của người chơi* (Hình 1.11(a): Đàn piano của người chơi.), trở nên khá phổ biến với đỉnh cao vào năm 1924, trước khi bị thay thế bằng các bản ghi âm máy hát. Đàn piano của người chơi có cơ chế khí nén để tự động vận hành các chuyển động của phím & bàn đạp theo hướng dẫn được chỉ định bởi 1 phương tiện piano-roll được lưu trữ trước. *đàn piano roll* là 1 cuộn giấy liên tục có các lỗ đục trên đó. Các lỗ đục biểu diễn dữ liệu điều khiển nốt nhạc (Hình 1.11(b): Đàn piano roll.). Cuộn di chuyển trên 1 hệ thống đọc được gọi là *thanh theo dõi*, & chu kỳ chơi cho mỗi nốt nhạc được kích hoạt khi 1 lỗ đục ngang qua thanh & được đọc. Các cuộn cho đàn piano của người chơi thường được tạo ra từ các buổi biểu diễn được ghi âm của các nhạc sĩ. Theo cách này, bản nhạc của nhiều nghệ sĩ piano nổi tiếng & nhà soạn nhạc bao gồm *Gustav Mahler, Edvard Grieg, Scott Joplin, George Gershwin* được lưu giữ trên các cuộn piano. Thông thường, 1 nghệ sĩ piano sẽ ngồi vào 1 chiếc đàn piano tự chơi được thiết kế đặc biệt, & cao độ & thời lượng duy trì & bàn đạp mềm. Đàn piano tự chơi cũng có thể tái tạo động lực biểu diễn của nghệ sĩ piano bằng cách sử dụng các lỗ điều khiển được mã hóa đặc biệt đặt ở các cạnh của cuộn nhạc.

    In following, a *piano-roll representation* is understood to be a geometric visualization of note information as specified by a piano roll. Horizontal axis of this 2D representation encodes time, whereas vertical axis encodes pitch. Every note is described by an axis-parallel rectangle coding 3 parameters. 1st parameter is onset time, given by leftmost horizontal coordinate of rectangle, & 2nd is pitch, given by lower vertical coordinate of rectangle. Finally, 3rd parameter is duration of note, encoded by width of rectangle.

    – Sau đây, 1 *piano-roll representation* được hiểu là 1 hình ảnh trực quan hình học của thông tin nốt nhạc được chỉ định bởi 1 piano roll. Trục ngang của biểu diễn 2D này mã hóa thời gian, trong khi trục dọc mã hóa cao độ. Mỗi nốt nhạc được mô tả bằng 1 hình chữ nhật song song trục mã hóa 3 tham số. Tham số thứ nhất là thời điểm bắt đầu, được chỉ định bởi tọa độ ngang ngoài cùng bên trái của hình chữ nhật, & thứ hai là cao độ, được chỉ định bởi tọa độ dọc dưới cùng của hình chữ nhật. Cuối cùng, tham số thứ ba là thời lượng của nốt nhạc, được mã hóa theo chiều rộng của hình chữ nhật.

    Fig. 1.12(b): piano-roll representation of beginning of Fugue BWV 846 in C major by Johann Sebastian Bach. 4 occurrences of theme are highlighted. shows a piano-roll representation of beginning of Fugue BWV 846 in C major by Johann

Sebastian Bach. For comparison, Fig. 1.12(a): Sheet music representation. shows corresponding part in a sheet music representation. Generally, a *fugue* is a compositional technique using ≥ 2 musical voices, built on a musical theme (or subject) that is introduced at beginning by 1 voice & then repeated at different pitches in other voices. Fugue BWV 846 consists of 4 voices. Although played on a keyboard instrument, 4 voices are referred to as soprano (highest voice), alto (2nd highest voice), tenor (3rd highest voice), & bass (lowest voice). Fugue starts with main theme in alto, which is then repeated in soprano, tenor, & finally in bass. As shown in Fig. 1.12, 4 occurrences of theme are hard to detect in sheet music representation, but can be easily seen in piano-roll representation, where each one corresponds to a pattern shifted in time-pitch plane.

– Hình 1.12(b): biểu diễn piano-roll phần đầu của Fugue BWV 846 cung Đô trưởng của Johann Sebastian Bach. 4 lần xuất hiện của chủ đề được tô sáng. cho thấy biểu diễn piano-roll phần đầu của Fugue BWV 846 cung Đô trưởng của Johann Sebastian Bach. Để so sánh, Hình 1.12(a): Biểu diễn bản nhạc. cho thấy phần tương ứng trong biểu diễn bản nhạc. Nói chung, *fugue* là 1 kỹ thuật sáng tác sử dụng ≥ 2 giọng nhạc, được xây dựng trên 1 chủ đề âm nhạc (hoặc chủ đề) được giới thiệu ở phần đầu bởi 1 giọng & sau đó được lặp lại ở các cao độ khác nhau ở các giọng khác. Fugue BWV 846 bao gồm 4 giọng. Mặc dù được chơi trên 1 nhạc cụ bàn phím, 4 giọng được gọi là soprano (giọng cao nhất), alto (giọng cao thứ 2), tenor (giọng cao thứ 3), & bass (giọng thấp nhất). Fugue bắt đầu với chủ đề chính ở giọng alto, sau đó được lặp lại ở giọng soprano, tenor, & cuối cùng ở giọng bass. Như thể hiện trong Hình 1.12, 4 lần xuất hiện chủ đề khó phát hiện trong biểu diễn bản nhạc, nhưng có thể dễ dàng thấy trong biểu diễn piano-roll, trong đó mỗi lần tương ứng với 1 mẫu được dịch chuyển trong mặt phẳng cao độ thời gian.

While they are a considerable simplification of what is notated in sheet music, piano-roll representations visually describe most important attributes of musical notes in an easy-to-understand way. Therefore, will often use piano-roll representations when describing & talking about symbolic music. Furthermore, one can also derive similar representations from other music encodings including MIDI & audio. In this sense, piano rolls can be seen as a kind of *mid-level representation* on basis of which semantic relations can be established across various manifestations of music.

– Mặc dù chúng là 1 sự đơn giản hóa đáng kể những gì được ghi trong bản nhạc, các biểu diễn piano-roll mô tả trực quan các thuộc tính quan trọng nhất của các nốt nhạc theo cách dễ hiểu. Do đó, thường sẽ sử dụng các biểu diễn piano-roll khi mô tả & nói về âm nhạc tượng trưng. Hơn nữa, người ta cũng có thể suy ra các biểu diễn tương tự từ các mã hóa âm nhạc khác bao gồm MIDI & âm thanh. Theo nghĩa này, piano roll có thể được coi là 1 loại *biểu diễn ở mức trung bình* trên cơ sở đó các mối quan hệ ngữ nghĩa có thể được thiết lập trên nhiều biểu hiện khác nhau của âm nhạc.

∗ 1.2.2. MIDI Representations. Next symbolic representation want to discuss is based on MIDI standard, which stands for *Musical Instrument Digital Interface*. Although MIDI was not originally developed to be used as a symbolic music format & imposes many limitations on what can actually be represented, importance of MIDI is due to its widespread usage over last 3 decades ,& abundance of MIDI data freely available on web. From a music encoding point of view, one needs to keep in mind: quality of available MIDI data is sometimes questionable.

– Biểu diễn MIDI. Biểu diễn ký hiệu tiếp theo muốn thảo luận dựa trên chuẩn MIDI, viết tắt của *Musical Instrument Digital Interface*. Mặc dù MIDI ban đầu không được phát triển để sử dụng như 1 định dạng nhạc ký hiệu & áp đặt nhiều hạn chế đối với những gì thực sự có thể được biểu diễn, tầm quan trọng của MIDI là do việc sử dụng rộng rãi trong 3 thập kỷ qua,& dữ liệu MIDI dồi dào có sẵn miễn phí trên web. Theo quan điểm mã hóa âm nhạc, người ta cần lưu ý: chất lượng dữ liệu MIDI khả dụng đôi khi không chắc chắn.

MIDI was originally developed as an industry standard to get digital electronic musical instruments from different manufacturers to work & play together. It was advent of MIDI in 1981–1983 that caused a rapid growth of electronic musical instrument market. MIDI allows a musician to remotely & automatically control an electronic instrument or a digital synthesizer in real time. E.g., consider a digital piano, where a musician pushes a key of piano keyboard to start a sound. Intensity of sound is controlled by velocity of keystroke. Releasing key stops sound. Instead of physically pushing & releasing piano key, musician may also trigger instrument to produce same sound by transmitting suitable MIDI messages, which encode note-on, velocity, note-off, & other information. These MIDI messages may be automatically generated by some other electronic instrument or may be provided by a computer. An important fact: MIDI does not represent musical directly, but only represent performance information encoding instructions about how an instrument has been played or how music is to be produced.

– MIDI ban đầu được phát triển như 1 tiêu chuẩn công nghiệp để đưa các nhạc cụ điện tử kỹ thuật số từ các nhà sản xuất khác nhau hoạt động & chơi cùng nhau. Sự ra đời của MIDI vào năm 1981–1983 đã khiến thị trường nhạc cụ điện tử tăng trưởng nhanh chóng. MIDI cho phép nhạc công điều khiển từ xa & tự động 1 nhạc cụ điện tử hoặc 1 bộ tổng hợp kỹ thuật số theo thời gian thực. E.g., hãy xem xét 1 cây đàn piano kỹ thuật số, nơi nhạc công nhấn 1 phím của bàn phím piano để bắt đầu phát ra âm thanh. Cường độ âm thanh được điều khiển bởi tốc độ nhấn phím. Nhả phím sẽ dừng âm thanh. Thay vì nhấn & nhả phím piano, nhạc công cũng có thể kích hoạt nhạc cụ để tạo ra âm thanh tương tự bằng cách truyền các thông điệp MIDI phù hợp, mã hóa thông tin về nốt bật, tốc độ, nốt tắt & khác. Các thông điệp MIDI này có thể được tạo tự động bởi 1 số nhạc cụ điện tử khác hoặc có thể được cung cấp bởi máy tính. 1 sự thật quan trọng: MIDI không biểu diễn trực tiếp âm nhạc mà chỉ biểu diễn thông tin biểu diễn mã hóa các hướng dẫn về cách chơi nhạc cụ hoặc cách tạo ra âm nhạc.

Original MIDI standard was later augmented to include *Standard MIDI File* (SMF) specification, which describes how MIDI data should be stored on a computer. In following, denote SMF files simply as *MIDI files* or *MIDI representations*. SMF file format allows users to exchange MIDI data regardless of computer OS & has provided a basis for an efficient internet-wide distribution of music data, including numerous websites devoted to sale & exchange of music. A MIDI file contains a list of MIDI messages together with timestamps, which are required to determine timing of messages. Further

information (called *meta messages*) is relevant to software that processes MIDI files.
– Tiêu chuẩn MIDI gốc sau đó được bổ sung để bao gồm thông số kỹ thuật *Standard MIDI File* (SMF), mô tả cách dữ liệu MIDI nên được lưu trữ trên máy tính. Sau đây, hãy chỉ định các tệp SMF đơn giản là *MIDI files* hoặc *MIDI representations*. Định dạng tệp SMF cho phép người dùng trao đổi dữ liệu MIDI bất kể hệ điều hành máy tính & đã cung cấp cơ sở cho việc phân phối dữ liệu âm nhạc hiệu quả trên toàn internet, bao gồm nhiều trang web dành riêng cho việc bán & trao đổi nhạc. Tệp MIDI chứa danh sách các thông báo MIDI cùng với dấu thời gian, cần thiết để xác định thời gian của các thông báo. Thông tin bổ sung (được gọi là *meta messages*) có liên quan đến phần mềm xử lý các tệp MIDI.
For our purposes, most important MIDI messages are note-on & note-off commands, which correspond to start & end of a note, resp. Each note-on & note-off message is, among others, equipped with a MIDI note number, a value for key velocity, a channel specification, as well as a timestamp. *MIDI note number* $\in [0, 127] \cap \mathbb{N}$ & encodes a note's pitch. Here, MIDI pitches are based on equal-tempered scale as discussed in Sect. 1.1.1. Similarly to an acoustic piano, where 88 keys of keyboard corresponds to musical pitches A0 to C8, MIDI note numbers encode, in increasing order, musical pitches C0 to G♯9. E.g., note C4 has MIDI note number 60, whereas concert pitch A4 has MIDI note number 69.
– Đối với mục đích của chúng tôi, hầu hết các thông điệp MIDI quan trọng là các lệnh bật nốt & tắt nốt, tương ứng với bắt đầu & kết thúc của 1 nốt, tương ứng. Mỗi thông điệp bật nốt & tắt nốt, trong số những thông điệp khác, được trang bị 1 số nốt MIDI, 1 giá trị cho tốc độ phím, 1 thông số kênh cũng như 1 dấu thời gian. *MIDI note number* $\in [0, 127] \cap \mathbb{N}$ & mã hóa cao độ của 1 nốt. Ở đây, cao độ MIDI dựa trên thang âm đều như đã thảo luận trong Phần 1.1.1. Tương tự như đàn piano cơ, trong đó 88 phím đàn tương ứng với cao độ nhạc từ A0 đến C8, thì số nốt MIDI mã hóa, theo thứ tự tăng dần, cao độ nhạc từ C0 đến G♯9. E.g., nốt C4 có số nốt MIDI là 60, trong khi cao độ hòa nhạc A4 có số nốt MIDI là 69.
Key velocity is again $\in [0, 127] \cap \mathbb{N}$, which controls intensity of sound – in case of a note-on event it determines volume, whereas in case of a note-off event it controls decay during release phase of tone. Exact interpretation of key velocity, however, depends on respective instrument or synthesizer. *MIDI channel* is $\in [0, 15] \cap \mathbb{N}$. Intuitively speaking, this number prompts synthesizer to use instrument that has been previously assigned to respective channel number. Note: each channel, in turn, supports polyphony, i.e., multiple simultaneous notes. Finally, *time stamp* is an integer value that represents how many clock pulses or *ticks* to wait before respective note-on or note-off command is executed. Before comment in more detail on timing concept employed by MIDI, illustrate MIDI representation by means of our Beethoven example. Fig. 1.13(b): MIDI representation (in a simplified, tabular form). shows a (simplified & tabular) MIDI encoding of 1st fate motif corresponding to 12 notes of score in Fig. 1.13: Various symbolic music representations of 1st 12 notes of Beethoven's 5th. (a) Sheet music representation. In this example, notes of right hand are assigned to channel 1 & notes of left hand to channel 2. Notes specified by corresponding note-on & note-off events in MIDI file can also be visualized by a piano-roll representation (Fig. 1.13(c): Piano-roll representation.). In case only interested in note events (& note channel & velocity information), this is how represent MIDI information.
– *Key velocity* 1 lần nữa là $\in [0, 127] \cap \mathbb{N}$, điều khiển cường độ âm thanh – trong trường hợp sự kiện bật nốt, nó xác định âm lượng, trong khi trong trường hợp sự kiện tắt nốt, nó điều khiển sự suy giảm trong pha nhả âm. Tuy nhiên, cách diễn giải chính xác về key velocity phụ thuộc vào từng nhạc cụ hoặc bộ tổng hợp âm thanh. *MIDI channel* là $\in [0, 15] \cap \mathbb{N}$. Theo trực giác, con số này nhắc bộ tổng hợp âm thanh sử dụng nhạc cụ đã được gán trước đó cho số kênh tương ứng. Lưu ý: mỗi kênh, lần lượt, hỗ trợ đa âm, tức là nhiều nốt nhạc đồng thời. Cuối cùng, *time stamp* là 1 giá trị số nguyên biểu thị số xung nhịp hoặc *tick* phải đợi trước khi lệnh bật nốt hoặc tắt nốt tương ứng được thực thi. Trước khi bình luận chi tiết hơn về khái niệm thời gian được MIDI sử dụng, hãy minh họa biểu diễn MIDI bằng ví dụ Beethoven của chúng tôi. Hình 1.13(b): Biểu diễn MIDI (dưới dạng bảng đơn giản hóa). hiển thị mã hóa MIDI (dạng bảng đơn giản hóa) của họa tiết số phận đầu tiên tương ứng với 12 nốt nhạc trong Hình 1.13: Nhiều biểu diễn âm nhạc tượng trưng của 12 nốt đầu tiên trong bản giao hưởng số 5 của Beethoven. (a) Biểu diễn bản nhạc. Trong ví dụ này, các nốt của tay phải được gán cho kênh 1 & các nốt của tay trái được gán cho kênh 2. Các nốt được chỉ định bởi các sự kiện bật nốt & tắt nốt tương ứng trong tệp MIDI cũng có thể được trực quan hóa bằng biểu diễn piano-roll (Hình 1.13(c): Biểu diễn piano-roll.). Trong trường hợp chỉ quan tâm đến các sự kiện nốt (& kênh nốt & thông tin về vận tốc), đây là cách biểu diễn thông tin MIDI.
An important feature of MIDI format: it can handle musical as well as physical onset times & note durations. Similarly to sheet music representations, MIDI can express timing information in terms of musical entities rather than using absolute time units e.g. microseconds. To this end, MIDI subdivides a quarter note into basic time units referred to as *clock pulses* or *ticks*. Number of pulses per quarter note (PPQN) is to be specified at beginning, in so-called *header* of a MIDI file, & refers to all subsequent MIDI messages. A common value is 120 PPQN, which determines resolution of time stamps associated to note events. A time stamp indicates how many ticks to wait before a certain MIDI message is executed, relative to previous MIDI message. E.g., 1st note-on message with MIDI note number 67 is executed after 60 ticks, corresponding to 8th rest at beginning of Beethoven's 5th. 2nd & 3rd note-on messages are executed at same time as 1st one, encoded by tick value 0. Then, after 55 ticks, MIDI note 67 is switched off by note-off message & so on.
– 1 tính năng quan trọng của định dạng MIDI: nó có thể xử lý thời gian bắt đầu theo nhạc cũng như thời gian bắt đầu theo vật lý & thời lượng nốt nhạc. Tương tự như các biểu diễn bản nhạc, MIDI có thể thể hiện thông tin về thời gian theo các thực thể âm nhạc thay vì sử dụng các đơn vị thời gian tuyệt đối, ví dụ như micro giây. Để đạt được mục đích này, MIDI chia nhỏ 1 nốt đen thành các đơn vị thời gian cơ bản được gọi là *xung nhịp* hoặc *tích tắc*. Số xung trên mỗi nốt đen (PPQN) phải được chỉ định ở phần đầu, trong cái gọi là *tiêu đề* của tệp MIDI, & tham chiếu đến tất cả các thông báo MIDI tiếp theo. 1 giá trị phổ biến là 120 PPQN, xác định độ phân giải của dấu thời gian liên quan đến các sự kiện nốt nhạc. Dấu thời gian cho biết cần đợi bao nhiêu tích tắc trước khi 1 thông báo MIDI nhất định được thực thi, so với thông báo MIDI trước đó. Ví dụ: thông báo nốt đầu tiên với số nốt MIDI là 67 được thực thi sau 60 tích tắc, tương ứng

với dấu nghỉ thứ 8 ở đầu bản giao hưởng số 5 của Beethoven. Tin nhắn bật nốt thứ 2 & thứ 3 được thực hiện cùng lúc với tin nhắn thứ 1, được mã hóa bằng giá trị tích 0. Sau đó, sau 55 tích, nốt MIDI 67 sẽ bị tắt bằng tin nhắn tắt nốt & cứ như vậy.

Like sheet music representation, MIDI also allows for encoding & storing absolute timing information, however, at a much finer resolution level & in a more flexible way. To this end, one can include additional tempo messages that specify number of microseconds per quarter note. From tempo message, one can compute absolute duration of a tick. E.g., having 600000 μs per quarter note & 120 PPQN, each tick corresponds to 5000 μs. Furthermore, one can derive from tempo message number of quarter notes played in a minute, which yields tempo measured in *beats per minute* (BPM). E.g., 600000 μs per quarter note correspond to 100 BPM. While number of pulses per quarter note is fixed throughout a MIDI file, absolute tempo information may be changed by inserting a tempo message between any 2 note-on or other MIDI messages. This makes it possible to account not only for global tempo information but also for local tempo changes e.g. accelerandi, ritardandi, or fermate.

– Giống như biểu diễn bản nhạc, MIDI cũng cho phép mã hóa & lưu trữ thông tin thời gian tuyệt đối, tuy nhiên, ở mức độ phân giải tốt hơn nhiều & theo cách linh hoạt hơn. Để đạt được mục đích này, người ta có thể bao gồm các thông điệp nhịp độ bổ sung chỉ định số micro giây cho mỗi nốt đen. Từ thông điệp nhịp độ, người ta có thể tính toán thời lượng tuyệt đối của 1 tích tắc. Ví dụ: có 600000 μs cho mỗi nốt đen & 120 PPQN, mỗi tích tắc tương ứng với 5000 μs. Hơn nữa, người ta có thể suy ra từ thông điệp nhịp độ số nốt đen được chơi trong 1 phút, tạo ra nhịp độ được đo bằng *nhịp mỗi phút* (BPM). Ví dụ: 600000 μs cho mỗi nốt đen tương ứng với 100 BPM. Trong khi số xung cho mỗi nốt đen được cố định trong toàn bộ tệp MIDI, thông tin nhịp độ tuyệt đối có thể thay đổi bằng cách chèn 1 thông điệp nhịp độ giữa bất kỳ 2 nốt trên hoặc các thông điệp MIDI khác. Điều này giúp có thể tính đến không chỉ thông tin về nhịp độ toàn cầu mà còn cả những thay đổi về nhịp độ cục bộ, ví dụ như accelerandi, ritardandi hoặc fermate.

In this sect, have briefly touched on MIDI & its functionality. MIDI was originally designed to solve problems in electronic music performance & is limited in terms of musical aspects it represents. E.g., MIDI is not capable of distinguishing between a D♯4 & an E♭4, both of which have MIDI note number 63. Also, information on representation of beams, stem directions, or clefs is not encoded by MIDI. Furthermore, MIDI does not define a note element explicitly; rather, notes are bounded by note-on & note-off events (or note-on events with velocity 0). Rests are not represented at all & must be inferred from absence of notes.

– Trong phần này, đã đề cập sơ qua về MIDI & chức năng của nó. MIDI ban đầu được thiết kế để giải quyết các vấn đề trong biểu diễn nhạc điện tử & bị hạn chế về các khía cạnh âm nhạc mà nó biểu diễn. E.g., MIDI không có khả năng phân biệt giữa D♯4 & E♭4, cả hai đều có số nốt MIDI là 63. Ngoài ra, thông tin về cách biểu diễn các thanh, hướng thân hoặc khóa nhạc không được MIDI mã hóa. Hơn nữa, MIDI không định nghĩa rõ ràng 1 thành phần nốt nhạc; thay vào đó, các nốt nhạc bị giới hạn bởi các sự kiện bật & tắt nốt nhạc (hoặc các sự kiện bật nốt nhạc với vận tốc 0). Các dấu lặng không được biểu diễn chút nào & phải được suy ra từ sự vắng mặt của các nốt nhạc.

∗ 1.2.3. Score Representations. Within class of symbolic music representations, want to distinguish 1 subclass referred to as *score representations*. A representation from this subclass is defined to yield explicit information about musical symbols e.g. staff system, clefs, time signatures, notes, rests, accidentals, & dynamics. In this sense, score representations are, compared with MIDI representations, much closer to what is actually shown in sheet music. E.g., in a score representation, notes D♯4 & E♭b would be distinguishable, & musical onset times are specified. However, a score representation may not contain a description of final layout & particular shape of musical symbols. Process of generating or rendering visually pleasing sheet music representations from score representations is an art in itself. In former days, art of drawing high-quality music notation for mechanical reproduction was called *music engraving*. Nowadays, computer software or *scorewriters* have been designed for purpose of writing, editing, & printing music, though only a few produce results comparable to high-quality traditional engraving. Fig. 1.14: Different sheet music representations corresponding to same score representation of beginning of Prelude BWV 846 (C major) by Johann Sebastian Bach. From top left to bottom right, a computer-generated, a handwritten, & 2 traditionally engraved representations are shown. illustrates this by showing different sheet music representations corresponding to same score.

– Biểu diễn bản nhạc. Trong lớp biểu diễn âm nhạc tượng trưng, muốn phân biệt 1 phân lớp được gọi là *biểu diễn bản nhạc*. 1 biểu diễn từ phân lớp này được định nghĩa để tạo ra thông tin rõ ràng về các ký hiệu âm nhạc, ví dụ như hệ thống khuông nhạc, khóa nhạc, nhịp điệu, nốt nhạc, dấu lặng, dấu hóa, & cường độ. Theo nghĩa này, biểu diễn bản nhạc, so với biểu diễn MIDI, gần hơn nhiều với những gì thực sự được hiển thị trong bản nhạc. E.g., trong biểu diễn bản nhạc, các nốt Rê thăng 4 & Mi giáng b có thể phân biệt được, & thời điểm bắt đầu của bản nhạc được chỉ định. Tuy nhiên, biểu diễn bản nhạc không được chứa mô tả về bố cục cuối cùng & hình dạng cụ thể của các ký hiệu âm nhạc. Quá trình tạo hoặc hiển thị các biểu diễn bản nhạc đẹp mắt từ các biểu diễn bản nhạc là 1 nghệ thuật. Trước đây, nghệ thuật vẽ ký hiệu âm nhạc chất lượng cao để tái tạo cơ học được gọi là *khắc nhạc*. Ngày nay, phần mềm máy tính hoặc *scorewriters* đã được thiết kế cho mục đích viết, biên tập, & in nhạc, mặc dù chỉ 1 số ít tạo ra kết quả tương đương với bản khắc truyền thống chất lượng cao. Hình 1.14: Các biểu diễn bản nhạc khác nhau tương ứng với cùng 1 biểu diễn bản nhạc của phần đầu Prelude BWV 846 (C trưởng) của Johann Sebastian Bach. Từ trên cùng bên trái xuống dưới cùng bên phải, 1 bản do máy tính tạo ra, 1 bản viết tay, & 2 biểu diễn được khắc theo cách truyền thống được hiển thị. minh họa điều này bằng cách hiển thị các biểu diễn bản nhạc khác nhau tương ứng với cùng 1 bản nhạc.

In this book, do not give an overview of existing symbolic score formats. Instead, e.g., discuss some aspects of *MusicXML*, which has been developed to serve as a universal format for storing music files & sharing them between different music notation applications. Following general XML (Extensible Markup Language) paradigm, MusicXML is a textual data format that defines a set of rules for encoding documents in a way that is both human & machine readable. E.g., Fig.

1.15: Textual description in MusicXML format of a half note E♭4. Clef, key signature, & time signature are defined at beginning of MusicXML file. shows how a note E♭4 is encoded. In MusicXML encoding of half note E♭4, tags `<note>, </note>` mark beginning & end of a MusicXML note element. Pitch element, delimited by tags `<pitch>, </pitch>`, consists of a pitch class element `E` (denoting letter name of pitch), alter element −1 (changing E to E flat), & octave element 4 (fixing octave). Thus, resulting note is an E♭4. Element `<duration>2</duration>` encodes duration of note measured in quarter notes. Finally, element `<type>half</type>` tells us how this note is actually depicted in rendered sheet music.

– Trong cuốn sách này, không cung cấp tổng quan về các định dạng bản nhạc ký hiệu hiện có. Thay vào đó, ví dụ, thảo luận 1 số khía cạnh của *MusicXML*, được phát triển để phục vụ như 1 định dạng chung để lưu trữ các tệp nhạc & chia sẻ chúng giữa các ứng dụng ký hiệu âm nhạc khác nhau. Theo mô hình XML (Ngôn ngữ đánh dấu mở rộng) chung, MusicXML là 1 định dạng dữ liệu văn bản xác định 1 tập hợp các quy tắc để mã hóa tài liệu theo cách mà cả con người & máy có thể đọc được. Ví dụ: Hình 1.15: Mô tả văn bản trong định dạng MusicXML của 1 nốt đen E♭4. Khóa nhạc, dấu hóa, & nhịp được xác định ở đầu tệp MusicXML. cho thấy cách 1 nốt E♭4 được mã hóa. Trong mã hóa MusicXML của nốt đen E♭4, các thẻ `<note>, </note>` đánh dấu bắt đầu & kết thúc của 1 phần tử nốt MusicXML. Phần tử cao độ, được phân cách bằng các thẻ `<pitch>, </pitch>`, bao gồm 1 phần tử lớp cao độ `E` (biểu thị tên chữ cái của cao độ), phần tử thay đổi −1 (thay đổi E thành E giáng), & phần tử quãng tám 4 (cố định quãng tám). Do đó, nốt nhạc kết quả là E♭4. Phần tử `<duration>2</duration>` mã hóa độ dài của nốt nhạc được đo bằng nốt đen. Cuối cùng, phần tử `<type>half</type>` cho chúng ta biết nốt nhạc này thực sự được mô tả như thế nào trong bản nhạc đã kết xuất.

There are various ways to generate digital score representations. E.g., one could manually input score information in a format e.g. MusicXML. This, however, is a tedious & error-prone procedure. Music notation software or scorewriters support users in task of writing & editing digitized sheet music. Such software allows a user to conveniently input & modify note objects by standard computer input devices or electronic keyboards. In next sect, discuss another way for generating score representation from scanned images of printed sheet music, which is, in a sense, inverse of a rendering process.

– Có nhiều cách khác nhau để tạo biểu diễn bản nhạc số. E.g., người ta có thể nhập thủ công thông tin bản nhạc theo định dạng ví dụ như MusicXML. Tuy nhiên, đây là 1 quy trình & dễ xảy ra lỗi. Phần mềm ký hiệu âm nhạc hoặc trình soạn nhạc hỗ trợ người dùng trong nhiệm vụ viết & chỉnh sửa bản nhạc số hóa. Phần mềm như vậy cho phép người dùng nhập & sửa đổi các đối tượng nốt nhạc 1 cách thuận tiện bằng các thiết bị đầu vào máy tính tiêu chuẩn hoặc bàn phím điện tử. Trong phần tiếp theo, hãy thảo luận về 1 cách khác để tạo biểu diễn bản nhạc từ hình ảnh được quét của bản nhạc đã in, theo 1 nghĩa nào đó, là ngược lại với quy trình kết xuất.

∗ 1.2.4. Optical Music Recognition. Sheet music is widely available, & many people are trained to use music notation for studying & playing music. For centuries, music has been documented, transmitted, & distributed in form of printed sheet music. Music libraries & archives possess huge collections comprising millions of sheet music books, which are now successively being transferred into digital domain using scanning devices. A digital image resulting from such a scanning process consists of a number of rows & columns of pixels, each pixel encoding color at a specific point of scanned page. I.e., a digital image of a sheet music page is by itself a mere accumulation of colored (often black & white) pixels without expressing & deeper musical meaning.

– Nhận dạng nhạc quang học. Bản nhạc có sẵn rộng rãi, & nhiều người được đào tạo để sử dụng ký hiệu âm nhạc để học & chơi nhạc. Trong nhiều thế kỷ, âm nhạc đã được ghi chép, truyền tải, & phân phối dưới dạng bản nhạc in. Các thư viện âm nhạc & lưu trữ sở hữu các bộ sưu tập khổng lồ bao gồm hàng triệu cuốn sách bản nhạc, hiện đang được chuyển liên tục vào miền kỹ thuật số bằng các thiết bị quét. 1 hình ảnh kỹ thuật số thu được từ quá trình quét như vậy bao gồm 1 số hàng & cột pixel, mỗi pixel mã hóa màu tại 1 điểm cụ thể của trang được quét. Tức là, 1 hình ảnh kỹ thuật số của 1 trang bản nhạc tự nó chỉ là sự tích tụ của các pixel có màu (thường là đen & trắng) mà không thể hiện & ý nghĩa âm nhạc sâu sắc hơn.

Process of converting digital images of sheet music into symbolic music representations e.g. MIDI or MusicXML is commonly referred to as *optical music recognition* (OMR). [Equivalent in text domain is known as *optical character recognition* (OCR) with goal of converting scanned images of printed text into machine-encoded text.] During this process, image pixels have to be suitably grouped & interpreted in terms of musical symbols. This process is not easy, because of many ways musical symbols may be engraved into sheet music. As discussed in last sect & illustrated by Fig. 1.14, there may be substantial variations in layout of symbols & staff system. Symbols do not always look exactly same across different editions & may also be degraded in quality by artifacts of printing or scanning process. Furthermore, musical symbols often intersect with staff lines, & several symbols may be stacked & combined (e.g., several notes sharing same stem or combined with a beam). As a result, musical scores & interrelations between musical symbols can become quite complex.

– Quá trình chuyển đổi hình ảnh kỹ thuật số của bản nhạc thành biểu diễn âm nhạc tượng trưng, e.g., MIDI hoặc MusicXML thường được gọi là *nhận dạng nhạc quang học* (OMR). [Tương đương trong miền văn bản được gọi là *nhận dạng ký tự quang học* (OCR) với mục tiêu chuyển đổi hình ảnh được quét của văn bản in thành văn bản được mã hóa bằng máy.] Trong quá trình này, các điểm ảnh của hình ảnh phải được nhóm lại 1 cách phù hợp & diễn giải theo các ký hiệu âm nhạc. Quá trình này không dễ dàng, vì có nhiều cách để khắc các ký hiệu âm nhạc vào bản nhạc. Như đã thảo luận trong phần trước & minh họa bằng Hình 1.14, có thể có những thay đổi đáng kể trong cách bố trí các ký hiệu & hệ thống khuông nhạc. Các ký hiệu không phải lúc nào cũng giống hệt nhau trong các phiên bản khác nhau & cũng có thể bị giảm chất lượng do các hiện tượng lạ của quá trình in hoặc quét. Hơn nữa, các ký hiệu âm nhạc thường giao nhau với các dòng khuông nhạc, & 1 số ký hiệu có thể được xếp chồng & kết hợp (ví dụ: nhiều nốt nhạc có chung 1 thân hoặc kết hợp với 1 nhịp). Do đó, bản nhạc & mối quan hệ giữa các ký hiệu âm nhạc có thể trở nên khá phức tạp.

Correctly recognizing & interpreting meaning of all musical symbols is easy for a trained human, but hard for a computer. Fig. 1.16(a) Examples of typical OMR errors (top: original score; bottom: OMR result). shows some examples of typical errors produced by automated OMR procedures. Some of these errors e.g. missing notes, flags or beams are of local nature, while other errors, e.g. an incorrectly detected key signature, affect all notes of a staff line. Even worse is presence of a *transposing instrument*, whose music is notated at a pitch different from pitch that is actually played (Fig. 1.16(c) Transposed instruments often not interpreted correctly by OMR.). E.g., a clarinet in B♭ is a transposed instrument, where a C in a score sounds like a B♭. Missing this information, which is encoded in textual form in front of a staff line, leads to a misrepresentation of all notes' pitches. A score may also contain repeat signs with alternative endings or textual jump directives as shown in Fig. 1.16(b) Jump directives & repeats often not detected by OMR. This information is required to derive correct sequence of measures to be performed by a musician. Consequently, an error in detecting jump directives may lead to structural misinterpretations of score. Another problem: even small artifacts in scan may lead to confusion with musical symbols, e.g., a small dot being mixed up with a staccato mark. Even though current OMR software is reported to yield highly accurate results, manual postprocessing still seems necessary to obtain high-quality symbolic representation.

– Việc nhận dạng & diễn giải đúng ý nghĩa của tất cả các ký hiệu âm nhạc là điều dễ dàng đối với con người được đào tạo, nhưng lại khó đối với máy tính. Hình 1.16(a) Ví dụ về các lỗi OMR điển hình (trên cùng: bản nhạc gốc; dưới cùng: kết quả OMR). cho thấy 1 số ví dụ về các lỗi điển hình do các quy trình OMR tự động tạo ra. 1 số lỗi này, ví dụ như thiếu nốt, cờ hoặc dầm, có bản chất cục bộ, trong khi các lỗi khác, ví dụ như phát hiện không đúng dấu khóa, ảnh hưởng đến tất cả các nốt của 1 dòng khuông nhạc. Tệ hơn nữa là sự hiện diện của 1 *nhạc cụ chuyển vị*, có bản nhạc được ký hiệu ở cao độ khác với cao độ thực sự được chơi (Hình 1.16(c) Các nhạc cụ chuyển vị thường không được OMR diễn giải chính xác.). E.g., 1 cây kèn clarinet trong B♭ là 1 nhạc cụ chuyển vị, trong đó nốt C trong 1 bản nhạc nghe giống như nốt B♭. Thiếu thông tin này, được mã hóa dưới dạng văn bản trước 1 dòng khuông nhạc, dẫn đến việc trình bày sai cao độ của tất cả các nốt. 1 bản nhạc cũng có thể chứa các ký hiệu lặp lại với các kết thúc thay thế hoặc các chỉ thị nhảy theo văn bản như được hiển thị trong Hình 1.16(b) Chỉ thị nhảy & lặp lại thường không được OMR phát hiện. Thông tin này là cần thiết để suy ra trình tự chính xác của các ô nhịp do 1 nhạc sĩ thực hiện. Do đó, 1 lỗi trong việc phát hiện các chỉ thị nhảy có thể dẫn đến việc hiểu sai về mặt cấu trúc của bản nhạc. 1 vấn đề khác: ngay cả các hiện vật nhỏ trong quá trình quét cũng có thể dẫn đến nhầm lẫn với các ký hiệu âm nhạc, ví dụ, 1 dấu chấm nhỏ bị trộn lẫn với 1 dấu ngắt quãng. Mặc dù phần mềm OMR hiện tại được báo cáo là mang lại kết quả có độ chính xác cao, nhưng việc xử lý hậu kỳ thủ công dường như vẫn cần thiết để có được biểu diễn ký hiệu chất lượng cao.

- 1.3. Audio Representation. Music is much more than a symbolic description of notes to be played. Music is about making, creating, & shaping sounds. When musicians start delving into music, playing instructions recede into background. Musical meter turns into a rhythmic flow, different note objects melt into harmonic sounds & smooth melody lines, & instruments communicate with each other. Musicians get emotionally involved with their music & react to it by continuously adapting tempo, dynamics, & articulation. Instead of playing mechanically, they speed up at some points & slow down at others in order to shape a piece of music. Similarly, they continuously change sound intensity & stress certain notes. All of this results in a unique performance or an interpretation of piece of music.

  – Biểu diễn âm thanh. Âm nhạc còn hơn cả 1 mô tả tượng trưng về các nốt nhạc cần chơi. Âm nhạc là về việc tạo ra, sáng tạo, & định hình âm thanh. Khi các nhạc sĩ bắt đầu đào sâu vào âm nhạc, các hướng dẫn chơi nhạc sẽ lùi vào nền. Nhịp điệu âm nhạc chuyển thành 1 dòng chảy nhịp nhàng, các đối tượng nốt nhạc khác nhau hòa quyện thành âm thanh hài hòa & các giai điệu mượt mà, & các nhạc cụ giao tiếp với nhau. Các nhạc sĩ tham gia vào âm nhạc của họ 1 cách đầy cảm xúc & phản ứng với nó bằng cách liên tục điều chỉnh nhịp độ, cường độ, & cách phát âm. Thay vì chơi 1 cách máy móc, họ tăng tốc ở 1 số điểm & chậm lại ở những điểm khác để định hình 1 bản nhạc. Tương tự như vậy, họ liên tục thay đổi cường độ âm thanh & nhấn mạnh 1 số nốt nhạc nhất định. Tất cả những điều này dẫn đến 1 màn trình diễn độc đáo hoặc 1 cách diễn giải 1 bản nhạc.

  From a physical point of view, performing music results in *sounds* or *acoustic waves*, which are transmitted through air as pressure oscillations. Term *audio* is used to refer to transmission, reception, or reproduction of sounds that lie within limits of human hearing. An *audio signal* is a representation of sound. As opposed to sheet music & symbolic representations, an audio representation encodes all information needed to reproduce an acoustic realization of a piece of music. This includes temporal, dynamic, & tonal microdeviations that make up specific performance style of a musician. However, in an audio representation, note parameters e.g. onset times, pitches or note durations are not given explicitly. This makes analysis & comparison of music signals a difficult task, in particular with regard to polyphonic music, where different instruments & voices are superimposed upon each other. Furthermore, perception of sounds does not only depend on objective properties of acoustic wave, but also on subjective criteria as a result of complex processing a sound undergoes by both human ear & brain. Study of subjective human sound perception is called *psychoacoustics* – for further details see [5, 17]. In this sect, after having a look at waves & waveforms, summarize most important properties of audio representations: frequency & pitch, dynamics, intensity & loudness, as well as timbre.

  – Theo quan điểm vật lý, biểu diễn âm nhạc tạo ra *sounds* hoặc *acoustic waves*, được truyền qua không khí dưới dạng dao động áp suất. Thuật ngữ *audio* được sử dụng để chỉ việc truyền, tiếp nhận hoặc tái tạo âm thanh nằm trong giới hạn thính giác của con người. *audio signal* là 1 biểu diễn của âm thanh. Trái ngược với bản nhạc & biểu diễn ký hiệu, 1 biểu diễn âm thanh mã hóa tất cả thông tin cần thiết để tái tạo hiện thực hóa âm thanh của 1 bản nhạc. Điều này bao gồm các vi sai về thời gian, động, & âm sắc tạo nên phong cách biểu diễn cụ thể của 1 nhạc sĩ. Tuy nhiên, trong 1 biểu diễn âm thanh, các tham số nốt nhạc như thời điểm bắt đầu, cao độ hoặc thời lượng nốt nhạc không được nêu rõ ràng. Điều này khiến việc phân tích & so sánh các tín hiệu âm nhạc trở thành 1 nhiệm vụ khó khăn, đặc biệt là đối với âm nhạc đa âm, trong đó các

nhạc cụ & giọng hát khác nhau được chồng lên nhau. Hơn nữa, nhận thức về âm thanh không chỉ phụ thuộc vào các đặc tính khách quan của sóng âm mà còn phụ thuộc vào các tiêu chí chủ quan do quá trình xử lý phức tạp mà âm thanh trải qua bởi cả tai & não của con người. Nghiên cứu về nhận thức âm thanh chủ quan của con người được gọi là *tâm lý học âm thanh* – để biết thêm chi tiết, hãy xem [5, 17]. Trong phần này, sau khi xem xét sóng & dạng sóng, hãy tóm tắt các đặc tính quan trọng nhất của biểu diễn âm thanh: tần số & cao độ, động lực, cường độ & độ to, cũng như âm sắc.

∗ 1.3.1. Waves & Waveforms. A *sound* is generated by a vibrating object e.g. vocal cords of a singer, string & soundboard of a violin, diaphragm of a kettledrum, or prongs of a tuning fork. These vibrations cause displacements & oscillations of air molecules, resulting in local regions of compression & rarefaction. Alternating pressure travels through air as a *wave*, from its source to a listener or a microphone. At its destination, it can then be perceived as sound by human or converted into an electrical signal by a microphone (Fig. 1.17: Vibrating tuning fork resulting in a back & forth vibration of surrounding air particles. Pressure oscillation propagates as a longitudinal wave through air. Waveform shows deviation over time of air pressure from average air pressure at a specific location (as indicated by microphone).). In case of a listener, outer part of ear captures sound wave & passes it to eardrum, which in turn starts vibrating according to pressure oscillations. After further processing in middle & inner ear, sound wave is transformed into nerve impulses, which are finally sent to & interpreted by brain. Graphically, change in air pressure at a certain location can be represented by a *pressure–time plot*, also referred to as *waveform* of sound. Waveform shows deviation of air pressure from average air pressure. Fig. 1.18: (a) Waveform of 1st 8 secs of a recording of 1st 5 measures of Beethoven's 5th as indicated by Fig. 1.1. (b) Enlargement of sect between 7.3 & 7.8 s. shows a waveform representation of a recording of Beethoven's 5th Symphony.

– Sóng & Dạng sóng. Âm thanh được tạo ra bởi 1 vật rung, ví dụ như dây thanh quản của ca sĩ, dây đàn & mặt cộng hưởng của đàn violin, màng loa của trống cái, hoặc chấu của 1 âm thoa. Những rung động này gây ra sự dịch chuyển & dao động của các phân tử không khí, dẫn đến các vùng nén & loãng cục bộ. Áp suất thay đổi truyền qua không khí dưới dạng *sóng*, từ nguồn của nó đến người nghe hoặc micrô. Tại đích đến, sau đó nó có thể được con người cảm nhận dưới dạng âm thanh hoặc được micrô chuyển đổi thành tín hiệu điện (Hình 1.17: Âm thoa rung dẫn đến rung & tới của các hạt không khí xung quanh. Dao động áp suất lan truyền dưới dạng sóng dọc trong không khí. Dạng sóng cho thấy độ lệch theo thời gian của áp suất không khí so với áp suất không khí trung bình tại 1 vị trí cụ thể (như micrô chỉ ra).). Trong trường hợp của người nghe, phần ngoài của tai bắt sóng âm & truyền đến màng nhĩ, sau đó màng nhĩ bắt đầu rung theo dao động áp suất. Sau khi xử lý thêm ở tai giữa & trong, sóng âm được chuyển thành xung thần kinh, cuối cùng được gửi đến & não diễn giải. Về mặt đồ họa, sự thay đổi áp suất không khí tại 1 vị trí nhất định có thể được biểu diễn bằng *biểu đồ áp suất-thời gian*, còn được gọi là *dạng sóng* của âm thanh. Dạng sóng cho thấy độ lệch của áp suất không khí so với áp suất không khí trung bình. Hình 1.18: (a) Dạng sóng của 8 giây đầu tiên của bản ghi âm 5 nhịp đầu tiên của Bản giao hưởng số 5 của Beethoven như được chỉ ra trong Hình 1.1. (b) Phóng to phần giữa 7,3 & 7,8 giây cho thấy biểu diễn dạng sóng của bản ghi âm Bản giao hưởng số 5 của Beethoven.

In general terms, a (mechanical) *wave* can be described as an oscillation that travels through space, where energy is transferred from 1 point to another. When a wave travels through some medium, substance of this medium is temporarily deformed. Sound waves propagate via air molecules colliding with their neighbors. After air molecules collide, they bounce away from each other (a restoring force). This keeps molecules from continuing to travel in direction of wave. Instead, they oscillate around almost fixed locations. A general wave can be *transverse* or *longitudinal*, depending on direction of its oscillation. Transverse waves occur when a disturbance creates oscillations perpendicular (at right angles) to propagation (direction of energy transfer). Longitudinal waves occur when oscillations are parallel to direction of propagation. According to this definition, a vibration in a string is an example of a transverse wave, whereas a sound wave has form of a longitudinal wave. A transverse wave can in fact generate a longitudinal wave & vice versa. An instrument's vibrating string, which oscillates between 2 fixed end points, gradually emits its energy to air, generating a longitudinal sound wave. If this wave, in turn, hits an eardrum, again a transverse wave is generated.

– Nói chung, 1 sóng (cơ học) có thể được mô tả là 1 dao động truyền qua không gian, trong đó năng lượng được truyền từ điểm này sang điểm khác. Khi sóng truyền qua 1 môi trường nào đó, chất của môi trường này bị biến dạng tạm thời. Sóng âm lan truyền qua các phân tử không khí va chạm với các phân tử lân cận. Sau khi các phân tử không khí va chạm, chúng nảy ra xa nhau (một lực phục hồi). Điều này ngăn các phân tử tiếp tục di chuyển theo hướng sóng. Thay vào đó, chúng dao động quanh các vị trí gần như cố định. 1 sóng tổng quát có thể là *ngang* hoặc *dọc*, tùy thuộc vào hướng dao động của nó. Sóng ngang xảy ra khi nhiễu động tạo ra dao động vuông góc (vuông góc) với hướng truyền (hướng truyền năng lượng). Sóng dọc xảy ra khi dao động song song với hướng truyền. Theo định nghĩa này, dao động trong 1 sợi dây là 1 ví dụ về sóng ngang, trong khi sóng âm có dạng sóng dọc. Sóng ngang thực tế có thể tạo ra sóng dọc & ngược lại. Dây rung của nhạc cụ, dao động giữa 2 điểm cố định, dần dần phát ra năng lượng của nó vào không khí, tạo ra sóng âm dọc. Nếu sóng này, đến lượt nó, đập vào màng nhĩ, 1 sóng ngang nữa lại được tạo ra.

∗ 1.3.2. Frequency & Pitch. Have seen: a sound wave can be visually represented by a waveform. If points of high & low air pressure repeat in an alternating & regular fashion, resulting waveform is called *periodic*. In this case, *period* of wave is defined as time required to complete a cycle. *Frequency*, measured in *Hertz* (Hz), is reciprocal of period. Fig. 1.19: Waveform of a sinusoid with a frequency of 4 Hz. shows a *sinusoid*, which is simplest type of periodic waveform. In this example, waveform has a period of a quarter sec & hence a frequency of 4 Hz. A sinusoid is completely specified by its frequency, its *amplitude* (peak deviation of sinusoid from its mean), & its *phase* (determining where in its cycle sinusoid is at time 0). These 3 attributes of a sinusoid will become important when analyzing general audio signals (Sect. 2.3).

– Tần số & Cao độ. Đã thấy: sóng âm có thể được biểu diễn trực quan bằng dạng sóng. Nếu các điểm có áp suất không khí cao & thấp lặp lại theo cách xen kẽ & đều đặn, dạng sóng kết quả được gọi là *tuần hoàn*. Trong trường hợp này, *chu kỳ* của sóng được định nghĩa là thời gian cần thiết để hoàn thành 1 chu kỳ. *Tần số*, được đo bằng *Hertz* (Hz), là nghịch

đảo của chu kỳ. Hình 1.19: Dạng sóng của sóng sin có tần số 4 Hz. cho thấy 1 *sin*, là loại sóng tuần hoàn đơn giản nhất. Trong ví dụ này, dạng sóng có chu kỳ là 1 phần tư giây & do đó có tần số là 4 Hz. 1 sóng sin được xác định hoàn toàn bởi tần số, *biên độ* (độ lệch cực đại của sóng sin so với giá trị trung bình), & *pha* (xác định vị trí của sóng sin trong chu kỳ của nó tại thời điểm 0). 3 thuộc tính này của sóng sin sẽ trở nên quan trọng khi phân tích các tín hiệu âm thanh nói chung (Phần 2.3).
Higher frequency of a sinusoidal wave, higher it sounds. Audible frequency range for humans is between about 20 Hz & 20000 Hz (20 kHz). Other species have different hearing ranges. E.g., top end of a dog's hearing range is about 45 kHz, a cat's is 64 kHz, while bats can even detect frequencies beyond 100 kHz. This is why one can use a dog whistle, which emits *ultrasonic sound* beyond human hearing capability, to train & to command animals without disturbing nearby people.
– Tần số càng cao của sóng sin, âm thanh càng lớn. Dải tần số âm thanh mà con người có thể nghe được nằm trong khoảng từ 20 Hz & 20000 Hz (20 kHz). Các loài khác có dải tần số nghe khác nhau. E.g., giới hạn trên của dải tần số nghe của chó là khoảng 45 kHz, của mèo là 64 kHz, trong khi dơi thậm chí có thể phát hiện tần số vượt quá 100 kHz. Đây là lý do tại sao người ta có thể sử dụng còi chó, phát ra *âm thanh siêu âm* vượt quá khả năng nghe của con người, để huấn luyện & ra lệnh cho động vật mà không làm phiền những người ở gần.
Sinusoid can be considered prototype of an acoustic realization of a musical note. Sometimes sound resulting from a sinusoid is called a *harmonic sound* or *pure tone*.As indicated in Sect. 1.1.1, notion of frequency is closely related to what determines *pitch* of a sound. In general, pitch is a subjective attribute of sound. In case of complex sound mixtures its relation to frequency can be especially ambiguous. In case of pure tones, however, relation between frequency & pitch is clear. E.g., a sinusoid having a frequency of 440 Hz corresponds to pitch A4. This particular pitch is known as *concert pitch*, & it is used as reference pitch to which a group of musical instruments are tuned for a performance. Sine a slight change in frequency does not necessarily lead to a perceived change, one usually associates an entire range of frequencies with a single pitch.
– Sóng sin có thể được coi là nguyên mẫu của 1 hiện thực âm thanh của 1 nốt nhạc. Đôi khi âm thanh phát ra từ sóng sin được gọi là *âm thanh hài hòa* hoặc *âm thuần túy*. Như đã chỉ ra trong Mục 1.1.1, khái niệm tần số có liên quan chặt chẽ đến yếu tố quyết định *cao độ* của âm thanh. Nhìn chung, cao độ là 1 thuộc tính chủ quan của âm thanh. Trong trường hợp các hỗn hợp âm thanh phức tạp, mối quan hệ của nó với tần số có thể đặc biệt mơ hồ. Tuy nhiên, trong trường hợp các âm thuần túy, mối quan hệ giữa tần số & cao độ là rõ ràng. E.g., sóng sin có tần số 440 Hz tương ứng với cao độ A4. Cao độ cụ thể này được gọi là *cao độ hòa nhạc*, & nó được sử dụng làm cao độ tham chiếu mà 1 nhóm nhạc cụ được lên dây để biểu diễn. Vì 1 thay đổi nhỏ về tần số không nhất thiết dẫn đến sự thay đổi được nhận thức, người ta thường liên kết toàn bộ 1 dải tần số với 1 cao độ duy nhất.
2 frequencies are perceived as similar if they differ by a power of 2, which has motivated notion of an octave. E.g., pitches A3 (220 Hz), A4 (440 Hz), & A5 (880 Hz) sound similar. Furthermore, perceived distance between pitches A3 & A4 is same as perceived distance between pitches A4 & A5. I.e., human perception of pitch is logarithmic in nature. This perceptual property has already been used in Sect. 1.1.1 when defining equal-tempered scale that subdivides an octave into 12 semitones based on a logarithmic frequency axis. More formally, using MIDI note numbers introduced in Sect. 1.2.2, can associate to each pitch $p \in [0:127]$ a *center frequency* $F_{\text{pitch}}(p)$ (measured in Hz) by (1.1)

$$F_{\text{pitch}}(p) = 2^{\frac{p-69}{12}} \cdot 440.$$

Indeed, this formula yields frequency $F_{\text{pitch}}(p) = 440$ for reference pitch $p = 69$ (A4). Increasing pitch number by 12 (an octave) leads to an increase by a factor of 2, i.e., $F_{\text{pitch}}(p+12) = 2 \cdot F_{\text{pich}}(p)$. Similarly, easy to show: frequency ratio

$$\frac{F_{\text{pitch}}(p+1)}{F_{\text{pitch}}(p)} = 2^{\frac{1}{12}} \approx 1.059463$$

of 2 subsequent pitches $p+1$ & p is constant (Exercise 1.6)

Problem 2. *Using (1.1), compute frequency ratio* $\frac{F_{\text{pitch}}(p+1)}{F_{\text{pitch}}(p)}$ *of 2 subsequent pitches* $p+1, p$ *(see (1.2)). How does frequency* $F_{\text{pitch}}(p+k)$ *for some* $k \in \mathbb{Z}$ *relate to* $F_{\text{pitch}}(p)$*? Furthermore, derive a formula for distance (in semitones) for 2 arbitrary frequencies* ω_1, ω_2.

I.e., multiplying center frequency of an arbitrary pitch by this constant, pitch is raised by a semitone. Generalizing notion of semitones, *cent* denotes a logarithmic unit of measure used for musical intervals. By def, an octave is divided into 1200 cents, so that each semitone corresponds to 100 cents. Again ratio of frequencies 1 cent apart is constant, yielding value

$$2^{\frac{1}{1200}} \approx 1.0005777895.$$

Difference in cents between 2 frequencies, say ω_1, ω_2, is given by

$$\log_2 \frac{\omega_1}{\omega_2} \cdot 1200.$$

Interval of 1 cent is much too small to be heard between successive notes. Threshold of what is perceptible, also called *just noticeable difference*, varies from person to person & depends on other aspects e.g. timbre (Sect. 1.3.4) & musical context. As a rule of thumb, normal adults are able to recognize pitch differences as small as 25 cents very reliably, with differences of 10 cents being recognizable only by trained listeners.
– Khoảng cách 1 cent là quá nhỏ để có thể nghe thấy giữa các nốt nhạc liên tiếp. Ngưỡng của những gì có thể nhận biết được, còn được gọi là *chỉ là sự khác biệt đáng chú ý*, thay đổi tùy theo từng người & phụ thuộc vào các khía cạnh khác,

ví dụ như âm sắc (Phần 1.3.4) & ngữ cảnh âm nhạc. Theo nguyên tắc chung, người lớn bình thường có thể nhận ra sự khác biệt về cao độ nhỏ tới 25 cent 1 cách rất đáng tin cậy, với sự khác biệt 10 cent chỉ có thể nhận ra được bởi những người nghe được đào tạo.

Real-world sounds are far from being a simple pure tone with a well-defined frequency. Playing a single note on an instrument may result in a complex sound that contains a mixture of different frequencies changing over time. Intuitively, such a *musical tone* can be described as a superposition of pure tones or sinusoids, each with its own frequency of vibration, amplitude, & phase. A *partial* is any of sinusoids by which a musical tone is described. Frequency of lowest partial present is called *fundamental frequency* of sound. Pitch of a musical tone is usually determined by fundamental frequency, which is the one created by vibration over full length of a string or air column of an instrument. A *harmonic* (or a *harmonic partial*) is a partial that is an integer multiple of fundamental frequency. Partials, as well as harmonics, are counted upwards along frequency axis. This convention implies: fundamental frequency is 1st partial, as well as 1st harmonic of a musical tone. Term *inharmonicity* is used to denote a measure of deviation of a partial from closest ideal harmonic, typically measured in cents for each partial. Finally, another term often used in music theory is *overtone*, which is any partial except lowest. This can lead to numbering confusion when comparing overtones with partials, since 1st overtone is 2nd partial.

– Âm thanh trong thế giới thực không phải là 1 âm thanh thuần túy đơn giản với tần số được xác định rõ ràng. Chơi 1 nốt nhạc duy nhất trên 1 nhạc cụ có thể tạo ra 1 âm thanh phức tạp chứa hỗn hợp các tần số khác nhau thay đổi theo thời gian. Theo trực giác, 1 *giai điệu âm nhạc* như vậy có thể được mô tả là sự chồng chất của các âm thanh thuần túy hoặc sin, mỗi âm thanh có tần số rung động, biên độ, & pha riêng. 1 *1 phần* là bất kỳ sin nào mà theo đó 1 âm nhạc được mô tả. Tần số của phần thấp nhất hiện tại được gọi là *tần số cơ bản* của âm thanh. Cao độ của 1 âm nhạc thường được xác định bởi tần số cơ bản, là tần số được tạo ra bởi sự rung động trên toàn bộ chiều dài của 1 dây đàn hoặc cột không khí của 1 nhạc cụ. 1 *hài hòa* (hoặc *1 phần hài hòa*) là 1 phần là bội số nguyên của tần số cơ bản. Các phần, cũng như các hài, được đếm theo hướng lên trên trục tần số. Quy ước này ngụ ý rằng: tần số cơ bản là phần thứ nhất, cũng như hài thứ nhất của 1 âm nhạc. Thuật ngữ *inharmonicity* được sử dụng để biểu thị 1 phép đo độ lệch của 1 phần so với hài hòa lý tưởng gần nhất, thường được đo bằng cent cho mỗi phần. Cuối cùng, 1 thuật ngữ khác thường được sử dụng trong lý thuyết âm nhạc là *overtone*, là bất kỳ phần nào ngoại trừ phần thấp nhất. Điều này có thể dẫn đến nhầm lẫn về số lượng khi so sánh các âm bội với các phần, vì âm bội thứ nhất là phần thứ hai.

Most pitched instruments are designed to have partials that are close to being harmonics, with very low inharmonicity. Thus, for simplicity, one often speaks of partials in those instruments' sounds as harmonics, even if they have some inharmonicity. Other pitched instruments, especially certain percussion instruments, e.g. marimba, vibraphone, bells, & kettledrums (timpani), contain nonharmonic partials, yet give ear a good sense of pitch. Nonpitched, or indefinite-pitched, instruments, e.g. cymbals, gongs, or tam-tams, make sounds rich in inharmonic partials. As an example of a harmonic sound, Fig. 1.18 shows in its lower part an enlargement of waveform of sect between 7.3 & 7.8 secs, which reveals almost periodic nature of sound signal. Waveform within these 500 ms corresponds to sound of a decaying D, which is played by orchestra in unison of 4th & 5th measure (Fig. 1.1). Indeed, one counts 37 periods within this sect, corresponding to a frequency of 74 Hz – fundamental frequency of D2.

– Hầu hết các nhạc cụ có cao độ được thiết kế để có các phần gần giống với hài âm, với độ bất hòa âm rất thấp. Do đó, để đơn giản, người ta thường nói về các phần trong âm thanh của các nhạc cụ đó là hài âm, ngay cả khi chúng có 1 số độ bất hòa âm. Các nhạc cụ có cao độ khác, đặc biệt là 1 số nhạc cụ gõ, ví dụ như đàn marimba, đàn vibraphone, chuông, & trống đồng (timpani), chứa các phần không hài hòa, nhưng vẫn mang lại cho tai cảm giác tốt về cao độ. Các nhạc cụ không có cao độ hoặc có cao độ không xác định, ví dụ như chũm chọe, cồng hoặc tam-tam, tạo ra âm thanh giàu các phần không hài hòa. Là 1 ví dụ về âm thanh hài hòa, Hình 1.18 cho thấy ở phần dưới của nó là sự mở rộng dạng sóng của phần giữa 7,3 & 7,8 giây, điều này cho thấy bản chất gần như tuần hoàn của tín hiệu âm thanh. Dạng sóng trong 500 ms này tương ứng với âm thanh của nốt D đang suy yếu, được dàn nhạc chơi đồng thanh ở nhịp 4 & 5 (Hình 1.1). Thật vậy, người ta đếm được 37 chu kỳ trong phần này, tương ứng với tần số 74 Hz – tần số cơ bản của D2.

Close this sect on frequency & pitch by looking at harmonics in terms of musical pitches. Let ω denote center frequency of a musical note, e.g., $\omega = 65.4$ Hz for C2 (having MIDI note number $p = 36$). Harmonic series is an arithmetic series $\omega, 2\omega, 3\omega, 4\omega, \ldots$, where difference between consecutive harmonics is constant & = fundamental. Since our perception of pitch is logarithmic in frequency, perceive higher harmonics as "closer together" than lower ones. On other hand, octave series is a geometric progression $\omega, 2\omega, 4\omega, 8\omega, \ldots$ & hear these distances as "the same" in sense of musical interval. Consequently, in terms of what we hear, each octave in harmonic series is divided into increasingly "smaller" & more numerous intervals. In our example, 2nd harmonic 2ω sounds like a C3 (1 octave higher), 3rd harmonic 3ω like a G3 (a so-called *perfect 5th* above C3), & 4th harmonic 4ω like a C4 (2 octaves higher). Starting with a C2, Fig. 1.20: Illustration of harmonic series in music notation. Starting with note C2, for each of 1st 16 harmonics closest musical note is shown. On top, difference (in cents) between a harmonic's frequency & center frequency of closest note is shown. shows for each of 1st 16 harmonics musical note that is closest in terms of difference between harmonic's frequency & center frequency of note as specified in (1.1) (see also Exercise 1.9):

Problem 3. *Write a small computer program to calculate differences (in cents) between 1st 16 harmonics of note C2 & center frequencies of closest notes of 12-tone equal-tempered scale (*Fig. 1.20*). What are corresponding differences when considering harmonics of another note e.g. B♭4?*

E.g., frequency of 3rd harmonic is just 2 cents above center frequency of G3, which is much smaller than just noticeable difference. In contrast, frequency of 11th harmonic is 49 cents below center frequency of note F♯5, which is nearly half a semitone & clearly audible. If harmonics are transposed into span of 1 octave (by suitably multiplying or dividing frequencies by a power of 2), they approximate certain notes of 12-tone equal-tempered scale. Some of 12 scale steps are

approximated well e.g. ones for C (1st harmonic), G (3rd harmonic), or D (9th harmonic), whereas others are problematic e.g. F♯ (11th harmonic), A♭ (13th harmonic), or B♭ (7th harmonic).

–Đóng phần này lại về tần số & cao độ bằng cách xem xét các họa âm theo cao độ âm nhạc. Giả sử ω biểu thị tần số trung tâm của 1 nốt nhạc, ví dụ, $\omega = 65,4$ Hz đối với C2 (có số nốt MIDI $p = 36$). Chuỗi họa âm là 1 chuỗi số học $\omega, 2\omega, 3\omega, 4\omega, \ldots$, trong đó sự khác biệt giữa các họa âm liên tiếp là hằng số & = cơ bản. Vì nhận thức của chúng ta về cao độ là logarit theo tần số, hãy cảm nhận các họa âm cao hơn là "gần nhau hơn" so với các họa âm thấp hơn. Mặt khác, chuỗi quãng tám là 1 cấp số nhân $\omega, 2\omega, 4\omega, 8\omega, \ldots$ & nghe những khoảng cách này là "giống nhau" theo nghĩa là khoảng cách âm nhạc. Do đó, xét về những gì chúng ta nghe được, mỗi quãng tám trong chuỗi họa âm được chia thành các khoảng cách ngày càng "nhỏ hơn" & nhiều hơn. Trong ví dụ của chúng tôi, âm bội thứ 2 2ω nghe giống như C3 (cao hơn 1 quãng tám), âm bội thứ 3 3ω giống như G3 (cái gọi là *hoàn hảo thứ 5* trên C3), & âm bội thứ 4 4ω giống như C4 (cao hơn 2 quãng tám). Bắt đầu bằng C2, Hình 1.20: Minh họa chuỗi âm bội trong ký hiệu âm nhạc. Bắt đầu bằng nốt C2, đối với mỗi âm bội đầu tiên trong số 16 âm bội, nốt nhạc gần nhất sẽ được hiển thị. Trên cùng, sự khác biệt (tính bằng cent) giữa tần số của 1 sóng hài & tần số trung tâm của nốt gần nhất được hiển thị. cho thấy đối với mỗi nốt nhạc sóng hài đầu tiên trong số 16 nốt nhạc sóng hài gần nhất về sự khác biệt giữa tần số của sóng hài & tần số trung tâm của nốt như được chỉ định trong (1.1) (xem thêm Bài tập 1.9):

1. *Viết 1 chương trình máy tính nhỏ để tính toán sự khác biệt (tính bằng cent) giữa 16 sóng hài đầu tiên của nốt C2 & tần số trung tâm của các nốt gần nhất của thang âm đồng âm 12 cung (*Hình 1.20*). Sự khác biệt tương ứng là gì khi xem xét sóng hài của 1 nốt khác, ví dụ B♭4?*

E.g., tần số của sóng hài thứ 3 chỉ cao hơn 2 cent so với tần số trung tâm của G3, nhỏ hơn nhiều so với sự khác biệt đáng chú ý. Ngược lại, tần số của hài bậc 11 thấp hơn 49 cent so với tần số trung tâm của nốt F♯5, gần bằng 1 nửa cung & nghe rõ. Nếu hài bậc 1 được chuyển sang quãng 1 quãng tám (bằng cách nhân hoặc chia tần số 1 cách thích hợp cho lũy thừa của 2), chúng sẽ xấp xỉ 1 số nốt nhất định của thang âm 12 cung có cùng tông. 1 số trong 12 bước thang âm được xấp xỉ tốt, ví dụ như đối với C (hài bậc 1), G (hài bậc 3) hoặc D (hài bậc 9), trong khi những bước khác lại có vấn đề, ví dụ như F♯ (hài bậc 11), A♭ (hài bậc 13) hoặc B♭ (hài bậc 7).

∗ 1.3.3. Dynamics, Intensity, & Loudness. A further important property of music concerns *dynamics*, a general term that is used to refer to volume of a sound as well as to musical symbols that indicate volume. E.g., a piano (notated as *p*) indicates: notes are to be played softly, whereas a forte (notated as *f*) indicates notes are to be played loudly. There are many more indicators for describing dynamics of notes in sheet music. On audio side, dynamics correlate with a perceptual property called *loudness*, by which sounds can be ordered on a scale extending from quiet to loud. Similarly to relation between pitch & frequency, loudness is a subjective measure which correlates to objective measures of *sound intensity* & *sound power*. However, loudness also depends on other sound characteristics e.g. duration or frequency. Will come back to some of these subjective phenomena after having a closer look at objective measures.

– Động lực, Cường độ, & Độ to. 1 tính chất quan trọng khác của âm nhạc liên quan đến *động lực*, 1 thuật ngữ chung được sử dụng để chỉ âm lượng của âm thanh cũng như các ký hiệu âm nhạc biểu thị âm lượng. E.g., 1 cây đàn piano (ký hiệu là *p*) biểu thị: các nốt nhạc phải được chơi nhẹ nhàng, trong khi 1 cây đàn forte (ký hiệu là *f*) biểu thị các nốt nhạc phải được chơi to. Có nhiều chỉ báo khác để mô tả động lực của các nốt nhạc trong bản nhạc. Về mặt âm thanh, động lực tương quan với 1 tính chất nhận thức được gọi là *độ to*, theo đó âm thanh có thể được sắp xếp theo thang âm kéo dài từ nhỏ đến lớn. Tương tự như mối quan hệ giữa cao độ & tần số, độ to là 1 phép đo chủ quan tương quan với các phép đo khách quan về *cường độ âm thanh* & *công suất âm thanh*. Tuy nhiên, độ to cũng phụ thuộc vào các đặc điểm âm thanh khác, ví dụ như thời lượng hoặc tần số. Sẽ quay lại 1 số hiện tượng chủ quan này sau khi xem xét kỹ hơn các phép đo khách quan.

From a physical point of view, not easy to strictly define intensity or power of a sound. In following, only give some intuitive explanations. In general, *power* is rate at which energy is transferred, used, or transformed. Power is measured in units of *watt* (W), which is defined as 1 joule per sec. E.g., rate at which a light bulb transforms electrical energy into heat & light is measured in watts – more wattage, more power, or equivalently more electrical energy is used per unit time. Similarly, *sound power* expresses how much energy per unit time is emitted by a sound source passing in all directions through air. Term *sound intensity* is then used to denote sound power per unit area.

– Theo quan điểm vật lý, không dễ để định nghĩa chính xác cường độ hoặc công suất của âm thanh. Sau đây, chỉ đưa ra 1 số giải thích trực quan. Nói chung, *power* là tốc độ năng lượng được truyền, sử dụng hoặc biến đổi. Công suất được đo bằng đơn vị *watt* (W), được định nghĩa là 1 joule trên giây. E.g., tốc độ bóng đèn biến đổi năng lượng điện thành nhiệt & ánh sáng được đo bằng watt – công suất lớn hơn, công suất lớn hơn hoặc tương đương là nhiều năng lượng điện hơn được sử dụng trên 1 đơn vị thời gian. Tương tự như vậy, *sound power* thể hiện lượng năng lượng trên 1 đơn vị thời gian được phát ra bởi 1 nguồn âm thanh truyền qua không khí theo mọi hướng. Sau đó, thuật ngữ *sound intensity* được sử dụng để biểu thị công suất âm thanh trên 1 đơn vị diện tích.

In practice, sound power & sound intensity can show extremely small values that are still relevant for human listeners. E.g., *threshold of hearing* (TOH), which is minimum sound intensity of a pure tone a human can hear, is as small as

$$I_{\text{TOH}} \coloneqq 10^{-12}\ \text{W/m}^2.$$

Range of intensities a human can perceive is extremely large with $I_{\text{TOP}} \coloneqq 10\ \text{W/m}^2$ being *threshold of pain* (TOP). For practical reasons, one switches to a logarithmic scale to express power & intensity. More precisely, one uses a *decibel* (dB) scale, which is a logarithmic unit expressing ratio between 2 values. Typically, 1 of values serves as a reference, e.g., I_{TOH}

in case of sound intensity. Then intensity measured in dB is defined as

$$\mathrm{dB}(I) \coloneqq 10 \log_{10} \frac{I}{I_{\mathrm{TOH}}}.$$

From this def, one obtains $\mathrm{dB}(I_{\mathrm{TOH}}) = 0$, & a doubling of intensity results in an increase of roughly 3 dB:

$$\mathrm{dB}(2I) = 10 \log_{10} 2 + \mathrm{dB}(I) \approx 3 + \mathrm{dB}(I).$$

When specifying intensity values in terms of decibels, one also speaks of *intensity levels*. Table 1.1: Typical intensity values given W/m^2 (intensity), in decibels (intensity level), & by a factor compared with TOH. shows some typical intensity values given in W/m^2 as well as in decibels for some sound sources & dynamic indicators. E.g., notes being played pianissimo ("very softly") typically result in intensity levels around 40 dB, whereas notes being played fortissimo ("very loudly") can reach levels up to 100 dB.

– Khi chỉ định các giá trị cường độ theo decibel, người ta cũng nói về *mức cường độ*. Bảng 1.1: Các giá trị cường độ điển hình được đưa ra theo W/m^2 (cường độ), tính bằng decibel (mức cường độ), & theo 1 hệ số so với TOH. cho thấy 1 số giá trị cường độ điển hình được đưa ra theo W/m^2 cũng như theo decibel đối với 1 số nguồn âm thanh & chỉ báo động. E.g., các nốt được chơi pianissimo ("rất nhẹ") thường dẫn đến mức cường độ khoảng 40 dB, trong khi các nốt được chơi fortissimo ("rất to") có thể đạt tới mức lên tới 100 dB.

Now come back to concept of *loudness*, which is perceptual correlate to sound intensity [6, 17]. As said before, loudness is affected by a number of factors. 1st of all, same sound may be perceived to have different loudness depending on individual. In particular, age is 1 factor that affects human ear's response to a sound. Also, duration of sound influences perception, since human auditory system averages effect of sound intensity over an interval up to a sec. Therefore, a human has feeling that a sound lasting for 200 ms is louder than a similar sound only lasting 50 ms. 2 sounds with same intensity but different frequencies are generally not perceived to have same loudness. Humans with normal hearing are most sensitive to sounds around 2–4 kHz, with sensitivity declining for lower as well as higher frequencies. Based on psychoacoustic experiments, perceived loudness of pure tones depending on frequency has been determined & expressed by unit *phon*. Fig. 1.21: Equal loudness contours (see [6, 17]. shows *equal loudness contours*. Each contour line specifies for a fixed loudness given in phons sound intensities over a (logarithmically spaced) frequency axis. Unit of a phon is normalized w.r.t. frequency of 1000 Hz, where a phon value equals intensity level in dB. Contour for 0 phon shows how threshold of hearing depends on frequency.

– Bây giờ quay lại khái niệm *độ to*, có mối tương quan về mặt nhận thức với cường độ âm thanh [6, 17]. Như đã nói trước đó, độ to bị ảnh hưởng bởi 1 số yếu tố. Trước hết, cùng 1 âm thanh có thể được cảm nhận có độ to khác nhau tùy thuộc vào từng cá nhân. Đặc biệt, tuổi tác là 1 yếu tố ảnh hưởng đến phản ứng của tai người đối với âm thanh. Ngoài ra, thời lượng âm thanh cũng ảnh hưởng đến nhận thức, vì hệ thống thính giác của con người tính trung bình tác động của cường độ âm thanh trong 1 khoảng thời gian lên đến 1 giây. Do đó, con người có cảm giác rằng âm thanh kéo dài trong 200 ms to hơn âm thanh tương tự chỉ kéo dài 50 ms. 2 âm thanh có cùng cường độ nhưng tần số khác nhau thường không được cảm nhận là có cùng độ to. Con người có thính giác bình thường nhạy cảm nhất với âm thanh trong khoảng 2–4 kHz, độ nhạy giảm dần đối với cả tần số thấp hơn & cao hơn. Dựa trên các thí nghiệm về tâm lý âm học, độ to được cảm nhận của âm thanh thuần túy tùy thuộc vào tần số đã được xác định & thể hiện bằng đơn vị *phon*. Hình 1.21: Đường đồng mức âm lượng bằng nhau (xem [6, 17]. hiển thị *đường đồng mức âm lượng bằng nhau*. Mỗi đường đồng mức chỉ định 1 độ lớn cố định được đưa ra theo cường độ âm thanh phon trên 1 trục tần số (cách đều theo logarit). Đơn vị của phon được chuẩn hóa theo tần số 1000 Hz, trong đó giá trị phon bằng mức cường độ tính bằng dB. Đường đồng mức cho phon 0 hiển thị ngưỡng nghe phụ thuộc vào tần số như thế nào.

∗ 1.3.4. Timbre. Besides pitch, loudness, & duration, there is another fundamental aspect of sound referred to as *timbre* or *tone color*. Timbre allows a listener to distinguish musical tone of a violin, an oboe, or a trumpet even if the tone is played at same pitch & with same loudness. As with pitch & loudness, timbre is a perceptual property of sound [22]. However, timbre is very hard to grasp, & because of its vagueness, often describe in an indirect way: timbre is attribute whereby a listener can judge 2 sounds as dissimilar using any criterion other than pitch, loudness, & duration. E.g., timbre information allows us to tell apart sounds produced by oboe & violin, even when pitch & loudness of sounds are identical [19]. Sound of a musical instrument may be described with such words as bright, dark, warm, harsh, & other terms. Researchers have tried to approach timbre by looking at correlations to more objective sound characteristics e.g. temporal & spectral evolution, absence or presence of tonal & noise-like components, or energy distribution across partials of a tone. In following, take a closer look at some of these characteristics.

– Âm sắc. Bên cạnh cao độ, độ to, & thời lượng, còn có 1 khía cạnh cơ bản khác của âm thanh được gọi là *âm sắc* hoặc *màu sắc âm thanh*. Âm sắc cho phép người nghe phân biệt âm điệu của đàn violin, ô-boa hoặc kèn trumpet ngay cả khi âm điệu được chơi ở cùng cao độ & với cùng độ to. Giống như cao độ & độ to, âm sắc là 1 đặc tính nhận thức của âm thanh [22]. Tuy nhiên, âm sắc rất khó nắm bắt, & vì tính mơ hồ của nó, thường được mô tả theo cách gián tiếp: âm sắc là thuộc tính mà người nghe có thể đánh giá 2 âm thanh là không giống nhau bằng bất kỳ tiêu chí nào khác ngoài cao độ, độ to, & thời lượng. E.g., thông tin về âm sắc cho phép chúng ta phân biệt các âm thanh do ô-boa & vĩ cầm tạo ra, ngay cả khi cao độ & độ to của các âm thanh là giống hệt nhau [19]. Âm thanh của 1 nhạc cụ có thể được mô tả bằng các từ như sáng, tối, ấm, chói tai, & các thuật ngữ khác. Các nhà nghiên cứu đã cố gắng tiếp cận âm sắc bằng cách xem xét các mối tương quan với các đặc điểm âm thanh khách quan hơn, ví dụ: sự tiến hóa theo thời gian & quang phổ, sự vắng mặt hoặc có mặt của các thành phần giống như âm & tiếng ồn, hoặc sự phân bố năng lượng trên các phần của 1 âm. Sau đây, hãy xem xét kỹ hơn 1 số đặc điểm này.

When striking a piano key, resulting sound is much more than a superposition of pure sinusoids that correspond to fundamental frequency & its overtones. Playing a single note already produces a complex sound mixture with characteristics that many constantly change over time, containing periodic as well as nonperiodic components. At beginning of a musical tone, there is often a sudden increase of energy. In this short phase, *attack phase* of tone, sound builds up. It contains a high degree of nonperiodic components that are spread over entire range of frequencies, a property that is also inherent to *noise*. In acoustics, such a noise-like short-duration sound of high amplitude occurring at beginning of a waveform is also called a *transient*. In case of a piano, striking a key triggers an entire chain of mechanical actions before a hammer hits 1 or several strings. All these actions, starting with finger touching key & ending with hammer hitting strings, produce mechanical noise that merges with acoustic effects of strings' excitation. After attack phase, sound of a musical tone stabilizes (*decay phase*) & reaches a steady phase with a (more or less) periodic pattern. This 3rd phase, also called *sustain phase*, makes up most of duration of a musical tone, where energy remains more or less constant or slightly decreases as is case with a piano sound. In final phase of a musical tone, also called *release phase*, musical tone fades away. For a piano, this phase starts as soon as finger leaves key & damper stops strings' vibrations.

– Khi nhấn phím đàn piano, âm thanh tạo ra không chỉ là sự chồng chập của các sóng sin thuần túy tương ứng với tần số cơ bản & các âm bội của nó. Chơi 1 nốt nhạc đơn lẻ đã tạo ra 1 hỗn hợp âm thanh phức tạp với các đặc điểm liên tục thay đổi theo thời gian, bao gồm các thành phần tuần hoàn cũng như không tuần hoàn. Khi bắt đầu 1 giai điệu nhạc, thường có sự gia tăng năng lượng đột ngột. Trong pha ngắn này, *pha tấn công* của giai điệu, âm thanh tích tụ. Nó chứa 1 mức độ cao các thành phần không tuần hoàn trải rộng trên toàn bộ dải tần số, 1 đặc tính cũng vốn có của *tiếng ồn*. Trong âm học, âm thanh có thời lượng ngắn giống như tiếng ồn với biên độ cao xảy ra ở đầu dạng sóng cũng được gọi là *thoáng qua*. Trong trường hợp của đàn piano, việc nhấn phím sẽ kích hoạt toàn bộ chuỗi hành động cơ học trước khi búa đập vào 1 hoặc nhiều dây đàn. Tất cả các hành động này, bắt đầu bằng việc ngón tay chạm vào phím & kết thúc bằng việc búa đập vào dây đàn, tạo ra tiếng ồn cơ học hòa quyện với các hiệu ứng âm thanh của sự kích thích dây đàn. Sau giai đoạn tấn công, âm thanh của 1 giai điệu nhạc ổn định (*giai đoạn suy yếu*) & đạt đến 1 giai đoạn ổn định với 1 mô hình (ít nhiều) tuần hoàn. Giai đoạn thứ 3 này, còn được gọi là *giai đoạn duy trì*, chiếm phần lớn thời lượng của 1 giai điệu nhạc, trong đó năng lượng vẫn ít nhiều không đổi hoặc giảm nhẹ như trường hợp của âm thanh piano. Ở giai đoạn cuối của 1 giai điệu nhạc, còn được gọi là *giai đoạn giải phóng*, giai điệu nhạc mờ dần. Đối với đàn piano, giai đoạn này bắt đầu ngay khi ngón tay rời khỏi phím & bộ giảm âm dừng rung động của dây đàn.

Intuitively, *envelope* of a waveform can be regarded to be a smooth curve outlining its extremes in amplitude (Fig. 1.22(a): Envelope of a signal.). Different phases as described above have a strong influence on shape of envelope of a musical tone. In sound synthesis, envelope of a signal to be generated is often described by a model called *ADSR*, which consists of an attack (A), decay (D), sustain (S), & release (R) phase (Fig. 1.22(b): Schematic view of an ADSR envelope.). Relative durations & amplitudes of 4 phases have a significant impact on how synthesized tone will sound.

– Theo trực giác, *envelope* của 1 dạng sóng có thể được coi là 1 đường cong trơn tru phác thảo các cực trị của nó về biên độ (Hình 1.22(a): Đường bao của tín hiệu.). Các pha khác nhau như mô tả ở trên có ảnh hưởng mạnh đến hình dạng đường bao của 1 giai điệu nhạc. Trong tổng hợp âm thanh, đường bao của tín hiệu được tạo ra thường được mô tả bằng 1 mô hình gọi là *ADSR*, bao gồm pha tấn công (A), suy giảm (D), duy trì (S), & giải phóng (R) (Hình 1.22(b): Sơ đồ dạng đường bao ADSR.). Thời lượng tương đối & biên độ của 4 pha có tác động đáng kể đến cách âm thanh tổng hợp sẽ phát ra.

ADSR model in a strong simplification & only yields a meaningful approximation for amplitude envelopes of tones that are generated by certain instruments. E.g., musical tone shown in Fig. 1.23(a): Wave form, amplitude envelope, & spectrogram representation for different instruments playing same note C4 (261.6 Hz). (a) Piano., which is note C4 played on a piano, has an envelope that is similar to one suggested by ADSR model. After a sharp attack (when hammer hits string) & a stabilizing decay, tone continuously fades out. In case of a piano sound, decrease in sound intensity is very slow as long as damper does not touch string. Therefore, one can regard this phase as a kind of sustain phase. When piano key is released & damper stops string's vibration, sound quickly comes to an end. For other instruments, however, amplitude may evolve in a completely different fashion. This is illustrated by Fig. 1.23(b): Violin., which shows an envelope for note C4 played on a violin. 1st of all, since tone is played softly with a gradual increase in volume, attack phase is spread out in time. Furthermore, there does not seem to be any decay phase & subsequent sustain phase is not steady; instead, amplitude envelope oscillates in a regular fashion. Release phase starts when violinist stops exciting string with bow. Sound then quickly fades out.

– Mô hình ADSR trong 1 sự đơn giản hóa mạnh mẽ & chỉ đưa ra 1 phép tính gần đúng có ý nghĩa cho các bao biên độ của âm thanh được tạo ra bởi 1 số nhạc cụ nhất định. E.g., âm thanh nhạc được hiển thị trong Hình 1.23(a): Dạng sóng, bao biên độ, & biểu diễn phổ cho các nhạc cụ khác nhau chơi cùng 1 nốt C4 (261,6 Hz). (a) Piano., là nốt C4 được chơi trên đàn piano, có bao tương tự như bao do mô hình ADSR gợi ý. Sau 1 cú đánh mạnh (khi búa đập vào dây đàn) & sự suy giảm ổn định, âm thanh liên tục mờ dần. Trong trường hợp âm thanh piano, cường độ âm thanh giảm rất chậm miễn là bộ giảm âm không chạm vào dây đàn. Do đó, người ta có thể coi pha này là 1 loại pha duy trì. Khi phím đàn piano được nhả & bộ giảm âm dừng rung động của dây đàn, âm thanh nhanh chóng kết thúc. Tuy nhiên, đối với các nhạc cụ khác, biên độ có thể phát triển theo 1 cách hoàn toàn khác. Điều này được minh họa bằng Hình 1.23(b): Violin., cho thấy 1 đường bao cho nốt C4 được chơi trên đàn violin. Trước hết, vì âm thanh được chơi nhẹ nhàng với âm lượng tăng dần, nên pha tấn công được trải ra theo thời gian. Hơn nữa, dường như không có pha suy giảm nào & pha duy trì tiếp theo không ổn định; thay vào đó, đường bao biên độ dao động theo cách đều đặn. Pha nhả bắt đầu khi nghệ sĩ violin ngừng kích thích dây đàn bằng vĩ. Sau đó, âm thanh nhanh chóng mờ dần.

For our violin example, one can observe periodic variations in amplitude. This phenomenon, known as *tremolo*, is generated

by certain playing styles used for string or wind instruments. Effect of tremolo often goes along with *vibrato*, which is a musical effect consisting of a regular, pulsating change of frequency. Besides string music, vibrato is mainly used by human singers to add expression. In technical terms, tremolo corresponds to an *amplitude modulation*, whereas vibrato corresponds to a *frequency modulation*. Both tremolo & vibrato depend on 2 parameters: extent of variation & rate at which amplitude or frequency is varied. Even though tremolo & vibrato are simply local changes in intensity & frequency, they do not necessarily evoke a perceived change in loudness or pitch of overall musical tone.
– Đối với ví dụ về đàn violin của chúng ta, người ta có thể quan sát thấy những biến đổi tuần hoàn về biên độ. Hiện tượng này, được gọi là *tremolo*, được tạo ra bởi 1 số phong cách chơi được sử dụng cho các nhạc cụ dây hoặc hơi. Hiệu ứng của tremolo thường đi kèm với *vibrato*, đây là 1 hiệu ứng âm nhạc bao gồm sự thay đổi tần số đều đặn, dao động. Bên cạnh nhạc cụ dây, vibrato chủ yếu được các ca sĩ sử dụng để tăng thêm biểu cảm. Về mặt kỹ thuật, tremolo tương ứng với *điều chế biên độ*, trong khi vibrato tương ứng với *điều chế tần số*. Cả tremolo & vibrato đều phụ thuộc vào 2 thông số: mức độ biến đổi & tốc độ mà biên độ hoặc tần số thay đổi. Mặc dù tremolo & vibrato chỉ đơn giản là những thay đổi cục bộ về cường độ & tần số, nhưng chúng không nhất thiết gợi lên sự thay đổi nhận thức về độ to hoặc cao độ của toàn bộ âm điệu âm nhạc.
Perhaps most important & well-known property for characterizing timbre is existence of certain partials & their relative strengths [19]. Recall from Sect. 1.3.2 that partials are dominant frequencies of a musical tone with lowest partial being fundamental frequency. Inharmonicity expresses extent to which a partial deviates from closest ideal harmonic. For harmonic sounds e.g. a musical tone with a clearly perceivable pitch, most of partials are close to being harmonics. However, not all partials need to occur with same strength.
– Có lẽ tính chất quan trọng nhất & nổi tiếng để mô tả âm sắc là sự tồn tại của 1 số phần & cường độ tương đối của chúng [19]. Nhớ lại từ Mục 1.3.2 rằng các phần là tần số chủ đạo của 1 âm nhạc với phần thấp nhất là tần số cơ bản. Sự bất hòa âm thể hiện mức độ mà 1 phần lệch khỏi hài hòa lý tưởng gần nhất. Đối với âm thanh hài hòa, ví dụ như 1 âm nhạc có cao độ có thể nhận biết rõ ràng, hầu hết các phần đều gần với hài hòa. Tuy nhiên, không phải tất cả các phần đều cần phải xảy ra với cùng 1 cường độ.
Composition of a sound in terms of its partials can be visualized by a so-called *spectrogram*, which shows intensity of occurring frequencies over time. For a detailed introduction on such time-frequency representations refer to Sect. 2.5. Fig. 1.23(a) shows at bottom a spectrogram for note C4 played on a piano, where intensity is reflected by shade of gray (darker more intense). Both fundamental frequency of note (261.6 Hz) as well as its harmonics (integer multiples of 261.6 Hz) are visible as horizontal lines. Decay of musical tone is reflected by a corresponding decay in each of partials. Most of tone's energy is contained in lower partials, & energy tends to lower for higher partials. Such a distribution is typical for many instruments.
– Thành phần của âm thanh theo từng phần của nó có thể được hình dung bằng cái gọi là *spectrogram*, cho thấy cường độ của các tần số xảy ra theo thời gian. Để biết phần giới thiệu chi tiết về các biểu diễn tần số thời gian như vậy, hãy tham khảo Mục 2.5. Hình 1.23(a) cho thấy ở dưới cùng là 1 phổ đồ cho nốt C4 được chơi trên đàn piano, trong đó cường độ được phản ánh bằng sắc thái xám (tối hơn cường độ cao hơn). Cả tần số cơ bản của nốt (261,6 Hz) cũng như các sóng hài của nó (bội số nguyên của 261,6 Hz) đều có thể nhìn thấy dưới dạng các đường ngang. Sự suy giảm của âm sắc được phản ánh bằng sự suy giảm tương ứng trong mỗi phần. Hầu hết năng lượng của âm thanh nằm trong các phần thấp hơn, & năng lượng có xu hướng thấp hơn đối với các phần cao hơn. Sự phân bố như vậy là điển hình cho nhiều nhạc cụ.
For string instruments, sounds tend to have a rich spectrum of partials, where lots of energy may also be contained in upper harmonics (Fig. 1.23(b)). This figure also reveals vibrato as a regular oscillation in time-frequency plane. Certain classes of wind instruments including clarinet (so-called *closed-pipe* wind instruments) produce a very characteristic spectrum of partials. For a cylindrical wind instrument that is open at 1 end, but closed at the other (at mouthpiece), one can show: even harmonics do not show up. I.e., most energy is contained in odd harmonics $\omega_0, 3\omega_0, 5\omega_0, \ldots$, with ω_0 denoting fundamental frequency. For a musical tone played on a bassoon, fundamental frequency often contains much less energy compared with higher partials. In contrast, for a tuning fork, most energy is contained in fundamental frequency, resulting in a sound that is close to a synthesized sinusoid. Instruments e.g. bells have a very complex spectrum with lots of inharmonicities, which often evokes in listener feeling of a bell being out-of-tune. For stringed instruments, one can often measure substantial deviations between higher partials & theoretical harmonics. Less elastic a string is (i.e., shorter, thicker, higher tension or stiffer it is), more inharmonicity it may exhibit. This particularly holds for piano, where such inharmonicities have a crucial influence on timbre.
– Đối với các nhạc cụ dây, âm thanh có xu hướng có phổ các phần phong phú, trong đó nhiều năng lượng cũng có thể được chứa trong các hài bậc cao (Hình 1.23(b)). Hình này cũng cho thấy rung âm là dao động đều trong mặt phẳng thời gian-tần số. 1 số loại nhạc cụ hơi bao gồm kèn clarinet (còn gọi là nhạc cụ hơi dạng ống kín) tạo ra phổ các phần rất đặc trưng. Đối với 1 nhạc cụ hơi hình trụ hở ở 1 đầu nhưng đóng ở đầu kia (ở miệng thổi), người ta có thể thấy: các hài bậc chẵn không xuất hiện. Tức là, hầu hết năng lượng được chứa trong các hài bậc lẻ $\omega_0, 3\omega_0, 5\omega_0, \ldots$, với ω_0 biểu thị tần số cơ bản. Đối với 1 âm nhạc được chơi trên kèn bassoon, tần số cơ bản thường chứa ít năng lượng hơn nhiều so với các phần cao hơn. Ngược lại, đối với 1 âm thoa, hầu hết năng lượng được chứa trong tần số cơ bản, tạo ra âm thanh gần với âm sin tổng hợp. Các nhạc cụ như chuông có phổ rất phức tạp với nhiều bất hòa, thường gợi lên trong người nghe cảm giác như chuông bị lệch tông. Đối với nhạc cụ dây, người ta thường có thể đo được độ lệch đáng kể giữa các phần cao hơn & các hài âm lý thuyết. Dây đàn càng ít đàn hồi (tức là ngắn hơn, dày hơn, căng hơn hoặc cứng hơn), thì nó có thể biểu hiện bất hòa nhiều hơn. Điều này đặc biệt đúng với đàn piano, nơi mà những bất hòa như vậy có ảnh hưởng quan trọng đến âm sắc.
With this discussion, want to indicate that timbre is a multidimensional phenomenon that is hard to measure. It is irregularities & variations that make a musical tone sound interesting & that give it a particular & natural quality.

– Với cuộc thảo luận này, muốn chỉ ra rằng âm sắc là 1 hiện tượng đa chiều khó đo lường. Chính sự bất thường & biến thể làm cho 1 giai điệu âm nhạc nghe thú vị & mang lại cho nó 1 & chất lượng tự nhiên đặc biệt.

- 1.4. Summary & Further Readings. In this chap, looked at 3 different classes of music representations while introducing some musical & technical terminology used throughout this book. Use term *sheet music* to refer to visual representations of a musical score either given in printed form or encoded digitally in some image format. Term *symbolic* stands for any kind of symbolic representation where entities have an explicit musical meaning. Finally, term *audio* is used to denote music recordings given in form of acoustic waveforms. Boundaries between these classes are not clear. In particular, as illustrated by Fig. 1.24: Illustration of 3 classes of music representation & their relations., symbolic representations may be close to both sheet music as well as audio representations [24]. On 1 hand, symbolic representations e.g. MusicXML are used for *rendering* sheet music, where shape of note objects & their arrangement on a page are determined. As seen, optical music recognition (OMR) is inverse process with goal of transforming sheet music into a symbolic representation. On other hand, symbolic representations e.g. MIDI are used for *synthesizing* audio, where note objects are transformed into musical tones & real sounds. Inverse process is known as *music transcription*, where objective: extract note events, key signature, time signature, instrumentation, & other score parameters from a given music recording [2, 13].

  – Trong chương này, chúng ta sẽ xem xét 3 lớp biểu diễn âm nhạc khác nhau trong khi giới thiệu 1 số thuật ngữ kỹ thuật & âm nhạc được sử dụng trong toàn bộ cuốn sách này. Sử dụng thuật ngữ *bản nhạc* để chỉ các biểu diễn trực quan của 1 bản nhạc được đưa ra dưới dạng in hoặc được mã hóa kỹ thuật số ở 1 số định dạng hình ảnh. Thuật ngữ *biểu tượng* là bất kỳ loại biểu diễn biểu tượng nào trong đó các thực thể có ý nghĩa âm nhạc rõ ràng. Cuối cùng, thuật ngữ *âm thanh* được sử dụng để biểu thị các bản ghi âm nhạc được đưa ra dưới dạng sóng âm. Ranh giới giữa các lớp này không rõ ràng. Đặc biệt, như được minh họa bởi Hình 1.24: Minh họa về 3 lớp biểu diễn âm nhạc & mối quan hệ của chúng., các biểu diễn biểu tượng có thể gần với cả bản nhạc cũng như biểu diễn âm thanh [24]. Mặt khác, các biểu diễn biểu tượng, ví dụ như MusicXML được sử dụng để *kết xuất* bản nhạc, trong đó hình dạng của các đối tượng nốt nhạc & cách sắp xếp của chúng trên 1 trang được xác định. Như đã thấy, nhận dạng âm nhạc quang học (OMR) là quá trình ngược lại với mục tiêu chuyển đổi bản nhạc thành biểu diễn biểu tượng. Mặt khác, các biểu diễn tượng trưng ví dụ như MIDI được sử dụng để *tổng hợp* âm thanh, trong đó các đối tượng nốt nhạc được chuyển đổi thành âm điệu nhạc & âm thanh thực. Quá trình ngược lại được gọi là *phiên âm nhạc*, trong đó mục tiêu: trích xuất các sự kiện nốt nhạc, chữ ký khóa, chữ ký nhịp, nhạc cụ, & các tham số bản nhạc khác từ 1 bản ghi âm nhạc nhất định [2, 13].

  In a sense, symbolic representations can be regarded as link between visual (or graphical) domain accommodating sheet music representations & acoustic (or physical) domain accommodating audio representations [24]. In 1st case, timing is specified in terms of shape & relative arrangement of musical symbols & is typically given in musical units e.g. measures or beats. In latter case, timing is specified in physical units e.g. secs. For music recordings, there are often no sharp note onsets or offsets (think of a soft onset for a note played on a violin or a gradual fade-out) & specification of beginning & end of musical events becomes an ill-defined problem. For a general discussion of alignment procedures to bridge gap between sheet music & audio representations, refer to [24].

  – Theo 1 nghĩa nào đó, các biểu diễn tượng trưng có thể được coi là mối liên kết giữa miền thị giác (hoặc đồ họa) chứa các biểu diễn bản nhạc & miền âm thanh (hoặc vật lý) chứa các biểu diễn âm thanh [24]. Trong trường hợp thứ nhất, thời gian được chỉ định theo hình dạng & sự sắp xếp tương đối của các ký hiệu âm nhạc & thường được đưa ra theo các đơn vị âm nhạc, ví dụ như ô nhịp hoặc phách. Trong trường hợp sau, thời gian được chỉ định theo các đơn vị vật lý, ví dụ như giây. Đối với các bản ghi âm nhạc, thường không có điểm bắt đầu hoặc kết thúc nốt sắc nét (hãy nghĩ đến điểm bắt đầu nhẹ nhàng cho 1 nốt nhạc được chơi trên đàn vĩ cầm hoặc sự mờ dần dần) & việc chỉ định điểm bắt đầu & kết thúc của các sự kiện âm nhạc trở thành 1 vấn đề không được xác định rõ ràng. Để biết thảo luận chung về các quy trình căn chỉnh để thu hẹp khoảng cách giữa các biểu diễn bản nhạc & âm thanh, hãy tham khảo [24].

  Of course, any kind of categorization of music representations goes along with an oversimplification. Our categorization is far from being comprehensive. Have seen: when describing musical attributes e.g. pitch, loudness, & timbre, human perception is a crucial factor. Therefore, besides acoustic & visual domain, Babbitt [1] considers an additional *auditory* domain. In his taxonomy, a graphemic note (blob on page) corresponds in meaning with (auditory) percept of note. From a philosophical point of view, as argued by Wiggins et al. [25], music is actually sth abstract & intangible which does not have real existence in itself. In this sense, all of domain-specific representations are *aspects* of music, but none of them *is* music, individually. Mazzola [15] considers music to be universe of all different perspectives one may assume. For a psychologically based approach to music along with expectations & emotions it evokes, refer to book by Huron [12]. Running risk of oversimplification, adopt in this book a more technically oriented view of music processing & leave out perhaps most important aspect of music: human mind.

  – Tất nhiên, bất kỳ loại phân loại nào về biểu diễn âm nhạc đều đi kèm với sự đơn giản hóa quá mức. Phân loại của chúng tôi còn lâu mới toàn diện. Đã thấy: khi mô tả các thuộc tính âm nhạc ví dụ như cao độ, độ to, & âm sắc, nhận thức của con người là 1 yếu tố quan trọng. Do đó, bên cạnh phạm vi âm thanh & thị giác, Babbitt [1] xem xét 1 phạm vi *thính giác* bổ sung. Trong phân loại của mình, 1 nốt nhạc đồ họa (đốm trên trang) tương ứng về mặt ý nghĩa với nhận thức (thính giác) về nốt nhạc. Theo quan điểm triết học, như Wiggins & cộng sự [25] lập luận, âm nhạc thực sự là thứ gì đó trừu tượng & vô hình, không tồn tại thực sự trong chính nó. Theo nghĩa này, tất cả các biểu diễn cụ thể theo phạm vi đều là *khía cạnh* của âm nhạc, nhưng không có biểu diễn nào trong số chúng *là* âm nhạc, riêng lẻ. Mazzola [15] coi âm nhạc là vũ trụ của tất cả các quan điểm khác nhau mà người ta có thể cho là. Để biết cách tiếp cận dựa trên tâm lý đối với âm nhạc cùng với những kỳ vọng & cảm xúc mà nó gợi lên, hãy tham khảo cuốn sách của Huron [12]. Có nguy cơ đơn giản hóa quá mức, cuốn sách này áp dụng quan điểm thiên về kỹ thuật hơn về xử lý âm nhạc & bỏ qua khía cạnh có lẽ là quan trọng nhất của âm nhạc: tâm trí con người.

Sheet music has a history of hundreds of years, & basic concepts presented can be found in introductory textbooks on music notation [9], & also refer to Wikipedia as a rich source of useful information on this topic. Because of significant digitization efforts, sheet music is now widely available in digital formats. In particular for Western classical music, scanned versions of musical editions out of copyright are now freely accessible on world wide web. 1 prominent example: *Petrucci Music Library*, which is a virtual library of public-domain music scores organized & created by *International Music Score Library Project* (IMSLP) http://imslp.org. For symbolic music, many formats have been suggested in literature to represent sheet music in a digital, machine-readable form. A comprehensive account on MIDI http://www.midi.org & its use with electronic instruments & sequencers can be found in [11]. Extensions & challenges of MIDI format are summarized in [14]. In book edited by Selfridge-Field [21], one not only finds an introduction to MIDI format but also a detailed overview & description of symbolic formats up to year 1997. Since then, many new formats have been proposed & developed, including both open & well-documented formats, as well as proprietary formats that are bound to specific software packages. Music XML http://www.musicxml.com format [8] & MEI https://music-encoding.org/ format developed by community-driven Music Encoding Initiative [10], are only 2 prominent examples. Similarly, a multitude of commercial & noncommercial OMR software systems have been developed. While many of these systems only work for printed sheet music, others also address much harder problem of recognizing handwritten scores. In recent decades, significant research efforts have been directed towards improving, comparing, & evaluating OMR systems [3]. Even though substantial improvements could be achieved, also thanks to recent data-driven techniques based on DL, OMR can still not be regarded as a solved problem. For a comprehensive overview of OMR literature, refer to *Bibliography on Optical Music Recognition* https://omr-research.github.io/.

–Bản nhạc có lịch sử hàng trăm năm, & các khái niệm cơ bản được trình bày có thể được tìm thấy trong các sách giáo khoa nhập môn về ký hiệu âm nhạc [9], & cũng tham khảo Wikipedia như 1 nguồn thông tin hữu ích phong phú về chủ đề này. Do những nỗ lực số hóa đáng kể, bản nhạc hiện có sẵn rộng rãi ở các định dạng kỹ thuật số. Đặc biệt đối với nhạc cổ điển phương Tây, các phiên bản được quét của các ấn bản nhạc không có bản quyền hiện có thể truy cập miễn phí trên web toàn cầu. 1 ví dụ nổi bật: *Petrucci Music Library*, là 1 thư viện ảo chứa các bản nhạc thuộc phạm vi công cộng được tổ chức & tạo ra bởi *International Music Score Library Project* (IMSLP) http://imslp.org. Đối với nhạc tượng trưng, nhiều định dạng đã được đề xuất trong tài liệu để thể hiện bản nhạc ở dạng kỹ thuật số, có thể đọc bằng máy. 1 tài khoản toàn diện về MIDI http://www.midi.org & việc sử dụng nó với các nhạc cụ điện tử & trình sắp xếp có thể được tìm thấy trong [11]. Các phần mở rộng & thách thức của định dạng MIDI được tóm tắt trong [14]. Trong cuốn sách do Selfridge-Field biên tập [21], người ta không chỉ tìm thấy phần giới thiệu về định dạng MIDI mà còn có phần tổng quan & mô tả chi tiết về các định dạng ký hiệu cho đến năm 1997. Kể từ đó, nhiều định dạng mới đã được đề xuất & phát triển, bao gồm cả các định dạng mở & được ghi chép đầy đủ, cũng như các định dạng độc quyền được liên kết với các gói phần mềm cụ thể. Định dạng Music XML http://www.musicxml.com [8] & MEI https://music-encoding.org/ do Sáng kiến Mã hóa Âm nhạc do cộng đồng điều hành [10] phát triển, chỉ là 2 ví dụ nổi bật. Tương tự như vậy, vô số hệ thống phần mềm OMR thương mại & phi thương mại đã được phát triển. Trong khi nhiều hệ thống trong số này chỉ hoạt động đối với bản nhạc in, thì những hệ thống khác cũng giải quyết vấn đề khó khăn hơn nhiều là nhận dạng bản nhạc viết tay. Trong những thập kỷ gần đây, những nỗ lực nghiên cứu đáng kể đã được hướng tới việc cải thiện, so sánh, & đánh giá các hệ thống OMR [3]. Mặc dù có thể đạt được những cải tiến đáng kể, cũng nhờ vào các kỹ thuật dựa trên dữ liệu gần đây dựa trên DL, OMR vẫn không thể được coi là 1 vấn đề đã được giải quyết. Để có cái nhìn tổng quan toàn diện về tài liệu OMR, hãy tham khảo *Tài liệu tham khảo về Nhận dạng âm nhạc quang học* https://omr-research.github.io/.

There are many excellent books on foundations of acoustical properties of music & audio signals. E.g., classic book by Fletcher & Rossing [7] gives a detailed account on musical sound waves & physics behind their generation by musical instruments. Book by Fastl and Zwicker [5] as well as the one by Moore [17] give deeper insights into field of auditory perception & psycho-acoustics for general audio signals. A source of inspiration for this chap has been book by Sethares [22] on tuning, timbre, spectrum, & scale, which provides interesting insights (along with sound examples) on how these concepts are related. A signal-processing-oriented approach to concepts of timbre & instrumentation can be found in [19].

DL techniques have opened up new avenues for various tasks related to processing, converting, & linking music representations. E.g., this holds for task of OMR when a sufficient amount of well-annotated training data is available [3]. Similarly, major progress could be achieved in music transcription using DL techniques [2]. Data-driven techniques are also increasingly used for cross-modal retrieval & alignment tasks [4, 18]. However, music turns out to be a hard domain due to complexity & diversity of music, which would require vast amounts of data to efficiently cover all these aspects. E.g., OMR is still a hard problem for handwritten music or sheet music with a dense & complex layout. Similarly, while automated methods for music transcription work well for piano recordings of high acoustic quality (where one has a lot of training data), automatic conversion of complex orchestral or choir performances into score notation – a task Mozart was capable of after listening to a polyphonic choral piece only once – is still a largely open problem despite decades of research.

- 1.5. FMP Notebooks. In this chap, seen: musical information can be represented in many different ways, including sheet music, symbolic, & audio representations. In Part 1 of FMP notebooks [20], which is closely associated with this 1st chap, offer visual & acoustic material + Python code examples to study musical & acoustic properties of music. Now briefly go through FMP notebooks of Part 1 1 by 1 while indicating how these can be used for possible experiments & exercises.

  Start with [MAP ON FMP NOTEBOOKS] p. 32+++35

  In summary, in FMP notebooks of Part 1, provide basic Python code examples for parsing & visualizing various music representations. Furthermore, consider tangible music examples & suggest various experiments for deepening understanding of musical & acoustic properties of audio signals including aspects e.g. frequency, pitch, dynamics, & timbre. At same time, material is also intended for developing Python programming skills are required in subsequent FMP notebooks.

- 2. Fourier Analysis of Signals. Music signals are generally complex sound mixtures that consist of a multitude of different sound components. Because of this complexity, extraction of musically relevant information from a waveform constitutes a difficult problem. A 1st step in better understanding a given signal: decompose it into building blocks that are more accessible for subsequent processing steps. In case that these building blocks consist of sinusoidal functions, such a process is also called *Fourier analysis*. Sinusoidal functions are special in sense: they possess an explicit physical meaning in terms of frequency. As a consequence, resulting decomposition unfolds frequency spectrum of signal – similar to a prism that can be used to break light up into its constituent spectral colors. Fourier transform converts a signal that depends on time into a representation that depends on frequency. Being 1 of most important tools in signal processing, encounter Fourier transform in a variety of music processing tasks.

  – Phân tích Fourier của tín hiệu. Tín hiệu âm nhạc thường là hỗn hợp âm thanh phức tạp bao gồm nhiều thành phần âm thanh khác nhau. Do tính phức tạp này, việc trích xuất thông tin có liên quan đến âm nhạc từ dạng sóng là 1 vấn đề khó khăn. Bước đầu tiên để hiểu rõ hơn về 1 tín hiệu nhất định: phân tích nó thành các khối xây dựng dễ tiếp cận hơn cho các bước xử lý tiếp theo. Trong trường hợp các khối xây dựng này bao gồm các hàm sin, quá trình như vậy cũng được gọi là *Phân tích Fourier*. Các hàm sin có ý nghĩa đặc biệt: chúng sở hữu 1 ý nghĩa vật lý rõ ràng về mặt tần số. Do đó, quá trình phân tích kết quả mở rộng phổ tần số của tín hiệu – tương tự như lăng kính có thể được sử dụng để phân tách ánh sáng thành các màu quang phổ thành phần của nó. Biến đổi Fourier chuyển đổi tín hiệu phụ thuộc vào thời gian thành biểu diễn phụ thuộc vào tần số. Là 1 trong những công cụ quan trọng nhất trong xử lý tín hiệu, hãy gặp biến đổi Fourier trong nhiều tác vụ xử lý âm nhạc.

  In Sect. 2.1, introduce main ideas of Fourier transform & summarize most important facts needed for understanding subsequent chaps of book. Furthermore, introduce required mathematical notations. A good understanding of Sect. 2.1 is essential for various music processing tasks to be discussed. In Sects. 2.2–2.5, cover Fourier transform in greater mathematical depth. Reader who is mainly interested in music processing applications may skip these more technical sects on a 1st reading.

  In Sect. 2.2, take a closer look at signals & discuss their properties from a more abstract perspective. In particular, consider 2 classes of signals: analog signals that give us right physical interpretation & digital signals needed for actual digital processing by computers. Different signal classes lead to different versions of Fourier transform, which introduce with mathematical rigor along with intuitive explanations & numerous illustrating examples (Sect. 2.3). In particular, explain how different versions are interrelated & how they can be approximated by means of discrete Fourier transform (DFT). DFT can be computed efficiently by means of fast Fourier transform (FFT), discussed in Sect. 2.4. Finally, introduce short-time Fourier transform (STFT), which is a local variant of Fourier transform yielding a time-frequency representation of a signal (Sect. 2.5). By presenting this material from a different perspective as typically encountered in an engineering course, hope to refine & sharpen understanding of these important & beautiful concepts.

  - 2.1. Fourier Transform in a Nutshell. Start with an audio signal that represents sound off some music. E.g., analyze sound of a single note played on a piano (Fig. 2.1(a): Waveform of a note C4 (261.6 Hz) played on a piano.). How can find out which note as actually been played? Recall from Sect. 1.3.2: pitch of a musical tone is closely related to its fundamental frequency, frequency of lowest partial of sound. Therefore, need to determine frequency content, main periodic oscillations of signal. Zoom into signal considering only a 10-ms sect (Fig. 2.1(b): Zoom into a 10-ms sect starting at time position $t = 1$ s.) Figure shows: signal behaves in a nearly periodic way within this sect. In particular, one can observe 3 main crests of a sinusoidal-like oscillation (see Fig. 2.1(c): Comparison of waveform with sinusoids of various frequencies ω.) Having approximately 3 oscillation cycles within a 10-ms sect means: signal contains a frequency component of roughly 300 Hz.

    Main idea of *Fourier analysis*: compare signal with sinusoids of various [In following, also consider *negative frequencies* for mathematical reasons without explaining this concept in more detail. In our musical context, negative frequencies are redundant (having same interpretation as positive frequencies), but simplify mathematical formulation of Fourier transform.] frequencies $\omega \in \mathbb{R}$ (measured in Hz). Each such sinusoid or pure tone may be thought as a prototype oscillation. As a result, obtain for each considered frequency parameter $\omega \in \mathbb{R}$ a magnitude coefficient $d_\omega \in \mathbb{R}_{\geq 0}$ (along with a phase coefficient $\varphi_\Omega \in \mathbb{R}$, role of which is explained later). In case coefficient d_ω is large, there is a high similarity between signal & sinusoid of frequency ω, & signal contains a periodic oscillation at that frequency Fig. 2.1c. In case d_ω is small, signal does not contain a periodic component at that frequency Fig. 2.1(d).

    Plot coefficients d_ω over various frequency parameters $\omega \in \mathbb{R}$. This yields a graph as shown in Fig. 2.1(f): Magnitude coefficient d_ω in dependence on frequency ω. In this graph, highest value is assumed for frequency parameter $\omega = 262$ Hz. By (1.1), this is roughly center frequency of pitch $p = 60$ or note C4. Instead, this is exactly note played in our piano example. Furthermore, as illustrated in Fig. 2.1(e), one can also observe high similarity between signal & sinusoid of frequency $\omega = 523$ Hz. This is roughly frequency for 2nd partial of tone C4.

    With this example, already seen main idea behind Fourier transform. Fourier transform breaks up a signal into its frequency components. For each frequency $\omega \in \mathbb{R}$, Fourier transforms yields a coefficient d_ω (& a phase φ_ω) that tells us to which extent given signal matches a sinusoidal prototype oscillation of that frequency.

    1 important property of Fourier transform: original signal can be reconstructed from coefficients d_ω (along with coefficients φ_ω). This weighted superposition is also called *Fourier representation* of original signal. Original signal & Fourier transform contain same amount of information. This information, however, is represented in different ways. While signal displays information across time, Fourier transform displays information across frequency. As put by Hubbard [9], signal tells us when certain notes are played in time, but hides information about frequencies. In contrast, Fourier transform of music displays which notes (frequencies) are played, but hides information about when notes are played.

    In following sects, take a more detailed look at Fourier transform & some of its main properties.

* 2.1.1. Fourier Transform for Analog Signals. In Sect. 1.3.1, saw: a signal or sound wave yields a function that assigns to each point in time deviation of air pressure from average air pressure at a specific location. Consider case of an *analog* signal, where both time as well as amplitude (or deviation) are continuous, real-valued parameters. In this case, a signal can be modeled as a function $f : \mathbb{R} \to \mathbb{R}$, which assigns to each time point $t \in \mathbb{R}$ an amplitude value $f(t) \in \mathbb{R}$. Plotting amplitude over time, one obtains a graph of this function that corresponds to waveform of signal Fig. 1.17.
Term *function* may need some explanation. In mathematics, a function yields a relation between a set of input elements & a set of output elements, where each input element is related to exactly 1 output element. E.g., a function can be a polynomial $f : \mathbb{R} \to \mathbb{R}$ that assigns for each input element $t \in \mathbb{R}$ an output element $f(t) = t^2 \in \mathbb{R}$. Emphasize: one needs to differentiate between a function f & its output element $f(t)$ (also referred to as *value*) at a particular input element t (also referred to as *argument*). I.e., mathematicians think of a function f in an abstract way, where symbol or physical meaning of argument does not matter. As opposed to this, engineers often like to emphasize meaning of input argument & loosely speak of a function $f(t)$, even though this is strictly speaking an output value. In this book, assume viewpoint of a mathematician.
  - 2.1.1.1. Role of Phase. As this side note, turn towards spectral analysis of a given analog signal $f : \mathbb{R} \to \mathbb{R}$. As explained in our introductory example, compare signal f with prototype oscillations that are given in form of sinusoids. In Sect. 1.3.2 & Fig. 1.19, have already encountered such sinusoidal signals. Mathematical, a *sinusoid* is a function $g : \mathbb{R} \to \mathbb{R}$ defined by (2.1)
$$g(t) \coloneqq A \sin(2\pi(\omega t - \varphi)) \text{ for } t \in \mathbb{R}.$$
Parameter A corresponds to *amplitude*, parameter ω to *frequency* (measured in Hz), & parameter φ to *phase* (measured in normalized radians with 1 corresponding to an angle of 360°). In Fourier analysis, consider prototype oscillations that are normalized with regard to their power (average energy) by setting $A = \sqrt{2}$. Thus for each frequency parameter ω & phase parameter φ, obtain a sinusoid $\cos_{\omega,\varphi} : \mathbb{R} \to \mathbb{R}$ given by (2.2)
$$\cos_{\omega,\varphi}(t) \coloneqq \sqrt{2} \cos(2\pi(\omega t - \varphi)) \text{ for } t \in \mathbb{R}.$$
Since cosine function is periodic, parameters $\varphi, \varphi + k$ for integers $k \in \mathbb{Z}$ yield same function. Therefore, phase parameter only needs to be considered for $\varphi \in [0, 1)$.
When measuring how well given signal coincides with a sinusoid of frequency ω, have freedom of shifting sinusoid in time. This degree of freedom is expressed by phase parameter φ. As illustrated by Fig. 2.2: (a-d) Waveform & different sinusoids of a fixed frequency $\omega = 262$ Hz but different phases $\varphi \in \{0.05, 0.24, 0.45, 0.6\}$. (e) Values that express degree of similarity between waveform & 4 different sinusoids, degree of similarity between signal & sinusoid of fixed frequency crucially depends on phase. What have we done with phase when computing coefficient d_ω illustrated by Fig. 2.1? Procedure outlined in introduction was only half story. When comparing signal f with a sinusoid $\cos_{\omega,\varphi}$ of frequency ω, have explicitly used phase φ_ω that yields maximal possible similarity. To understand this better, 1st need to explain how actually compare signal & a sinusoid or, more generally, how we compare 2 given functions.
  - 2.1.1.2. Computing Similarity with Integrals. Assume given 2 functions of time $f : \mathbb{R} \to \mathbb{R}, g : \mathbb{R} \to \mathbb{R}$. What does it mean for f, g to be similar? Intuitively, one may agree f, g are similar if they show a similar behavior over time: if f assumes positive values, then so should g, & if f becomes negative, the same should happen to g. Joint behavior of these functions can be captured by forming integral of product of 2 functions: (2.3) $\int_{\mathbb{R}} f(t)g(t)\,\mathrm{d}t$. Integral measures area delimited by graph of product fg, where negative area (below horizontal axis) is subtracted from positive area (above horizontal axis) Fig. 2.3: Measuring similarity of 2 functions f (top) & g (middle) by computing integral of product (bottom). (a) 2 functions having high similarity. (b) 2 functions having low similarity. In case f, g are either both positive or both negative at most time instances, product is positive for most of time & integral becomes large Fig. 2.3a. However, if 2 functions are dissimilar, then overall positive & overall negative areas cancel out, yield a small overall integral Fig. 2.3b.
There are many more ways for comparing 2 given signals. E.g., integral of absolute difference between functions also yields a notion of how similar signals are. In formulation of Fourier transform, however, one encounters measure as considered in (2.3), which generalizes *inner product* known from linear algebra (2.37). Continue this discussion in Sect. 2.2.3.
  - 2.1.1.3. 1st Definition of Fourier Transform. Based on similarity measure (2.3), compare original signal f with sinusoids $g = \cos_{\omega,\varphi}$ as defined in (2.2). For a fixed frequency $\omega \in \mathbb{R}$, define
$$d_\omega \coloneqq \max_{\varphi \in [0,1)} \int_{\mathbb{R}} f(t) \cos_{\omega,\varphi}(t)\,\mathrm{d}t,$$
$$\varphi_\omega \coloneqq \operatorname{argmax}_{\varphi \in [0,1)} \int_{\mathbb{R}} f(t) \cos_{\omega,\varphi}(t)\,\mathrm{d}t,$$
As prev discussed, magnitude coefficient d_ω expresses intensity of frequency ω within signal f. Additionally, phase coefficient $\varphi_\omega \in [0, 1)$ tells us how sinusoid of frequency ω needs to be displaced in time to best fit signal f. *Fourier transform* of a function $f : \mathbb{R} \to \mathbb{R}$ is defined to be "collection" of all coefficients d_ω, φ_ω for $\omega \in \mathbb{R}$.
Computation of d_ω, φ_ω feels a bit awkward, since it involves an optimization step. Good news: there is a simple solution to this optimization, which results from existence of certain trigonometric identities that relate phases & amplitudes of certain sinusoidal functions. Using concept of complex numbers, these trigonometric identities become simple & lead

to an elegant formulation of Fourier transform. Discuss such issues in more detail in Sect. 2.3. In following, introduce standard complex-valued formulation of Fourier transform without giving any proofs.
– Việc tính toán d_ω, φ_ω có vẻ hơi khó hiểu, vì nó liên quan đến 1 bước tối ưu hóa. Tin tốt: có 1 giải pháp đơn giản cho việc tối ưu hóa này, xuất phát từ sự tồn tại của 1 số hằng đẳng thức lượng giác liên quan đến pha & biên độ của 1 số hàm sin. Sử dụng khái niệm về số phức, các hằng đẳng thức lượng giác này trở nên đơn giản & dẫn đến 1 công thức thanh lịch của phép biến đổi Fourier. Thảo luận về các vấn đề như vậy chi tiết hơn trong Phần 2.3. Sau đây, giới thiệu công thức chuẩn có giá trị phức của phép biến đổi Fourier mà không đưa ra bất kỳ bằng chứng nào.

· 2.1.1.4. Complex Numbers. 1st review concept of complex numbers. Complex number extend real numbers by introducing imaginary number $i \coloneqq \sqrt{-1}$ with property $i^2 = -1$. Each complex number can be written as $c = a + bi$, where $a \in \mathbb{R}$: real part & $b \in \mathbb{R}$: imaginary part of c. Set of all complex numbers is written as $\mathbb{C}$, which can be thought of as a 2D plane: horizontal dimension corresponds to real part, & vertical dimension to imaginary part. In this plane, number $c = a + ib$ is specified by Cartesian coordinates (a, b). As illustrated by Fig. 2.4a: Polar coordinate representation of a complex number $c = a + bi$., there is another way of representing a complex number, which is known as *polar coordinate* representation. In this case, a complex number c is described by its absolute value $|c|$ (distance from origin) & angle γ between positive horizontal axis & line from origin & c. Polar coordinates $|c| \in \mathbb{R}_{\geq 0}, \gamma \in [0, 2\pi)$ (given in radians) can be derived from coordinates (a, b) via following formulas:

$$|c| \coloneqq \sqrt{a^2 + b^2},\ \gamma \coloneqq \operatorname{atan2}(b, a).$$

Further details on polar coordinates & function atan2, which is a variant of inverse of tangent function, are explained in Sect. 2.3.2.2. To regain complex number c from its polar coordinates, one uses *exponential function*, which maps an angle $\gamma \in \mathbb{R}$ (given in radians) to a complex number defined by

$$\exp(i\gamma) \coloneqq \cos\gamma + i \sin\gamma,$$

see Fig. 2.4b: Def of exponential function. Values of this function turn around unit circle of complex plane with a period of 2π (see Sect. 2.3.2.1). From this, obtain following *polar coordinate representation* for a complex number $c = |c| \exp(i\gamma)$.

· 2.1.1.5. Complex Definition of Fourier Transform. What have gained by bringing complex numbers into play? Recall: have obtained a positive coefficient $d_\omega \in \mathbb{R}_{\geq 0}$ from (2.4) & a phase coefficient $\varphi_\omega \in [0, 1)$ from (2.5). Basic idea: use these coefficients as polar coordinates & encode both coefficients by a single complex number. Because of some technical reasons (a normalization issue that becomes clearer when discussing mathematical details), one introduces some additional factors & a sign in phase to yield complex coefficient (2.10)

$$c_\omega \coloneqq \frac{d_\omega}{2} \exp(2\pi i(-\varphi_\omega)).$$

– Đã đạt được gì khi đưa số phức vào chơi? Nhớ lại: đã thu được hệ số dương $d_\omega \in \mathbb{R}_{\geq 0}$ từ (2.4) & hệ số pha $\varphi_\omega \in [0, 1)$ từ (2.5). Ý tưởng cơ bản: sử dụng các hệ số này làm tọa độ cực & mã hóa cả hai hệ số bằng 1 số phức duy nhất. Do 1 số lý do kỹ thuật (một vấn đề chuẩn hóa trở nên rõ ràng hơn khi thảo luận về các chi tiết toán học), người ta đưa vào 1 số yếu tố bổ sung & 1 dấu trong pha để tạo ra hệ số phức (2.10)
This complex formulation directly leads us to Fourier transform of a real-valued function $f : \mathbb{R} \to \mathbb{R}$. For each frequency $\omega \in \mathbb{R}$, obtain a complex-valued coefficient $c_\omega \in \mathbb{C}$ as definedb y (2.4), (2.5), and (2.10). This collection of coefficients can be encoded by a complex-valued function $f : \mathbb{R} \to \mathbb{C}$ (called "f hat"), which assigns to each frequency parameter coefficient c_ω: (2.11) $\hat{f}(\omega) \coloneqq c_\omega$. The function $f\hat{f}$ is referred to as *Fourier transform* of f, & its value $\hat{f}(\omega) = c_\omega$ are called *Fourier coefficients*. 1 main result in Fourier analysis: Fourier transform can be computed via following compact formula: (2.12)

$$\hat{f}(\omega) = \int_{\mathbb{R}} f(t) \exp(-2\pi i \omega t)\, \mathrm{d}t = \int_{\mathbb{R}} f(t) \cos(-2\pi\omega t)\, \mathrm{d}t + i \int_{\mathbb{R}} f(t) \sin(-2\pi\omega t)\, \mathrm{d}t.$$

I.e., real part of complex coefficient $\hat{f}(\omega)$ is obtained by comparing original signal f with a cosine function of frequency ω, & imaginary part is obtained by comparing with a sine function of frequency ω. Absolute value $|\hat{f}(\omega)|$ is also called *magnitude* of Fourier coefficient. Similarly, real-valued function $|\hat{f}| : \mathbb{R} \to \mathbb{R}$, which assigns to each frequency parameter ω magnitude $|\hat{f}(\omega)|$, is called *magnitude Fourier transform* of f.
In standard literature on signal processing, formula (2.12) is often used to define Fourier transform $\hat{f}$ & then physical interpretation of Fourier coefficients is discussed. In particular, real-valued coefficients d_ω in (2.4) & φ_ω in (2.5) can be derived from $\hat{f}(\omega)$. Using (2.10), one obtains

$$d_\omega = \sqrt{2}|\hat{f}(\omega)|,\ \varphi_\omega = -\frac{\gamma_\omega}{2\pi},$$

· 2.1.1.6. Fourier Representation. Original signal f can be reconstructed from its Fourier transform. In principle, reconstruction is straightforward: one superimposes sinusoids of all possible frequency parameters $\omega \in \mathbb{R}$, each weighted by respective coefficient d_ω & shifted by φ_ω. Both kinds of information are encoded in complex Fourier coefficient

c_ω. In analog case considered so far, dealing with a continuum frequency parameters, where superposition becomes an integration over parameter space. Reconstruction is given by formulas

$$f(t) = \int_{\omega\in\mathbb{R}_{\geq 0}} d_\omega \sqrt{2}\cos(2\pi(\omega t - \varphi_\omega))\,\mathrm{d}\omega = \int_{\omega\in\mathbb{R}} c_\omega \exp(2\pi i\omega t)\,\mathrm{d}\omega,$$

1st given in real-valued formulation, & then given in complex-valued formulation with $c_\omega = \hat{f}(\omega)$. Representation of a signal in terms of a weighted superposition of sinusoidal prototype oscillations is also called *Fourier representation* of signal. Notice formula (2.12) for Fourier transform & (2.17) for Fourier representation are nearly identical. Main difference: roles of time parameter t & frequency parameter ω are interchanged. Beautiful relationship between these 2 formulas will be further discussed in later sects of this chap.

∗ 2.1.2. Examples. Consider some examples including in Fig. 2.1. Fig. 2.5: Waveform & magnitude Fourier transform of a tone C4 (261.6 Hz) played by different instruments. (a) Piano. (b) Trumpet. (c) Violin. (d) Flute. shows waveform & magnitude Fourier transform for some audio signals, where a single note C4 is played on different instruments: a piano, a trumpet, a violin, & a flute. Have already encountered this example in Fig. 1.23 of Sect. 1.3.4, where discussed aspect of timbre. Recall: existence of certain partials & their relative strengths have a crucial influence on timbre of a musical tone. In case of piano tone (Fig. 2.5a), Fourier transform has a sharp peak at 262 Hz, which reveals: most of signal's energy is contained in 1st partial or fundamental frequency of note C4. Further peaks (also beyond shown frequency range from 0–1000 Hz) can be found at integer multiples of fundamental frequency corresponding to higher partials.

– Hãy xem xét 1 số ví dụ bao gồm trong Hình 2.1. Hình 2.5: Biến đổi Fourier dạng sóng & độ lớn của 1 nốt C4 (261,6 Hz) được chơi bởi các nhạc cụ khác nhau. (a) Đàn piano. (b) Kèn trumpet. (c) Đàn violin. (d) Sáo. cho thấy biến đổi Fourier dạng sóng & độ lớn cho 1 số tín hiệu âm thanh, trong đó 1 nốt C4 duy nhất được chơi trên các nhạc cụ khác nhau: đàn piano, kèn trumpet, đàn violin, & sáo. Đã gặp ví dụ này trong Hình 1.23 của Mục 1.3.4, trong đó thảo luận về khía cạnh âm sắc. Nhớ lại: sự tồn tại của 1 số thành phần & cường độ tương đối của chúng có ảnh hưởng quan trọng đến âm sắc của 1 giai điệu nhạc. Trong trường hợp âm piano (Hình 2.5a), biến đổi Fourier có đỉnh nhọn ở 262 Hz, điều này cho thấy: hầu hết năng lượng của tín hiệu nằm ở tần số thành phần thứ nhất hoặc tần số cơ bản của nốt C4. Các đỉnh tiếp theo (cũng nằm ngoài dải tần số hiển thị từ 0-1000 Hz) có thể được tìm thấy ở bội số nguyên của tần số cơ bản tương ứng với các phần cao hơn.

Fig. 2.5b shows: same note played on a trumpet results in a similar frequency spectrum, where peaks appear again at integer multiples of fundamental frequency. However, most of energy is now contained in 3rd partial, & relative why a trumpet sounds different from a piano. For a violin, as shown by Fig. 2.5c, most energy is again contained in 1st partial. Observe: peaks are blurred in frequency, which is result of vibrato (see also Fig. 1.23b). Time-dependent frequency modulations of vibrato are averaged by Fourier transform. This yields a single coefficient for each frequency independent of spectro-temporal fluctuations. A similar explanation holds for flute tone shown in Fig. 2.5d.

– Hình 2.5b cho thấy: cùng 1 nốt nhạc được chơi trên kèn trumpet dẫn đến phổ tần số tương tự, trong đó các đỉnh lại xuất hiện ở bội số nguyên của tần số cơ bản. Tuy nhiên, phần lớn năng lượng hiện được chứa trong phần thứ 3, & lý do tại sao tiếng kèn trumpet lại khác với tiếng đàn piano. Đối với đàn violin, như thể hiện trong Hình 2.5c, phần lớn năng lượng lại được chứa trong phần thứ 1. Quan sát: các đỉnh bị nhòe về tần số, là kết quả của rung âm (xem thêm Hình 1.23b). Các điều chế tần số phụ thuộc thời gian của rung âm được tính trung bình bằng phép biến đổi Fourier. Điều này tạo ra 1 hệ số duy nhất cho mỗi tần số độc lập với các biến động phổ-thời gian. 1 lời giải thích tương tự cũng đúng đối với âm sáo được thể hiện trong Hình 2.5d.

Have seen: magnitude of Fourier transform tells us about signal's overall frequency content, but it does not tell us at which time frequency content occurs. Fig. 2.6: Missing time information of Fourier transform illustrated by 2 different signals & their magnitude Fourier transforms. (a) 2 subsequent sinusoids of frequency 1 Hz & 5 Hz. (b) Superposition of same sinusoids. illustrates this fact, showing waveform & magnitude Fourier transform for 2 signals. 1st signal consists of 2 parts with a sinusoid of $\omega = 1$ Hz & amplitude $A = 1$ in 1st part & a sinusoid of $\omega = 5$ Hz & amplitude $A = 0.7$ in 2nd part. Furthermore, signal is 0 outside interval $[0, 10]$. In contrast, 2nd signal is a superposition of these 2 sinusoids, being 0 outside interval $[0, 5]$. Even though 2 signals are different in nature, resulting magnitude Fourier transforms are more or less same. This demonstrates drawbacks of Fourier transform when analyzing signals with changing characteristics over time. In Sect. 2.1.4 & Sect. 2.5 discuss a short-time version of Fourier transform, where time information is recovered at least to some degree. Besides 2 peaks, one can observe in Fig. 2.6 a large number of small "ripples". Such phenomena as well as further properties of Fourier transform are discussed in Sect. 2.3.3.

– Đã thấy: độ lớn của phép biến đổi Fourier cho chúng ta biết về nội dung tần số tổng thể của tín hiệu, nhưng nó không cho chúng ta biết nội dung tần số xảy ra tại thời điểm nào. Hình 2.6: Thông tin thời gian bị thiếu của phép biến đổi Fourier được minh họa bằng 2 tín hiệu khác nhau & biến đổi Fourier độ lớn của chúng. (a) 2 sin liên tiếp có tần số 1 Hz & 5 Hz. (b) Sự chồng chập của cùng 1 sin. minh họa cho thực tế này, hiển thị dạng sóng & biến đổi Fourier độ lớn cho 2 tín hiệu. Tín hiệu thứ nhất bao gồm 2 phần với 1 sin $\omega = 1$ Hz & biên độ $A = 1$ trong phần thứ nhất & 1 sin $\omega = 5$ Hz & biên độ $A = 0,7$ trong phần thứ hai. Hơn nữa, tín hiệu là 0 bên ngoài khoảng $[0, 10]$. Ngược lại, tín hiệu thứ 2 là sự chồng chập của 2 sin này, là 0 bên ngoài khoảng $[0, 5]$. Mặc dù 2 tín hiệu có bản chất khác nhau, nhưng biên độ biến đổi Fourier thu được ít nhiều giống nhau. Điều này chứng minh những nhược điểm của biến đổi Fourier khi phân tích các tín hiệu có đặc điểm thay đổi theo thời gian. Trong Phần 2.1.4 & Phần 2.5 thảo luận về phiên bản biến đổi Fourier thời gian ngắn, trong đó thông tin thời gian được phục hồi ít nhất ở 1 mức độ nào đó. Bên cạnh 2 đỉnh, người ta có thể quan sát thấy trong Hình 2.6 1 số lượng lớn "gợn sóng" nhỏ. Các hiện tượng như vậy cũng như các tính chất khác của biến đổi Fourier được thảo luận trong Phần 2.3.3.

∗ 2.1.3. Discrete Fourier Transform. When using digital technology, only a finite number of parameters can be stored & processed. To this end, analog signals need to be converted into finite representations – a process commonly referred to as *digitization*. 1 step that is often applied in an analog-to-digital conversion is known as *equidistant sampling*. Given an analog signal $f : \mathbb{R} \to \mathbb{R}$ & $T \in (0, \infty)$, one defines a function $x : \mathbb{Z} \to \mathbb{R}$ by setting (2.18) $x(n) \coloneqq f(nT)$. Since x is only defined on a discrete set of time points, it is also referred to as a *discrete-time* (DT) signal (Sect. 2.2.2.1). Value $x(n)$ is called a *sample* taken at time $t = nT$ of original analog signal f. This procedure is also known as T*-sampling*, where number T is referred to as *sampling period*. Inverse $F_s \coloneqq \frac{1}{T}$ of sampling period is also called *sampling rate* of process. It specifies number of samples per sec & is measured in Hertz (Hz). Fig. 2.7a: Illustration of sampling process using a sampling rate of $F_s = 32$. Waveforms of analog signals are shown as curves & sampled versions as stem plots. (a) Signal f. shows an example of sampling an analog signal using $F_s = 32$ Hz.

In general, one loses information in sampling process. Famous *sampling theorem* says: original analog signal f can be reconstructed perfectly from its sampled version x, if f does not contain any frequencies higher than (2.20) $\Omega \coloneqq \frac{F_s}{2} = \frac{1}{2T}$ Hz. In this case, also say: f is an Ω*-bandlimited* signal, where frequency Ω is known as *Nyquist frequency*. In case f contains higher frequencies, sampling may cause artifacts referred to as *aliasing* (Sect. 2.2.2 for details). Sampling theorem will be further discussed in Exercise 2.28.

In following, assume: analog signal f satisfies suitable requirements so that sampled signal x does not contain major artifacts. Now, having a discrete number of samples to represent our signal, how do we calculate Fourier transform? Recall: idea of Fourier transform: compare signal with a sinusoidal prototype oscillation by computing integral over pointwise product (2.12). Therefore, in digital domain, it seems reasonable to sample sinusoidal prototype oscillation in same fashion as signal (Fig. 2.7b: Sinusoid $\cos_{\omega,\varphi}$ with $\omega = 2, \varphi = 0$.). By multiplying 2 sampled functions in a pointwise fashion, obtain a sampled product Fig. 2.7c: Product $f \cos_{\omega,\varphi}$ & its area. Finally, integration in analog case becomes summation in discrete case, where summands need to be weighted by sampling period T. As a result, one obtains approximation: (2.21)

$$\sum_{n\in\mathbb{Z}} Tf(nT) \exp(-2\pi i\omega nT) \approx \hat{f}(\omega).$$

In mathematical terms, sum can be interpreted as overall area of rectangular shapes that approximates area corresponding to integral (Fig. 2.7d: Approximation of integral by a Riemann sum obtained from sampled version.). Such an approximation is also known as a *Riemann sum*. As show in Sect. 2.3.4, quality of approximation is good for "well-behaved" signals f & "small" frequency parameters ω.

– Theo thuật ngữ toán học, tổng có thể được hiểu là tổng diện tích của các hình chữ nhật xấp xỉ diện tích tương ứng với tích phân (Hình 2.7d: Xấp xỉ tích phân bằng tổng Riemann thu được từ phiên bản lấy mẫu.). 1 phép xấp xỉ như vậy cũng được gọi là *tổng Riemann*. Như thể hiện trong Phần 2.3.4, chất lượng xấp xỉ là tốt đối với các tín hiệu "hoạt động tốt" f & các tham số tần số "nhỏ" ω.

One defines a discrete version of Fourier transform for a given DT-signal $x : \mathbb{Z} \to \mathbb{R}$ by setting (2.22)

$$\hat{x}(\omega) \coloneqq \sum_{n\in\mathbb{Z}} x(n) \exp(-2\pi i\omega n).$$

In this def, where a simple 1-sampling (i.e., T-sampling with $T = 1$) of exponential function is used, one does not assume: one knows relation between x & original signal f. If one is interested in recovering relation to Fourier transform $\hat{f}$, one needs to know sampling period T. Based on (2.21), an easy calculation shows that (2.23)

$$\hat{x}(\omega) \approx \frac{1}{T}\hat{f}\left(\frac{\omega}{T}\right).$$

In this approximation, frequency parameter ω used for $\hat{x}$ corresponds to frequency $\frac{\omega}{T}$ for $\hat{f}$. In particular, $\omega = \frac{1}{2}$ for $\hat{x}$ corresponds to Nyquist frequency $\Omega = \frac{1}{2T}$ of sampling process. Therefore, assuming f is bandlimited by $\Omega = \frac{1}{2T}$, one needs to consider only frequencies with $0 \le \omega \le \frac{1}{2}$ for $\hat{x}$. In digital case, all other frequency parameters are redundant & yield meaningless approximations.

– Trong phép xấp xỉ này, tham số tần số ω được sử dụng cho $\hat{x}$ tương ứng với tần số $\frac{\omega}{T}$ cho $\hat{f}$. Đặc biệt, $\omega = \frac{1}{2}$ cho $\hat{x}$ tương ứng với tần số Nyquist $\Omega = \frac{1}{2T}$ của quá trình lấy mẫu. Do đó, giả sử f bị giới hạn bởi $\Omega = \frac{1}{2T}$, người ta chỉ cần xem xét các tần số có $0 \le \omega \le \frac{1}{2}$ cho $\hat{x}$. Trong trường hợp kỹ thuật số, tất cả các tham số tần số khác đều dư thừa & tạo ra các phép xấp xỉ vô nghĩa.

For doing computations on digital machines, still have some problems. 1 problem: sum in (2.22) involves an infinite number of summands. Another problem: frequency parameter ω is a continuous parameter. For both problems, there are some pragmatic solutions. Regarding 1st problem, assume most of relevant information of f is limited to a certain duration in time. [Strictly speaking, this assumption is problematic since it conflicts with requirement of f being bandlimited. A mathematical fact states: there are no functions that are both limited in frequency (bandlimited) & limited in time (having finite duration).] E.g., a music recording of a song hardly lasts for > 10 minutes. Having a finite duration means: analog signal f is assumed to be 0 outside a compact interval. By possibly shifting signal, may assume: this interval starts at time $t = 0$. I.e., only need to consider a finite number of samples $x(0), x(1), \ldots, x(N-1)$ for some suitable number $N \in \mathbb{N}$. As a result, sum in (2.22) becomes finite.

Regarding 2nd problem, one computes Fourier transform only for a finite number of frequencies. Similar to sampling of time axis, one typically samples frequency axis by considering frequencies $\omega = \frac{k}{M}$ for some suitable $M \in \mathbb{N}$ & $k \in [0 : M-1]$.

In practice, one often couples number N of samples & number M that determines frequency resolution by setting $N = M$. Note: 2 numbers N, M refer to different aspects. However, coupling is convenient. It not only makes resulting transform invertible, but also leads to a computationally efficient algorithm, as see in Sect. 2.4.3. Setting $X(k) \coloneqq \hat{x}(\frac{k}{N})$ & assuming $x(0), x(1), \ldots, x(N-1)$ are relevant samples (all others being 0), obtain fro (2.22) formula

$$X(k) = \hat{x}\left(\frac{k}{N}\right) = \sum_{n}^{N-1} x(n) \exp\left(-\frac{2\pi i k n}{N}\right), \ \forall k \in [0 : M-1] = [0 : N-1].$$

This transform is also known as *discrete Fourier transform* (DFT), covered in Sect. 2.4.
Have a look at frequency information supplied by Fourier coefficient $X(k)$. By (2.23) frequency ω of $\hat{x}$ corresponds to $\frac{\omega}{T}$ of $\hat{f}$. Therefore, index k of $X(k)$ corresponds to physical frequency (2.25)

$$F_{\text{coef}}(k) \coloneqq \frac{k}{NT} = \frac{kF_s}{N}$$

given in Hertz. As discuss in Sect. 2.4.4, coefficients $X(k)$ need to be taken with care. 1st, approximation quality in (2.23) may be rather poor, in particular for frequencies close to Nyquist frequency. 2nd, for a real-valued signal x, Fourier transform fulfills certain symmetry properties (see Exercise 2.24). As a result, upper half of Fourier coefficients are redundant, & one only needs to consider coefficients $X(k)$ for $k \in [0 : \lfloor \frac{N}{2} \rfloor]$. Note: in case of an even number N, index $k = \frac{N}{2}$ corresponds to $F_{\text{coef}}(k) = \frac{F_s}{2}$, which is Nyquist frequency of sampling process.
Finally, consider some efficiency issues when computing DFT. To compute a single Fourier coefficient $X(k)$, one requires a number of multiplications & additions linear in N. Therefore, to compute all coefficients $X(k)$ for $[k : \frac{N}{2}]$ one after another, one requires a number of operations on order of N^2. Despite being a finite number of operations, e.g. a computational approach is too slow for many practical applications, in particular when N is large.
Number of operations can be reduced drastically by using an efficient algorithm known as *fast Fourier transform* (FFT). FFT algorithm, which was discovered by Gauss & Fourier 200 years ago, has changed whole industries & is now being used in billions of telecommunication & other devices. FFT exploits redundancies across sinusoids of different techniques to jointly compute all Fourier coefficients by a recursion. This recursion works particularly well in case N is a power of 2. As a result, FFT reduces overall number of operations from order of N^2 to order of $N \log_2 N$. Savings are enormous. E.g., using $N = 2^{10} = 1024$, FFT requires roughly $N \log_2 N = 10240$ instead of $N^2 = 1048576$ operations in naive approach – a savings factor of about 100. In case of $N = 2^{20}$, savings amount to a factor of about 50000 (Exercise 2.6). In Sect. 2.4.3, discuss algorithmic details of FFT.

∗ 2.1.4. Short-Time Fourier Transform. Fourier transform yields frequency information that is averaged over entire time domain. However, information on *when* these frequencies occur is hidden in transform. Have already seen this phenomenon in Fig. 2.6a, where change in frequency is not revealed when looking at magnitude of Fourier transform. To recover hidden time information, Dennis Gabor introduced in 1946 *short-time Fourier transform* (STFT). Instead of considering entire signal, main idea of STFT: consider only a small sect of signal. To this end, one fixes a so-called *window function*, which is a function that is nonzero for only a short period of time (defining considered sect). Original signal is then multiplied with window function to yield a *windowed signal*. To obtain frequency information at different time instances, one shifts window function across time & computes a Fourier transform for each of resulting windowed signals.
This idea is illustrated by Fig. 2.8: Signal & Fourier transform consisting of 2 subsequent sinusoids of frequency 1 Hz & 5 Hz. (a) Original signal. (b) Windowed signal centered at $t = 3$. (c) Windowed signal centered at $t = 5$. (d) Windowed signal centered at $t = 7$., which continues our example from Fig. 2.6a. To obtain local sections of original signal, one multiplies signal with suitably shifted rectangular window functions. In Fig. 2.8b, resulting local section only contains frequency content at 1 Hz, which leads to a single main peak in Fourier transform at $\omega = 1$. Further shifting time window to right, resulting section contains 1 Hz as well as 5 Hz components Fig. 2.8c. These components are reflected by 2 peaks at $\omega = 1$ & $\omega = 5$. Finally, section shown in Fig. 2.8d only contains frequency content at 5 Hz.
Already at this point, emphasize: STFT reflects not only properties of original signal but also those of window function. 1st of all, STFT depends on length of window, which determines size of section. Then, STFT is influenced by shape of window function. E.g., sharp edges of rectangular window typically introduce "ripple" artifacts. In Sect. 2.5.1, discuss such issues in more detail. In particular, introduce more suitable, bell-shaped window functions, which typically reduce such artifacts.
In Sect. 2.5, one finds a detailed treatment of analog & discrete versions of STFT & their relationship. In following, only consider discrete case & specify most important mathematical formulas as needed in practical applications. Let $x : \mathbb{Z} \to \mathbb{R}$ be a real-valued DT-signal obtained by equidistant sampling w.r.t. a fixed sampling rate F_s given in Hertz. Furthermore, let $w : [0 : N-1] \to \mathbb{R}$ be a sampled window function of length $N \in \mathbb{N}$. E.g., in case of a rectangular window one has $w(n) = 1$ for $n \in [0 : N-1]$. Implicitly, one assumes $w(n) = 0$ for all other time parameters $n \in \mathbb{Z} \backslash [0 : N-1]$ outside this window. Length parameter N determines duration of considered sections, which amounts to $\frac{N}{F_s}$ secs. One also introduces an additional parameter $H \in \mathbb{N}$, which is referred to as *hop size*. Hop size parameter is specified in samples & determines step size in which window is to be shifted across signal.
– Trong Phần 2.5, người ta tìm thấy cách xử lý chi tiết các phiên bản tương tự & rời rạc của STFT & mối quan hệ của chúng. Sau đây, chỉ xem xét trường hợp rời rạc & chỉ định các công thức toán học quan trọng nhất khi cần trong các ứng dụng thực tế. Cho $x : \mathbb{Z} \to \mathbb{R}$ là tín hiệu DT có giá trị thực thu được bằng cách lấy mẫu cách đều với tốc độ lấy mẫu cố định F_s được đưa ra theo Hertz. Hơn nữa, cho $w : [0 : N-1] \to \mathbb{R}$ là hàm cửa sổ lấy mẫu có độ dài $N \in \mathbb{N}$. E.g., trong

trường hợp cửa sổ hình chữ nhật, ta có $w(n) = 1$ đối với $n \in [0 : N-1]$. Theo mặc định, ta giả định $w(n) = 0$ đối với tất cả các tham số thời gian khác $n \in \mathbb{Z}\backslash[0 : N-1]$ bên ngoài cửa sổ này. Tham số chiều dài N xác định thời lượng của các phần được xem xét, tương đương với $\frac{N}{F_s}$ giây. Người ta cũng giới thiệu 1 tham số bổ sung $H \in \mathbb{N}$, được gọi là kích thước nhảy. Tham số kích thước nhảy được chỉ định trong các mẫu & xác định kích thước bước mà cửa sổ sẽ được dịch chuyển qua tín hiệu.
With regard to these parameters, *discrete STFT* $\mathcal{X}$ of signal x is given by (2.26)

$$\mathcal{X}(m,k) \coloneqq \sum_{n=0}^{N-1} x(n+mH)w(n)\exp\left(-\frac{2\pi ikn}{N}\right) \text{ with } m \in \mathbb{Z},\ k \in [0:K].$$

Number $K = \frac{N}{2}$ (assuming N is even) is frequency index corresponding to Nyquist frequency. Complex number $\mathcal{X}(m,k)$ denotes kth Fourier coefficient for mth time frame. Note: for each fixed time frame m, one obtains a *spectral vector* of size $K+1$ given by coefficients $\mathcal{X}(m,k)$ for $k \in [0:K]$. Computation of each such spectral vector amounts to a DFT of size N as in (2.24), which can be done efficiently using FFT.
What have we actually computed in (2.26) in relation to original analog signal f? As for temporal dimension, each Fourier coefficient $\mathcal{X}(m,k)$ is associated with physical time position (2.27)

$$T_{\text{coef}}(m) \coloneqq \frac{mH}{F_s}$$

given in secs. E.g., for smallest possible hop size $H = 1$, one obtains $T_{\text{coef}}(m) = \frac{m}{F_s} = mT$ sec. In this case, one obtains a spectral vector for each sample of DT-signal x, which results in a huge increase in data volume. Furthermore, considering sections that are only shifted by 1 sample generally yields very similar spectral vectors. To reduce this type of redundancy, one typically relates hop size to length N of window. E.g., one often chooses $H = \frac{N}{2}$, which constitutes a good trade-off between a reasonable temporal resolution & data volume comprising all generated spectral coefficients. As for frequency dimension, have in (2.25) $F_{\text{coef}} \coloneqq \frac{k}{NT} = \frac{kF_s}{N}$: index k of $\mathcal{X}(m,k)$ corresponds to physical frequency (2.28)

$$F_{\text{coef}}(k) \coloneqq \frac{k}{NT} = \frac{kF_s}{N}$$

given in Hertz.
Before look at some concrete examples, 1st introduce concept of a *spectrogram*, which denote by $\mathcal{Y}$. Spectrogram is a 2D representation of squared magnitude of STFT: (2.29)

$$\mathcal{Y}(m,k) \coloneqq |\mathcal{X}(m,k)|^2.$$

It can be visualized by means of a 2D image, where horizontal axis represents time & vertical axis represents frequency. In this image, spectrogram value $\mathcal{Y}(m,k)$ is represented by intensity or color in image at coordinate (m,k). Note: in discrete case, time axis is indexed by frame indices m & frequency axis is indexed by frequency indices k.
Continuing our running example from Fig. 2.8, consider a sampled version of analog signal using a sampling rate of $F_s = 32$ Hz. Having a physical duration of 10 sec, this results in 320 samples (Fig. 2.9a: DT-signal sampled with $F_s = 32$ Hz & spectrogram using a window length of $N = 64$ & a hop size of $H = 8$. (a) DT-signal with time axis given in samples.). Using a window length of $N = 64$ samples & a hop size of $H = 8$ samples, obtain spectrogram as shown in Fig. 2.9b: Spectrogram with time axis given in frames & frequency axis given in indices. In image, shade of gray encodes magnitude of a spectral coefficient, where darker colors correspond to larger values. By (2.27), mth frame corresponds to physical time $T_{\text{coef}}(m) = \frac{m}{4}$ sec. I.e., STFT has a time resolution of 4 frames per sec. Furthermore, by (2.28), kth Fourier coefficient corresponds to physical frequency $F_{\text{coef}}(k) \coloneqq \frac{k}{2}$ Hz. I.e., one obtains a frequency resolution of 2 coefficients per Hertz. Plots of waveform & spectrogram with physically correct time & frequency axes are shown in Fig. 2.9c: DT-signal with time axis given in secs. & Fig. 2.9d: Spectrogram with time axis given in secs & frequency axis given in Hertz., resp.
Consider some typical settings as encountered when processing music signals. E.g., in case of CD recordings one has a sampling rate of $F_{\text{s}} = 44100$ Hz. Using a window length of $N = 4096$ & a hop size of $H = \frac{N}{2}$, this results in a time resolution of $\frac{H}{F_s} \approx 46.4$ ms by (2.27) & a frequency resolution of $\frac{F_s}{N} \approx 10.8$ Hz by (2.28). To obtain a better frequency resolution, one may increase window length N. This, however, leads to a poorer localization in time so that resulting STFT loses its capability of capturing local phenomena in signal. This kind of trade-off is further discussed in Sect. 2.5.2 & exercises.
Close this sect with a further example shown in Fig. 2.10: Waveform & spectrogram of a music recording of a C-major scale played on a piano. (a) Recording's underlying musical score. (b) Waveform. (c) Spectrogram. (d) Spectrogram with magnitudes given in dB., which is a recording of a C-major scale played on a piano. 1st note of this scale is C4, which have already considered in Fig. 2.1. In Fig. 2.10c, spectrogram representation of recording is shown, where time & frequency axes are labeled in a physically meaningful way. Spectrogram reveals frequency information of played notes over time. For each note, one can observe horizontal lines that are stacked on top of each other. As discussed in Sect. 1.3.4, these equally spaced lines corresponds to partials, integer multiplies of fundamental frequency of a note. Obviously, higher partials contain less & less of signal's energy. Furthermore, decay of each note over time is reflected by fading out of horizontal lines. To enhance small sound components that may still be perceptually relevant, one often uses a logarithmic dB scale (Sect. 1.3.3). Fig. 2.10d illustrates effect when applying dB scale to values of spectrogram. Besides an enhancement of

higher partials, one can now observe vertical structures at notes' onset positions. These structures correspond to noise-like transients that occur in attack phase of piano sound (Sect. 1.3.4).

This concludes our "nutshell section" covering most important definitions & properties of Fourier transform as needed for subsequent chaps of this book. In particular, formula (2.26) of discrete STFT as well as physical interpretation of time parameter (2.27) & frequency parameter (2.28) are of central importance for most music processing applications to be discussed. As said in introduction, provide in subsequent sections of this chap some deeper insights into mathematics underlying Fourier transform. In particular, explain in more detail connection between various kinds of signals & associated Fourier transforms.

– Điều này kết thúc "phần tóm tắt" của chúng tôi bao gồm hầu hết các định nghĩa & tính chất quan trọng của phép biến đổi Fourier khi cần cho các chương tiếp theo của cuốn sách này. Đặc biệt, công thức (2.26) của STFT rời rạc cũng như cách diễn giải vật lý của tham số thời gian (2.27) & tham số tần số (2.28) có tầm quan trọng cốt lõi đối với hầu hết các ứng dụng xử lý âm nhạc sẽ được thảo luận. Như đã nói trong phần giới thiệu, hãy cung cấp trong các phần tiếp theo của chương này 1 số hiểu biết sâu sắc hơn về toán học cơ bản của phép biến đổi Fourier. Đặc biệt, hãy giải thích chi tiết hơn về mối liên hệ giữa các loại tín hiệu & các phép biến đổi Fourier liên quan.

- 2.2. Signals & Signal Spaces. In technical fields e.g. engineering or CS, a *signal* is a function that conveys information about state or behavior of a physical system. E.g., a signal may describe time-varying sound pressure at some place, motion of a particle through some space, distribution of light on a screen representing an image, or sequence of images as in case of a video signal. In following, consider case of audio signals as discussed in Sect. 1.3. Have seen: such a signal can be graphically represented by its waveform, which depicts amplitude of air pressure over time. In following, introduce mathematical notation that is necessary to formally model such a signal. Doing so, distinguish between 2 different types of signals: *analog signals* as occur around us in real world & *digital signals* as are processed by computers. Show how signals can be modified & combines to yield new signals by applying mathematical operations. Some operations can be applied only if involved signals satisfy certain properties. This leads us to concept of *signal spaces*, a kind of universe that comprises signals that share a certain property.

  – Tín hiệu & Không gian tín hiệu. Trong các lĩnh vực kỹ thuật như kỹ thuật hoặc khoa học máy tính, *tín hiệu* là 1 hàm truyền tải thông tin về trạng thái hoặc hành vi của 1 hệ thống vật lý. E.g., 1 tín hiệu có thể mô tả áp suất âm thanh thay đổi theo thời gian tại 1 số nơi, chuyển động của 1 hạt qua 1 không gian nào đó, phân bố ánh sáng trên màn hình biểu diễn 1 hình ảnh hoặc chuỗi hình ảnh như trong trường hợp tín hiệu video. Sau đây, hãy xem xét trường hợp tín hiệu âm thanh như đã thảo luận trong Mục 1.3. Đã thấy: 1 tín hiệu như vậy có thể được biểu diễn đồ họa bằng dạng sóng của nó, mô tả biên độ áp suất không khí theo thời gian. Sau đây, hãy giới thiệu ký hiệu toán học cần thiết để mô hình hóa chính thức 1 tín hiệu như vậy. Làm như vậy, hãy phân biệt giữa 2 loại tín hiệu khác nhau: *tín hiệu tương tự* xảy ra xung quanh chúng ta trong thế giới thực & *tín hiệu kỹ thuật số* được xử lý bởi máy tính. Hiển thị cách tín hiệu có thể được sửa đổi & kết hợp để tạo ra tín hiệu mới bằng cách áp dụng các phép toán. 1 số phép toán chỉ có thể được áp dụng nếu các tín hiệu liên quan thỏa mãn 1 số thuộc tính nhất định. Điều này đưa chúng ta đến khái niệm về *signal spaces*, 1 loại vũ trụ bao gồm các tín hiệu có cùng 1 đặc tính nhất định.

  * 2.2.1. Analog Signals. As already defined in Sect. 2.1.1, an *analog* signal is a function $f : \mathbb{R} \to \mathbb{R}$, which assigns an amplitude value $f(t) \in \mathbb{R}$ to each time point $t \in \mathbb{R}$. In analog case, both time domain as well as range of amplitude values are represented by $\mathbb{R}$, which is a continuous range of values. This makes it possible to model infinitesimally small changes in both time & amplitude. In case of having a continuous time axis (given by $\mathbb{R}$), one also speaks of *continuous-time* (CT) signals. A signal f is called *periodic* with *period* $\lambda \in \mathbb{R}_{>0}$ if $f(t) = f(t+\lambda)$, $\forall t \in \mathbb{R}$. If there exists a least positive constant with this property, it is called *prime period* of signal (Exercises 2.7–2.8).

    In Sects. 1.3.2 & 2.1.1.1, have already encountered an entire class of analog signals: *sinusoids*. Recall from (2.1): a sinusoid is a periodic function f defined by $f(t) := A\sin(2\pi(\omega t - \varphi))$, $t \in \mathbb{R}$. Parameter A describes *amplitude*, parameter ω: *frequency*, & parameter φ: *phase*. Frequency parameter ω determines period of sinusoid, which is $\lambda = \frac{1}{\omega}$. I.e., a sinusoid of frequency ω repeats every $\lambda = \frac{1}{\omega}$ unit times. In following, use secs as units of time if not specified otherwise. Fig. 2.11: Sinusoid $f(t) = A\sin(2\pi(\omega t - \varphi))$ displayed for $t \in [0,2]$ & for various values of A, ω, φ shows various sinusoids resulting from different parameter settings.

    Besides having a compact description, sinusoids also have an explicit physical meaning with a perceptual correspondence: amplitude A corresponds to loudness & frequency ω to pitch of a sinusoidal sound. Only phase φ, which indicates relative position of an oscillation within its cycle, does not have a direct perceptual correspondence. Note: because of periodicity of a sinusoid, a phase shift by $\varphi + 1$ has same effect as a phase shift by φ. I.e., integer shifts leave a sinusoid unaltered & parameter φ needs to be considered only in interval $[0, 1)$.

    Regarding a signal as a mathematical function is convenient, since this allows us to express modifications of signals in terms of mathematical operations. E.g., *superposition* of 2 signals f, g can be expressed by sum $f + g$ defined as pointwise addition $(f + g)(t) := f(t) + g(t)$, $\forall t \in \mathbb{R}$. Similarly, *scaling* of a signal f by a real factor a is scalar multiple af, which is also defined pointwise by $(af)(t) := af(t)$. Fig. 2.12: Superposition of 3 analog signals. shows an example of a superposition of 3 signals. Have seen in Sect. 2.1: Fourier transform can be regarded as a kind of inverse operation, where a given signal is decomposed into a weighted superposition of elementary signals.

    – Việc coi tín hiệu như 1 hàm toán học là thuận tiện, vì điều này cho phép chúng ta biểu thị các sửa đổi của tín hiệu theo các phép toán. E.g., *chồng chập* của 2 tín hiệu f, g có thể được biểu thị bằng tổng $f + g$ được định nghĩa là phép cộng từng điểm $(f + g)(t) := f(t) + g(t)$, $\forall t \in \mathbb{R}$. Tương tự như vậy, *tỷ lệ* của tín hiệu f theo 1 hệ số thực a là bội số vô hướng af, cũng được định nghĩa từng điểm bởi $(af)(t) := af(t)$. Hình 2.12: Chồng chập của 3 tín hiệu tương tự. cho thấy 1 ví dụ về chồng chập của 3 tín hiệu. Đã thấy trong Phần 2.1: Biến đổi Fourier có thể được coi là 1 loại phép toán nghịch

đảo, trong đó 1 tín hiệu nhất định được phân tích thành 1 phép chồng chập có trọng số của các tín hiệu cơ bản.

* 2.2.2. Digital Signals. Analog signals have a continuous range of values in both time & amplitude, which generally leads to an infinite number of values. Since a computer can only store & process a finite number of values, one has to convert waveform into some *discrete* representation – a process commonly referred to as *digitization*. Some analog signals e.g. sinusoids are already characterized by a small number of parameters, which can be used to represent signal, but for general analog signals one needs other ways for deriving a model that can be described by a finite number of parameters. Furthermore, it should be possible to perform signal manipulations directly in parameter domain s.t. computations become feasible & efficient. Most common approach for digitizing audio signals consists of 2 steps called *sampling & quantization* (Fig. 2.13: 2 steps of a digitization process to transform an analog signal (solid curve) into a digital signal (stem plot). (a) Sampling. (b) Quantization. for an illustration). Now explain these 2 steps in more detail.

  · 2.2.2.1. Sampling. In signal processing, term *sampling* refers to process of reducing a continuous-time (CT) signal to a *discrete-time* (DT) signal, which is defined only on a discrete subset of time axis. By means of a suitable encoding, one often assumes: this discrete set is a subset I of $\mathbb{Z}$. Then a DT-signal is defined to be a function $x : I \to \mathbb{R}$, where domain I corresponds to points in time. Since one can extend any DT-signal from domain I to domain $\mathbb{Z}$ simply by setting all values to 0s for points in $\mathbb{Z}\backslash I$, may assume $I = \mathbb{Z}$. Most common sampling procedure to transform a CT-signal $f : \mathbb{R} \to \mathbb{R}$ into a DT-signal $x : \mathbb{Z} \to \mathbb{R}$ is known as *equidistant sampling*. For convenience, repeat definitions from Sect. 2.1.3. Fixing a positive real $T > 0$, DT-signal x is obtained by setting (2.32)

$$x(n) \coloneqq f(nT),\ \forall n \in \mathbb{Z}.$$

  Value $x(n)$ is called *sample* taken at time $t = nT$ of original analog signal f. In short, this procedure is also called *T-sampling*. Number T is referred to as *sampling period* & inverse $F_s \coloneqq \frac{1}{T}$ as *sampling rate*. Sampling rate specifies number of samples per sec & is measured in Hertz (Hz).
  Fig. 2.13: 2 steps of a digitization process to transform an analog signal (solid curve) into a digital signal (stem plot). (a) Sampling. (b) Quantization. shows an illustrate example, where DT-signal x is represented by red stem plot. In this example, one has 13 samples in 1st 2 secs. Thus, sampling rate is roughly 6.5 Hz & sampling period 0.154 secs. In practical applications, typical sampling rates are 8 kHz (8000 Hz) for telephony, 32 kHz for digital radio, 44.1 kHz for CD recordings, & 48 kHz up to 96 kHz for professional studio technology.
  In general, sampling is a *lossy* operation in sense: information is lost in this process & original analog signal cannot be recovered from its sampled version. Only if analog signal has additional properties in terms of its frequency spectrum is a perfect reconstruction possible. This is assertion of famous *sampling theorem* (Exercise 2.28). Without such additional properties, sampling may cause an effect known as *aliasing*, where certain frequency components of signal become indistinguishable. This effect is illustrated by Fig. 2.14: Illustration of aliasing effect when reducing sampling rate. Figures show original analog signal (solid curve), sampled version (stem plot), & reconstructed analog signal (dotted curve) for sampling rates of (a) 12 Hz. (b) 6 Hz. (c) 3 Hz., which shows an analog signal that is superposition of 2 sinusoids. Using a high sampling rate in Fig. 2.14a, analog signal can be reconstructed with high accuracy. However, when decreasing sampling rate, higher-frequency component is not captured well & only a coarse approximation of original signal remains Fig. 2.14c.

  · 2.2.2.2. Quantization. Have seen how sampling transforms a continuous time axis (encoded by $\mathbb{R}$) into a discrete time axis (encoded by $\mathbb{Z}$). This is only 1st step in an analog-to-digital conversion of a signal. In 2nd step, one needs to replace continuous range of possible amplitudes (again encoded by $\mathbb{R}$) by a discrete range of possible values (encoded by a discrete set $\Gamma \subset \mathbb{R}$). This process is commonly known as *quantization*. Such a quantization can be modeled by a function $Q : \mathbb{R} \to \Gamma$, referred to as *quantizer*, which assigns to each amplitude value $a \in \mathbb{R}$ a value $Q(a) \in \Gamma$. Many of quantizers used simply round off or truncate analog value to some units of precision. E.g., a typical uniform quantizer with a *quantization step size* equal to some value Δ can be defined by

$$Q(a) \coloneqq \operatorname{sgn}(a)\Delta\lfloor\frac{|a|}{\Delta} + \frac{1}{2}\rfloor,\ \forall a \in \mathbb{R}.$$

  Note: in case of $\Delta = 1$, quantizer Q is simple rounding to nearest integer. Like sampling, quantization is generally a loss operation, because different analog values may be mapped to same digital value. Difference between actual analog value & quantized value is called *quantization error* (Exercise 2.9). Reducing quantization step size Δ typically leads to smaller quantization errors. However, at same time, number of quantized values (& therefore also number of bits needed to encode these values) increases. Fig. 2.13b shows result after sampling & quantizing an analog signal. In this example, quantization step size $\Delta = \frac{1}{3}$ is used, resulting in 8 different quantization values for given signal. Hence, a 3-bit coding scheme may be used to represent quantized values. For CD recordings, a 16-bit coding scheme is used, which allows representation of 65536 possible values.
  In summary, after using an analog-to-digital conversion based on sampling & quantization, generally not possible to reconstruct original waveform from digital representation. Aliasing & quantization may introduce audible sound artifacts e.g. harsh buzzing sounds or noise. For digital representations as used for CDs, however, sampling rate as well as quantization resolution are chosen in such ways: degradation of waveform is not noticeable by human ear.

* 2.2.3. Signal Spaces. In prev sects, considered analog & digital signals, which were modeled as CT-signals $f : \mathbb{R} \to \mathbb{R}$ & as DT-signal $x : \mathbb{Z} \to \mathbb{R}$, resp. In following discussion, use symbols f, g to denote CT-signals & symbols x, y to denote DT-signals. For time parameter, typically use parameter t in CT case & parameter n in DT case.

  · 2.2.3.1. Complex Numbers.

· 2.2.3.2. Vector Spaces.
· 2.2.3.3. Inner Products.
· 2.2.3.4. Space $l^2(\mathbb{Z})$.
· 2.2.3.5. Space $L^2(\mathbb{R})$.
· 2.2.3.6. Space $L^2([0,1))$.

◦ 2.3. Fourier Transform. Fourier transform is most important mathematical tool in audio signal processing. As discussed in Sect. 2.1, Fourier transform converts a time-dependent signal into a frequency-dependent function. Inverse process is realized by Fourier representation, which represents a signal as a weighted superposition of independent elementary functions. Each of weights expresses extent to which corresponding elementary function contributes to original signal, thus revealing a certain aspect of signal. Because of their explicit physical interpretation in terms of frequency, sinusoids are particularly suited to serve as elementary functions. Each of weights is then associated to a frequency value & expresses degree to which signal contains a periodic oscillation of that frequency. Fourier transform can be regarded as a way to compute frequency-dependent weights.

– Biến đổi Fourier. Biến đổi Fourier là công cụ toán học quan trọng nhất trong xử lý tín hiệu âm thanh. Như đã thảo luận trong Phần 2.1, biến đổi Fourier chuyển đổi tín hiệu phụ thuộc thời gian thành hàm phụ thuộc tần số. Quá trình ngược lại được thực hiện bằng biểu diễn Fourier, biểu diễn tín hiệu dưới dạng chồng chất có trọng số của các hàm cơ bản độc lập. Mỗi trọng số thể hiện mức độ mà hàm cơ bản tương ứng đóng góp vào tín hiệu gốc, do đó tiết lộ 1 khía cạnh nhất định của tín hiệu. Do cách diễn giải vật lý rõ ràng của chúng theo tần số, các hàm sin đặc biệt phù hợp để đóng vai trò là các hàm cơ bản. Sau đó, mỗi trọng số được liên kết với 1 giá trị tần số & thể hiện mức độ mà tín hiệu chứa dao động tuần hoàn của tần số đó. Biến đổi Fourier có thể được coi là 1 cách để tính toán các trọng số phụ thuộc tần số.

In following, depending on underlying signal space, introduce several variants of Fourier transform & its inverse, Fourier representation. Start with signal space $L^2([0,1))$ consisting of 1-periodic finite-energy CT-signals (Sect. 2.3.1). Continue by showing how formulation of Fourier transform in terms of complex-valued exponential functions (instead of real-valued sinusoids) makes mathematical handling much more convenient (Sect. 2.3.2). Discuss Fourier transform for signal space $L^2(\mathbb{R})$ (Sect. 2.3.3) as well as for signal space $l^2(\mathbb{Z})$ (Sect. 2.3.4). Important to note: each of these signal spaces possesses its own Fourier transform & mathematical concepts needed to prove existence & properties of respective Fourier transform are different for variants. While giving mathematically rigorous definitions of various Fourier transforms, do not provide proofs. In particular for analog case, proofs require results from measure & integration theory, which are outside scope of this book. Instead, try to give some intuitive explanations while highlighting meaning & interrelations of various variants.

– Tiếp theo, tùy thuộc vào không gian tín hiệu cơ bản, hãy giới thiệu 1 số biến thể của phép biến đổi Fourier & nghịch đảo của nó, biểu diễn Fourier. Bắt đầu với không gian tín hiệu $L^2([0,1))$ bao gồm các tín hiệu CT năng lượng hữu hạn chu kỳ 1 (Mục 2.3.1). Tiếp tục bằng cách chỉ ra cách xây dựng phép biến đổi Fourier theo các hàm mũ có giá trị phức (thay vì các sin có giá trị thực) giúp xử lý toán học thuận tiện hơn nhiều (Mục 2.3.2). Thảo luận về phép biến đổi Fourier cho không gian tín hiệu $L^2(\mathbb{R})$ (Mục 2.3.3) cũng như cho không gian tín hiệu $l^2(\mathbb{Z})$ (Mục 2.3.4). Điều quan trọng cần lưu ý: mỗi không gian tín hiệu này đều sở hữu phép biến đổi Fourier riêng & các khái niệm toán học cần thiết để chứng minh sự tồn tại & các tính chất của phép biến đổi Fourier tương ứng là khác nhau đối với các biến thể. Trong khi đưa ra các định nghĩa chặt chẽ về mặt toán học của nhiều phép biến đổi Fourier khác nhau, không cung cấp bằng chứng. Riêng đối với trường hợp tương tự, các bằng chứng đòi hỏi kết quả từ lý thuyết đo lường & tích phân, nằm ngoài phạm vi của cuốn sách này. Thay vào đó, hãy cố gắng đưa ra 1 số giải thích trực quan trong khi làm nổi bật ý nghĩa & mối quan hệ giữa các biến thể khác nhau.

∗ 2.3.1. Fourier Transform for Periodic CT-Signals. p. 69+++
∗ 2.3.2. Complex Formulation of Fourier Transform.

- 3. Music Synchronization.
- 4. Music Structure Analysis.
- 5. Chord Recognition.
- 6. Tempo & Beat Tracking.
- 7. Content-Based Audio Retrieval.
- 8. Musically Informed Audio Decomposition.

1.7 [Väl+06]. Vesa Välimäki, Jyri Pakarinen, Cumhur Erkut, Matti Karjalainen. Discrete-Time Modeling of Musical Instruments

[283 citations]

Question 1 (Cf. Continuous-time modeling vs. Discrete-time modeling)**.** *How about continuous-time modeling of musical instruments?*

Question 2 (Cf. Mathematical modeling technique vs. Physical modeling technique)**.** *Compare Mathematical modeling technique vs. Physical modeling technique.*

- **Abstract.** This article describes physical modeling techniques that can be used for simulating musical instruments. Methods are closely related to digital signal processing. They discretize system w.r.t. time, because aim: run simulation using a computer. Physics-based modeling methods can be classified as mass-spring, modal, wave digital, finite difference, digital waveguide & source-filter models. Present basic theory & a discussion on possible extensions for each modeling technique. For some methods, a simple model example is chosen from existing literature demonstrating a typical use of method. E.g., in case of digital waveguide modeling technique a vibrating string model is discussed, & in case of wave digital filter technique, present a classical piano hammer model. Tackle some nonlinear & time-varying models & include new results on digital waveguide modeling of a nonlinear string. Discuss current trends & future directions in physical modeling of musical instruments.

- 1. Introduction. Musical instruments have historically been among most complicated mechanical systems made by humans. They have been a topic of interest for physicists & acousticians for over a century. Modeling of musical instruments using computers is newest approach to understanding how these instruments work.

  This paper presents an overview of physics-based modeling of musical instruments. Specifically, this paper focuses on sound synthesis methods derived using physical modeling approach. Several previously published tutorial & review papers discussed physical modeling synthesis techniques for musical instruments sounds [73, 129, 251, 255, 256, 274, 284, 294]. Purpose of this paper: give a unified introduction to 6 main classes of discrete-time physical modeling methods, namely mass–spring, modal, wave digital, finite difference, digital waveguide & source–filter models. This review also tackles mixed & hybrid models in which usually 2 different modeling techniques are combined.

  Physical models of musical instruments have been developed for 2 main purposes: research of acoustical properties & sound synthesis. Methods discussed in this paper can be applied to both purpose, but here main focus is sound synthesis. Basic idea of physics-based sound synthesis: build a simulation model of sound production mechanism of a musical instrument & to generate sound with a computer program or signal processing hardware that implements that model. Motto of physical modeling synthesis: when a model has been designed properly, so that it behaves much like actual acoustic instrument, synthetic sound will automatically be natural in response to performance. In practice, various simplifications of model cause sound output to be similar to, but still clearly different from, original sound. Simplifications may be caused by intentional approximations that reduce computational cost or by inadequate knowledge of what is actually happening in acoustic instrument. A typical & desirable simplification: linearization of slightly nonlinear phenomena, which may avert unnecessary complexities, & hence may improve computational efficiency.

  In speech technology, idea of accounting for physics of sound source, human voice production organs, is an old tradition, which has led to useful results in speech coding & synthesis. While 1st experiments on physics-based musical sound synthesis were documented several decades ago, 1st commercial products based on physical modeling synthesis were introduced in 1990s. Thus, topic is still relatively young. Research in field has been very active in recent years.

  1 of motivations for developing a physically based sound synthesis: musicians, composers, & other users of electronic musical instruments have a constant hunger for better digital instruments & for new tools for organizing sonic events. A major problem in digital musical instruments has always been how to control them. For some time, researchers of physical models have hoped: these models would offer more intuitive, & in some ways better, controllability than previous sound synthesis methods. In addition to its practical applications, physical modeling of musical instruments is an interesting research topic for other reasons. It helps to solve old open questions, e.g. which specific features in a musical instrument's sound make it recognizable to human listeners or why some musical instruments sound sophisticated while others sound cheap. Yet another fascinating aspect of this field: when physical principles are converted into computational methods, possible to discover new algorithms. This way, possible to learn new signal processing methods from nature.

- 2. Brief history. Modeling of musical instruments is fundamentally based on understanding of their sound production principles. 1st person attempting to understand how musical instruments work might have been Pythagoras, who lived in ancient Greece around 500 BC. At that time, understanding of musical acoustics was very limited & investigations focused on tuning of string instruments. Only after late 18th century, when rigorous mathematical methods e.g. PDEs were developed, was it possible to build formal models of vibrating strings & plates.

  Earliest work on physics-based discrete-time sound synthesis was probably conducted by Kelly & Lochbaum in context of vocal-tract modeling [145]. A famous early musical example is 'Bicycle Built for 2' (1961), where singing voice was produced using a discrete-time model of human vocal tract. This was result of collaboration between Mathews, Kelly & Lochbaum [43]. 1st vibrating string simulations were conducted in early 1970s by Hiller & Ruiz [113, 114], who discretized wave equation to calculate waveform of a single point of a vibrating string. Computing 1 s of sampled waveform took minutes. A few years later, Cadoz & his colleagues developed discrete-time mass-spring models & built dedicated computing hardware to run real-time simulations [38].

  In late 1970s & early 1980s, McIntyre, Woodhouse, & Schumacher made important contributions by introducing simplified discrete-time models of bowed strings, clarinet & flute [173, 174, 235], & Karplus & Strong [144] invented a simple algorithm that produces string-instrument-like sounds with few arithmetic operations. Based on these ideas & their generalizations, Smith & Jaffe introduced a signal-processing oriented simulation technique for vibrating strings [120, 244]. Soon thereafter, Smith proposed term 'digital waveguide' & developed general theory [247, 249, 253].

  1st commercial product based on physical modeling synthesis, an electronic keyboard instrument by Yamaha, was introduced in 1994 [168]; it used digital waveguide techniques. More recently, digital waveguide techniques have been also employed in MIDI synthesizers on personal computer soundcards. Currently, much of practical sound synthesis is based on software, & there are many commercial & freely available pieces of synthesis software that apply 1 or more physical modeling methods.

- 3. General concepts of physics-based modeling. In this sect, discuss a number of physical & signal processing concepts & terminology that are important in understanding modeling paradigms discussed in subsequent sects. Each paradigm is also characterized briefly in end of this sect. A reader familiar with basic concepts in context of physical modeling & sound synthesis may go directly to Sect. 4.

  - 3.1. Physical domains, variables, & parameters. Physical phenomena can be categorized as belonging to different 'physical domains'. Most important ones for sound sources e.g. musical instruments are acoustical & mechanical domains. In addition, electrical domain is needed for electroacoustic instruments & as a domain to which phenomena from other domains are often mapped. Domains may interact with one another, or they can be used as analogies (equivalent models) of each other. Electrical circuits & networks are often applied as analogies to describe phenomena of other physical domains.

    Quantitative description of a physical system is obtained through measurable quantities that typically come in pairs of variables, e.g. force & velocity in mechanical domain, pressure & volume velocity in acoustical domain or voltage & current in electrical domain. Members of such dual variable pairs are categorized generically as 'across variable' or 'potential variable', e.g. voltage, force or pressure, & 'through variable' or 'kinetic variable', e.g. current, velocity or volume velocity. If there is a linear relationship between dual variables, this relation can be expressed as a parameter, e.g. impedance $Z = \frac{U}{I}$ being ratio of voltage U & current I, or by its inverse, admittance $Y = \frac{I}{U}$. An example from mechanical domain is mobility (mechanical admittance) defined as ratio of velocity & force. When using such parameters, only 1 of dual variables is needed explicitly, because the other one is achieved through constraint rule.

    Modeling methods discussed in this paper use 2 types of variables for computation, 'K-variables' & 'wave variables' (also denoted as 'W-variables'). 'K' comes from Kirchhoff & refers to Kirchhoff continuity rules of quantities in electric circuits & networks [185]. 'W' is shortform for wave, referring to wave components of physical variables. Instead of pairs of across & through as with K-variables, wave variables come in pairs of incident & reflected wave components. Details of wave modeling are discussed in Sects. 7–8, while K-modeling is discussed particularly in Sects. 4 & 10. It will become obvious: these are different formulations of same phenomenon, & possibility to combine both approaches in hybrid modeling will be discussed in Sect. 10.

    Decomposition into wave components is prominent in such wave propagation phenomena where opposite-traveling waves add up to actual observable K-quantities. A wave quantity is directly observable only when there is no other counterpart. It is, however, a highly useful abstraction to apply wave components to any physical case, since this helps in solving computability (causality) problems in discrete-time modeling.

  - 3.2. Modeling of physical structure & interaction. Physical phenomena are observed as structures & processes in space & time. In sound source modeling, interested in dynamic behavior that is modeled by variables, while slowly varying or constant properties are parameters. Physical interaction between entities in space always propagates with a finite velocity, which may differ by orders of magnitude in different physical domains, speed of light being upper limit.

    'Causality' is a fundamental physical property that follows from finite velocity of interaction from a cause to corresponding effect. In many mathematical relations used in physical models causality is not directly observable. E.g., relation of voltage across & current through an impedance is only a constraint, & variables can be solved only within context of whole circuit. Requirement of causality (more precisely temporal order of cause preceding effect) introduces special computability problems in discrete-time simulation, because 2-way interaction with a delay shorter than a unit delay (sampling period) leads to 'delay-free loop problem'. Use of wave variables is advantageous, since incident & reflected waves have a causal relationship. In particular, wave digital filter (WDF) theory, discussed in Sect. 8, carefully treats this problem through use of wave variables & specific scheduling of computation operations.

    Taking finite propagation speed into account requires using a spatially distributed model. Depending on case at hand, this can be a full 3D model e.g. used for room acoustics, a 2D model e.g. for a drum membrane (discarding air loading) or a 1D model e.g. for a vibrating sting. If object to be modeled behaves homogeneously enough as a whole, e.g. due to its small size compared with wavelength of wave propagation, it can be considered a lumped entity that does not need a description of spatial dimensions.

    – Việc tính đến tốc độ lan truyền hữu hạn đòi hỏi phải sử dụng 1 mô hình phân bố không gian. Tùy thuộc vào trường hợp cụ thể, đây có thể là mô hình 3D đầy đủ, ví dụ như được sử dụng cho âm học phòng, mô hình 2D, ví dụ như cho màng trống (loại bỏ tải trọng không khí) hoặc mô hình 1D, ví dụ như cho 1 cú chích rung. Nếu vật thể được mô hình hóa hoạt động đủ đồng nhất như 1 tổng thể, ví dụ như do kích thước nhỏ so với bước sóng truyền sóng, thì nó có thể được coi là 1 thực thể tập trung không cần mô tả về kích thước không gian.

  - 3.3. Signals, signal processing, & discrete-time modeling. In signal processing, signal relationships are typically represented as 1-directional cause-effect chains. Contrary to this, bi-directional interaction is common in (passive) physical systems, e.g. in systems where reciprocity principle is valid. In true physics-based modeling, 2-way interaction must be taken into account. I.e., from signal processing viewpoint, such models are full of feedback loops, which further implicates: concepts of computability (causality) & stability become crucial.

    In this paper, apply digital signal processing (DSP) approach to physics-based modeling whenever possible. Motivation for this: DSP is an advanced theory & tool that emphasizes computational issues, particularly maximal efficiency. This efficiency is crucial for real-time simulation & sound synthesis. Signal flow diagrams are also a good graphical means to illustrate algorithms underlying simulations. Assume: reader is fmailiar with fundamentals of DSP, e.g. sampling theorem [242] to avoid aliasing (also spatial aliasing) due to sampling in time & space as well as quantization effects due to finite numerical precision.

An important class of systems is those that are linear & time invariant (LTI). They can be modeled & simulated efficiently by digital filters. They can be analyzed & processed in frequency domain through linear transforms, particularly by Z-transform & discrete Fourier transform (DFT) in discrete-time case. While DFT processing through fast Fourier transform (FFT) is a powerful tool, it introduces a block delay & does not easily fit to sample-by-sample simulation, particularly when bi-directional physical interaction is modeled.

Nonlinear & time-varying systems bring several complications to modeling. Nonlinearities create new signal frequencies that easily spread beyond Nyquist limit, thus causing aliasing, which is perceived as very disturbing distortion. In addition to aliasing, delay-free loop problem & stability problems can become worse than they are in linear systems. If nonlinearities in a system to be modeled are spatially distributed, modeling task is even more difficult than with a localized nonlinearity. Nonlinearities will be discussed in several sects of this paper, most completely in Sect. 11.

- 3.4. Energetic behavior & stability. Product of dual variables e.g. voltage & current gives power, which, when integrated in time, yields energy. Conservation of energy in a closed system is a fundamental law of physics that should also be obeyed in true physics-based modeling. In musical instruments, resonators are typically passive, i.e. they do not produce energy, while excitation (plucking, bowing, blowing, etc.) is an active process that injects energy to passive resonators.

  Stability of a physical system is closely related to its energetic behavior. Stability can be defined so that energy of system remains finite for finite energy excitations. From a signal processing viewpoint, stability may also be defined so that variables, e.g. voltages, remain within a linear operating range for possible inputs in order to avoid signal clipping & distortion.

  In signal processing systems with 1-directional input–output connections between stable subblocks, an instability can appear only if there are feedback loops. In general, impossible to analyze such a system's stability without knowing its whole feedback structure. Contrary to this, in models with physical 2-way interaction, if each element is passive, then any arbitrary network of such elements remains stable.

- 3.5. Modularity & locality of computation. For a computational realization, desirable to decompose a model systematically into blocks & their interconnections. Such an object-based approach helps manage complex models through use of modularity principle. Abstractions to macro blocks on basis of more elementary ones helps hiding details when building excessively complex models.

  For 1-directional interactions used in signal processing, enough to provide input & output terminals for connecting blocks. For physical interaction, connections need to be done through ports, with each port having a pair of K- or wave variables depending on modeling method used. This allows mathematical principles used for electrical networks [185]. Details on block-wise construction of models will be discussed in following sects for each modeling paradigm.

  Locality of interaction is a desirable modeling feature, which is also related to concept of causality. For a physical system with a finite propagation speed of waves, enough: a block interacts only with its nearest neighbors; it does not need global connections to compute its task & effect automatically propagates throughout system.

  In a discrete-time simulation with bi-directional interactions, delays shorter than a unit delay (including 0 delay) introduce delay-free loop problem that we face several times in this paper. While possible to realize fractional delays [154], delayers shorter than unit delay contain a delay-free component. There are ways to make such 'implicit' system computable, but cost in time (or accuracy) may become prohibitive for real-time processing.

- 3.6. Physics-based discrete-time modeling paradigms. This paper presents an overview of physics-based methods & techniques for modeling & synthesizing musical instruments. Have excluded some methods often used in acoustics, because they do not easily solve task of efficient discrete-time modeling & synthesis. E.g., finite element & boundary element methods (FEM & BEM) are generic & powerful for solving system behavior numerically, particularly for linear systems, but focus on inherently time-domain methods for sample-by-sample computation.

  Main paradigms in discrete-time modeling of musical instruments can be briefly characterized as follows.

  - 3.6.1. Finite difference models. In Sect. 4, finite difference models are numerical replacement for solving PDEs. Differentials are approximated by finite differences so that time & position will be discretized. Through proper selection of discretization to regular meshes, computational algorithms become simple & relatively efficient. Finite difference time domain (FDTD) schemes are K-modeling methods, since wave components are not explicitly utilized in computation. FDTD schemes have been applied successfully to 1D, 2D, & 3D systems, although in linear 1D cases digital waveguides are typically superior in computational efficiency & robustness. In multidimensional mesh structures, FDTD approach is more efficient. It also shows potential to deal systematically with nonlinearities (Sect. 11). FDTD algorithms can be problematic due to lack of numerical robustness & stability, unless carefully designed.
  - 3.6.2. Mass–spring networks. In Sect. 5, mass–spring networks are a modeling approach, where intuitive basic elements in mechanics – masses, springs, & damping elements – are used to construct vibrating structures. It is inherently a K-modeling methodology, which has been used to construct small- & large-scale mesh-like & other structures. It has resemblance to FDTD schemes in mesh structures & to WDFs for lumped element modeling. Mass–spring networks can be realized systematically also by WDFs using wave variables (Sect. 8).
  - 3.6.3. Modal decomposition methods. In Sect. 6 modal decomposition methods represent another approach to look at vibrating systems, conceptually from a frequency-domain viewpoint. Eigenmodes of a linear system are exponentially decaying sinusoids at eigenfrequencies in response of a system to impulse excitation. Although thinking by modes is normally related to frequency domain, time-domain simulation by modal methods can be relatively efficient, & therefore suitable to discrete-time computation. Modal decomposition methods are inherently based on use of K-variables. Modal synthesis has been applied to make convincing sound synthesis of different musical instruments. Functional transform

method (FTM) is a recent development of systematically exploiting idea of spatially distributed modal behavior, & it has also been extended to nonlinear system modeling.

  - 3.6.4. Digital waveguides. Digital waveguides (DWGs) in Sect. 7 are most popular physics-based method of modeling & synthesizing musical instruments that are based on 1D resonators, e.g. strings & wind instruments. Reason for this is their extreme computational efficiency in their basic formulations. DWGs have been used also in 2D & 3D mesh structures, but in such cases wave-based DWGs are not superior in efficiency. Digital waveguides are based on use of traveling wave components; thus, they form a wave modeling (W-modeling) paradigm [Term digital waveguide is used also to denote K-modeling, e.g. FDTD mesh-structures, & source-filter models derived from traveling wave solutions, which may cause methodological confusion.]. Therefore, they are also compatible with WDFs (Sect. 8), but in order to be compatible with K-modeling techniques, special conversion algorithms must be applied to construct hybrid models, as discussed in Sect. 10.
  - 3.6.5. Wave digital filters. WDFs in Sect. 8 are another wave-based modeling technique, originally developed for discrete-time simulation of analog electric circuits & networks. In their original form, WDFs are best suited for lumped element modeling; thus, they can be easily applied to wave-based mass–spring modeling. Due to their compatibility with digital waveguides, these methods complement each other. WDFs have also been extended to multidimensional networks & to systematic & energetically consistent modeling of nonlinearities. They have been applied particularly to deal with lumped & nonlinear elements in models, where wave propagation parts are typically realized by digital waveguides.
  - 3.6.6. Source–filter models. In Sect. 9 source–filter models form a paradigm between physics-based modeling & signal processing models. True spatial structure & bi-directional interactions are not visible, but are transformed into a transfer function that can be realized as a digital filter. Approach is attractive in sound synthesis because digital filters are optimized to implement transfer functions efficiently. Source part of a source–filter model is often a wavetable, consolidating different physical or synthetic signal components needed to feed filter part. Source–filter paradigm is frequently used in combination with other modeling paradigms in more or less ad hoc ways.

- 4. Finite difference models. Finite difference schemes can be used for solving PDEs, e.g. those describing vibration of a string, a membrane or an air column inside a tube [264]. Key idea in finite difference scheme: replace derivatives with finite difference approximations. An early example of this approach in physical modeling of musical instruments is work done by Hiller & Ruiz in early 1970s [113, 114]. This line of research has been continued & extended by Chaigne & colleagues [45, 46, 48] & recently by others [25, 26, 29, 30, 81, 103, 131].

  Finite difference approach leads to a simulation algorithm that is based on a difference equation, which can be easily programmed with a computer. E.g., how basic wave equation, which describes small-amplitude vibration of a lossless, ideally flexible string, is discretized using this principle. Here present a formulation after Smith [253] using an ideal string as a starting point for discrete-time modeling. A more thorough continuous-time analysis of physics of strings can be found in [96].

  - 4.1. Finite difference models for an ideal vibrating string. Fig. 1: Part of an ideal vibrating string. depicts a snapshot of an ideal (lossless, linear, flexible) vibrating string by showing displacement as a function of position. Wave equation for string is given by $\boxed{Ky'' = \epsilon\ddot{y}}$
  - 4.2. Boundary conditions & string excitation.
  - 4.3. Finite difference approximation of a lossy string.
  - 4.4. Stiffness in finite difference strings.

- 5. Mass-spring networks.
  - 5.1. Basic theory.
  - 5.2. CORDIS-ANIMA.
  - 5.3. Other mass-spring systems.

- 6. Modal decomposition methods.
  - 6.1. Modal synthesis.
  - 6.2. Filter-based modal methods.
  - 6.3. Functional transform method.

- 7. Digital waveguides.
  - 7.1. From wave propagation to digital waveguides.
  - 7.2. Modeling of losses & dispersion.
  - 7.3. Modeling of waveguide termination & scattering.
  - 7.4. Digital waveguide meshes & networks.
  - 7.5. Reduction of a DWG model to a single delay loop structure.
  - 7.6. Commuted DWG synthesis.

- 7.7. Case study: modeling & synthesis of acoustic guitar. Acoustic guitar is an example of a musical instruments for which DWG modeling is found to be an efficient method, especially for real-time sound synthesis [134, 137, 142, 160, 286, 295]. DWG principle in Fig. 14: A DWG block diagram of 2 strings coupled through a common bridge impedance Z_{b} & terminated at other end by nut impedances $Z_{\mathrm{t}1}, Z_{\mathrm{t}2}$. Plucking points are for force insertion from wavetables WT_i into junctions in delay-lines DL_{ij}. Output is taken as bridge velocity. allows for true physically distributed modeling of strings & their interaction, while SDL commuted synthesis (Fig. 17: Reduction of bi-directional delay-line waveguide model (top) to a single delay line loop structure (bottom). & Fig. 18: Principles of commuted DWG synthesis: (a) cascaded excitation, string & body, (b) body & string blocks commuted & (c) excitation & body blocks consolidated into a wavetable for feeding string model.) allows for more efficient computation. In this subsect discuss principles of commuted waveguide synthesis as applied to high-quality synthesis of acoustic guitar.

  There are several features that must be added to simple commuted SDL structure in order to achieve natural sound & control of playing features. Fig. 19: Degrees of freedom for string vibration in guitar: Torsional, Longitudinal, Vertical, Horizontal. depicts degrees of freedom for vibration of strings in guitar. Transversal directions, i.e. vertical & horizontal polarizations of vibration, are most prominent ones. Vertical vibration connects strongly to bridge, resulting in stronger initial sound & faster decay than horizontal vibrations that start more weakly & decay more slowly. Effect of longitudinal vibration is weak but can be observed in generation of some partials of sound [320]. Longitudinal effects are more prominent in piano [16, 58], but are particularly important in such instruments as kantele [82] through nonlinear effect of tension modulation (Sect. 11). Torsional vibration of strings in guitar is not shown to have a remarkable effect on sound. In violin it has a more prominent physical role, although it makes virtually no contribution to sound.

  In commuted waveguide synthesis, 2 transversal polarizations can be realized by 2 separate SDL string models, $S_{\mathrm{v}}(z)$ for vertical & $S_{\mathrm{h}}(z)$ for horizontal polarization in Fig. 20: Dual-polarization string model with sympathetic vibration coupling between strings. Multiple wavetables are used for varying plucking styles. Filter $E(z)$ can control detailed timbre of plucking & $P(z)$ is a plucking point comb filter., each one with slightly different delay & decay parameters. Coefficient m_{p} is used to control relative excitation amplitudes of each polarization, depending on initial direction of string movement after plucking. Coefficients m_{o} can be used to mix vibration signal components at bridge.

  Fig. 20 also shows another inherent feature of guitar, sympathetic coupling between strings at bridge, which causes an undamped string to gain energy from another string set in vibration. While principle shown in Fig. 14 implements this automatically if string & bridge admittances are correctly set, model in Fig. 20 requires special signal connections from point C to vertical polarization model of other strings. This is just a rough approximation of physical phenomenon that guarantees stability of model. There is also a connection through g_{c} that allows for simple coupling from horizontal polarization to excite vertical vibration, with a final result of a coupling between polarizations.

  Dual-polarization model in Fig. 20 is excited by wavetables containing commuted waveguide excitations for different plucking styles. Filter $E(z)$ can be used to control timbre details of selected excitation, & filter $E(z)$ can be used to control timbre details of selected excitation, & filter $P(z)$ is a plucking point comb filter, as prev discussed.

  – Mô hình phân cực kép trong Hình 20 được kích thích bằng các bảng sóng chứa các kích thích ống dẫn sóng chuyển mạch cho các kiểu gảy khác nhau. Bộ lọc $E(z)$ có thể được sử dụng để kiểm soát các chi tiết âm sắc của kích thích đã chọn, & bộ lọc $E(z)$ có thể được sử dụng để kiểm soát các chi tiết âm sắc của kích thích đã chọn, & bộ lọc $P(z)$ là bộ lọc lược điểm gảy, như đã thảo luận trước đó.

  For solid body electric guitars, a magnetic pickup model is needed, but body effect can be neglected. Magnetic pickup can be modeled as a lowpass filter [124,137] in series with a comb filter similar to plucking point filter, but in this case corresponding to pickup position.

  Calibration of model parameters is an important task when simulating a particular instrument. Methods for calibrating a string instrument model are presented, e.g., in [8, 14, 24, 27, 137, 142, 211, 244, 286, 295, 320].

  – Hiệu chuẩn các tham số mô hình là 1 nhiệm vụ quan trọng khi mô phỏng 1 nhạc cụ cụ thể. Các phương pháp hiệu chuẩn mô hình nhạc cụ dây được trình bày ...

  A typical procedure: apply time-frequency analysis to recorded sound of plucked or struck string, in order to estimate decay rate of each harmonic. Parametric models e.g. FZ-ARMA analysis [133, 138] may yield more complete information of modal components in string behavior. This information is used to design a low-order loop filter which approximates frequency-dependent losses in SDL loop structure [14,17,79,244,286]. A recent novel idea has been to design a sparse FIR loop filter, which is of high order but has few nonzero coefficients [163, 209, 293]. This approach offers a computationally efficient way to imitate large deviations in decay rates of harmonic components. Through implementing a slight difference in delays & decay rates of 2 polarizations, beating or 2-stage decay of signal envelope can be approximated. for plucking point comb filter: required to estimate plucking point from a recorded tone [199, 276, 277, 286].

  Fig. 21: Detailed SDL loop structure for string instrument sound synthesis. depicts a detailed structure used in practice to realize SDL loop. Fundamental frequency of string sound is inversely proportional to total delay of loop blocks. Accurate tuning requires application of a fractional delay, because an integral number of unit delays is not accurate enough when a fixed sampling rate is used. Fractional delays are typically approximated by 1st-order allpass filters or 1st- to 5th-order Lagrange interpolators as discussed in [154].

  When loop filter properties are estimated properly, excitation wavetable signal is obtained by inverse filtering (deconvolution) of recorded sound by SDL response. For practical synthesis, only initial transient part of inverse-filtered excitation is used, typically covering several 10s of milliseconds.

  After careful calibration of model, a highly realistic sounding synthesis can be obtained by parametric control & modification of sound features. Synthesis is possible even in cases which are not achievable in practice in real acoustic instruments.

- 7.8. DWG modeling of various musical instruments. Digital waveguide modeling has been applied to a variety of musical instruments other than acoustic guitar. In this subsect, present a brief overview of such models & features that need special attention to each case. For an in-depth presentation on DWG modeling techniques applied to different instrument families, see [254].
  - 7.8.1. Other plucked string instruments.
  - 7.8.2. Struck string instruments.
  - 7.8.3. Bowed string instruments.
  - 7.8.4. Wind instruments.
  - 7.8.5. Percussion instruments.
  - 7.8.6. Speech & singing voice.
  - 7.8.7. Inharmonic SDL type of DWG models.

- 8. Wave digital filters. Purpose of this sect: provide a general overview of physical modeling using WDFs in context of musical instruments. Only essential basics of topic will be discussed in detail; the rest will be glossed over. For more information about project, reader is encouraged to refer to [254]. Also, another definitive work can be found in [94].
  - 8.1. What are wave digital filters? WDFs were developed in late 1960s by Alfred Fettweis [93] for digitizing lumped analog electrical circuits. Traveling-wave formulation of lumped electrical elements, where WDF approach is based, was introduced earlier by Belevitch [21, 254].

    WDFs are certain types of digital filters with valid interpretations in physical world. I.e., can simulate behavior of a lumped physical system (hệ thống vật lý tập trung) using a digital filter whose coefficients depend on parameters of this physical system. Alternatively, WDFs can be seen as a particular type of finite difference schemes with excellent numerical properties [254]. As discussed in Sect. 4, task of finite difference schemes in general: provide discrete versions of PDEs for simulation & analysis purposes.

    WDFs are useful for physical modeling in many respects. 1stly, they are modular: same building blocks can be used for modeling very different systems; all that needs to be changed: topology of wave digital network. 2ndly, preservation of energy & hence also stability is usually addressed, since elementary blocks can be made passive, & energy preservation between blocks are evaluated using Kirchhoff's laws. Finally, WDFs have good numerical properties, i.e., they do not experience artificial damping at high frequencies.

    Physical systems were originally considered to be lumped in basic wave digital formalism. I.e., system to be modeled, say a drum, will become a point-like black box, which has functionality of drum. However, its inner representation, as well as its spatial dimensions, is lost. Must bear in mind, however: question of whether a physical system can be considered lumped depends naturally not only on which of its aspects wish to model but also on frequency scale want to use in modeling (Sect. 3).
  - 8.2. Analog circuit theory.
  - 8.3. Wave digital building blocks.
  - 8.4. Interconnection & adaptors.
  - 8.5. Physical modeling using WDFs.
  - 8.6. Current research.

- 9. Source-filter models.
  - 9.1. Subtractive synthesis in computer music.
  - 9.2. Source-filter models in speech synthesis.
  - 9.3. Instrument body modeling by digital filters.
  - 9.4. Karplus–Strong algorithm.
  - 9.5. Virtual analog synthesis.

- 10. Hybrid models.
  - 10.1. KW-hybrids.
  - 10.2. KW-hybrids modeling examples.

- 11. Modeling of nonlinear & time-varying phenomena.
  - 11.1. Modeling of nonlinearities in musical instruments.
  - 11.2. Case study: nonlinear string model using generalized time-varying allpass filters.
  - 11.3. Modeling of time-varying phenomena.

- 12. Current trends & further research.

- 13. Conclusions.

2 Librosa

`librosa` is a Python package for music & audio analysis. In provides building blocks necessary to create music information retrieval systems.

For a quick introduction to using librosa, see Tutorial. For a more advanced introduction which describes package design principles, refer to librosa paper at SciPy 2015.

2.1 Brian McFee, Colin Raffel, Dawen Liang, Daniel P. W. Ellis, Matt McVicar, Eric Battenberg, Oriol Nieto. librosa: Audio & Music Signal Analysis in Python

[3562 citations]

- Abstract. This document describes version 0.4.0 of `librosa`: a Python package for audio & music signal processing. At a high level, `librosa` provides implementations of a variety of common functions used throughout field of music information retrieval. In this document, a brief overview of library's functionality is provided, along with explanations of design goals, software development practices, & notational conventions.

- Index terms. audio, music, signal processing.

- Introduction. Emerging research field of music information retrieval (MIR) broadly covers topics at intersection of musicology, digital signal processing, ML, information retrieval, & library science. Although field is relatively young – 1st international symposium on music information retrieval (ISMIR) `http://ismir.net` was held in Oct of 2000 – it is rapidly developing, thanks in part to proliferation & practical scientific needs of digital music services, e.g. iTunes, Pandora, & Spotify. While preponderance of MIR research has been conducted with custom tools & scripts developed by researchers in a variety of languages e.g. MATLAB or C++, stability, scalability, & ease of use these tools has often left much to be desired.

  – Lĩnh vực nghiên cứu mới nổi về truy xuất thông tin âm nhạc (MIR) bao gồm rộng rãi các chủ đề giao thoa giữa âm nhạc học, xử lý tín hiệu số, ML, truy xuất thông tin, & khoa học thư viện. Mặc dù lĩnh vực này còn khá mới mẻ – hội nghị chuyên đề quốc tế đầu tiên về truy xuất thông tin âm nhạc (ISMIR) `http://ismir.net` đã được tổ chức vào tháng 10 năm 2000 – nhưng nó đang phát triển nhanh chóng, 1 phần là nhờ sự gia tăng & nhu cầu khoa học thực tế của các dịch vụ âm nhạc số, ví dụ như iTunes, Pandora, & Spotify. Trong khi phần lớn nghiên cứu MIR được tiến hành bằng các công cụ tùy chỉnh & tập lệnh do các nhà nghiên cứu phát triển bằng nhiều ngôn ngữ khác nhau, ví dụ như MATLAB hoặc C++, tính ổn định, khả năng mở rộng, & dễ sử dụng của các công cụ này thường không được như mong đợi.

  In recent years, interest has grown within MIR community in using (scientific) Python as a viable alternative. This has been driven by a confluence of several factors, including availability of high-quality ML libraries e.g. `scikit-learn` [Pedregosa11] & tools based on `Theano` [Bergstra11], as well as Python's vast catalog of packages for dealing with text data & web services. However, adoption of Python has been slowed by absence of a stable core library that provides basic routines upon which many MIR applications are built. To remedy this situation, have developed librosa: `https://github.com/bmcfee/librosa` a Python package for audio & music signal processing. [Name `librosa` is borrowed from *LabROSA*: LABoratory for Recognition & Organization of Speech & Audio at Columbia University, where initial development of librosa took place.] In doing so, hope to both ease transition of MIR researchers into Python (& modern software development practices), & also to make core MIR techniques readily available to broader community of scientists & Python programmers.

  - Design principles. In designing librosa, have prioritized a few key concepts. 1st, strive for a low barrier to entry for researchers familiar with MATLAB. In particular, opted for a relatively flat package layout, & following `scipy` [Jones01] rely upon `numpy` data types & functions [VanDerWalt11], rather than abstract class hierarchies.

    2nd, expended considerable effort in standardizing interfaces, variable names, & (default) parameter settings across various analysis functions. This task was complicated by fact: reference implementations from which our implementations are derived come from various authors, & are often designed as 1-off scripts rather than proper library functions with well-defined interfaces.

    – Thứ 2, dành nhiều công sức để chuẩn hóa giao diện, tên biến, cài đặt tham số & (mặc định) trên nhiều hàm phân tích khác nhau. Nhiệm vụ này phức tạp hơn vì thực tế: các triển khai tham chiếu mà các triển khai của chúng tôi bắt nguồn từ nhiều tác giả khác nhau, & thường được thiết kế dưới dạng tập lệnh 1 lần thay vì các hàm thư viện phù hợp với giao diện được xác định rõ ràng.

    3rd, wherever possible, retain backwards compatibility against existing reference implementations. This is achieved via regression testing for numerical equivalence of outputs. All tests are implemented in `nose` framework `https://nose.readthedocs.org/en/latest/`.

    – Thứ 3, bất cứ khi nào có thể, hãy duy trì khả năng tương thích ngược với các triển khai tham chiếu hiện có. Điều này đạt được thông qua thử nghiệm hồi quy để có sự tương đương về mặt số của đầu ra. Tất cả các thử nghiệm đều được triển khai trong `nose`.

    4th, because MIR is a rapidly evolving field, recognize: exact implementations provided by librosa may not represent state of art for any particular task. Consequently, functions are designed to be *modular*, allowing practitioners to provide their own functions when appropriate, e.g., a custom onset strength estimate may be provided to beat tracker as a function argument. This allows researchers to leverage existing library functions while experimenting with improvements to specific components.

Although this seems simple & obvious, from a practical standpoint monolithic designs & lack of interoperability between different research codebases have historically made this difficult.

– Thứ 4, vì MIR là 1 lĩnh vực phát triển nhanh chóng, hãy nhận ra: các triển khai chính xác do librosa cung cấp có thể không đại diện cho trạng thái nghệ thuật cho bất kỳ nhiệm vụ cụ thể nào. Do đó, các hàm được thiết kế theo dạng *mô-đun*, cho phép các học viên cung cấp các hàm của riêng họ khi thích hợp, ví dụ, có thể cung cấp ước tính cường độ khởi phát tùy chỉnh cho beat tracker dưới dạng đối số hàm. Điều này cho phép các nhà nghiên cứu tận dụng các hàm thư viện hiện có trong khi thử nghiệm các cải tiến đối với các thành phần cụ thể. Mặc dù điều này có vẻ đơn giản & hiển nhiên, nhưng theo quan điểm thực tế, các thiết kế nguyên khối & thiếu khả năng tương tác giữa các cơ sở mã nghiên cứu khác nhau trong lịch sử đã khiến điều này trở nên khó khăn.

Finally, strive for readable code, thorough documentation & exhaustive testing. All development is conducted on GitHub. Apply modern software development practices, e.g. continuous integration testing (via Travis https://travis-ci.org) & coverage(via Coveralls https://coveralls.io). All functions are implemented in pure Python, thoroughly documented using Sphinx, & include example code demonstrating usage. Implementation mostly complies with PEP-8 recommendations, with a small set of exceptions for variable names that make code more concise without sacrificing clarity: e.g., `y, sr` are preferred over more verbose names e.g. `audio_buffer, sampling_rate`.

– Cuối cùng, hãy cố gắng tạo ra mã dễ đọc, tài liệu hướng dẫn đầy đủ & thử nghiệm toàn diện. Mọi hoạt động phát triển đều được thực hiện trên GitHub. Áp dụng các phương pháp phát triển phần mềm hiện đại, ví dụ như thử nghiệm tích hợp liên tục (qua Travis https://travis-ci.org) & phạm vi phủ sóng (qua Coveralls https://coveralls.io). Tất cả các chức năng đều được triển khai bằng Python thuần túy, được ghi chép đầy đủ bằng Sphinx, & bao gồm mã ví dụ minh họa cách sử dụng. Việc triển khai phần lớn tuân thủ các khuyến nghị của PEP-8, với 1 tập hợp nhỏ các ngoại lệ cho tên biến giúp mã ngắn gọn hơn mà không làm mất đi tính rõ ràng: ví dụ: `y, sr` được ưu tiên hơn các tên dài dòng hơn, ví dụ: `audio_buffer, sampling_rate`.

- Conventions. In general, librosa's functions tend to expose all relevant parameters to caller. While this provides a great deal of flexibility to expert users, it can be overwhelming to novice users who simply need a consistent interface to process audio files. To satisfy both needs, define a set of general conventions & standardized default parameter values shared across many functions.

  An audio signal is represented as a 1D `numpy` array, denoted as `y` throughout librosa. Typically signal `y` is accompanied by *sampling rate* (denoted `sr`) which denotes frequency (in Hz) at which values of `y` are sampled. Duration of a signal can then be computed by dividing number of samples by sampling rate:

  ```
  >>> duration_seconds = float(len(y)) / sr
  >>> duration_seconds = float(len(audio_buffer)) / sampling_rate
  ```

  By default, when loading stereo audio files, `librosa.load()` function downmixes to mono by averaging left- & right-channels, & then resamples monophonic signal to default rate `sr = 22050` Hz.

  Most audio analysis methods operate not at native sampling rate of signal, but over small *frames* of signal which are spaced by a *hop length* (in samples). Default frame & hop lengths are set to 2048 & 512 samples, resp. At default sampling rate of 22050 Hz, this corresponds to overlapping frames of ≈ 93 ms spaced by 23 ms. Frames are centered by default, so frame index `t` corresponds to slice:

  ```
  y[(t * hop_length - frame_length / 2):
  (t * hop_length + frame_length / 2)],
  ```

  where boundary conditions are handled by reflection-padding input signal y. Unless otherwise specified, all sliding-window analyses use Hann windows by default. For analyses that do not use fixed-width frames (e.g. constant-Q transform), default hop length of 512 is retained to facilitate alignment of results.

  Majority of feature analyses implemented by librosa produce 2D outputs stored as `numpy.ndarray`, e.g., `S[f, t]` might contain energy within a particular frequency band `f` at frame index `t`. Follow convention: final dimension provides index over time, e.g., `S[:, 0], S[:, 1]` access features at 1st & 2nd frames. Feature arrays are organized column-major (Fortran style) in memory, so that common access patterns benefit from cache locality.

  By default, all pitch-based analyses are assumed to be relative to a 12-bin equal-tempered chromatic scale with a reference tuning of `A440 = 440.0 Hz`. Pitch & pitch-class analyses are arranged s.t. 0th bin corresponds to `C` for pitch class or `C1` (32.7 Hz) for absolute pitch measurements.

- Package organization. In this sect, give a brief overview of structure of librosa software package. This overview is intended to be superficial & cover only most commonly used functionality. A complete API reference can be found.

  - *Core functionality.* `librosa.core` submodule includes a range of commonly used functions. Broadly, `core` functionality falls into 4 categories: audio & time-series operations, spectrogram calculation, time & frequency conversion, & pitch operations. For convenience, all functions within `core` submodule are aliased at top level of package hierarchy, e.g., `librosa.core.load` is aliased to `librosa.load`.

    Audio & time-series operations include functions e.g.: reading audio from disk via `audioread` package https://github.com/sampsyo/audioread `core.load`, resampling a signal at a desired rate `core.resample`, stereo to mono conversion `core.to_mono`, time-domain bounded auto-correlation `core.autocorrelate`, & 0-crossing detection `core.zero_crossings`.

Spectrogram operations include short-time Fourier transform `stft`, inverse STFT `istft`, & instantaneous frequency spectrogram `ifgram` [Abe95], which provide much of core functionality for down-stream feature analysis. Additionally, an efficient constant-Q transform `cqt` implementation based upon recursive down-sampling method of Schokerkhuber & Klapuri [Schoerkhuber10] is provided, which provides logarithmically-spaced frequency representations suitable for pitch-based signal analysis. Finally, `logamplitude` provides a flexible & robust implementation of log-amplitude scaling, which can be used to avoid numerical underflow & set an adaptive noise floor when converting from linear amplitude.

– Các hoạt động phổ đồ bao gồm biến đổi Fourier thời gian ngắn `stft`, STFT nghịch đảo `istft`, phổ đồ tần số tức thời `ifgram` [Abe95], cung cấp nhiều chức năng cốt lõi cho phân tích tính năng hạ lưu. Ngoài ra, 1 triển khai biến đổi Q hằng số `cqt` hiệu quả dựa trên phương pháp lấy mẫu xuống đệ quy của Schokerkhuber & Klapuri [Schoerkhuber10] được cung cấp, cung cấp các biểu diễn tần số cách đều theo logarit phù hợp cho phân tích tín hiệu dựa trên cao độ. Cuối cùng, `logamplitude` cung cấp 1 triển khai linh hoạt & mạnh mẽ của tỷ lệ biên độ logarit, có thể được sử dụng để tránh tràn số & thiết lập sàn nhiễu thích ứng khi chuyển đổi từ biên độ tuyến tính.

Because data may be represented in a variety of time or frequency units, provide a comprehensive set of convenience functions to map between different time representations: secs, frames, or samples; & frequency representations: hertz, constant-Q basis index, Fourier basis index, Mel basis index, MIDI note number, or note in scientific pitch notation.

Finally, core submodule provides functionality to estimate dominant frequency of STFT bins via parabolic interpolation `piptrack` [Smith11], & estimation of tuning deviation (in cents) from reference `A440`. These functions allow pitch-based analyses (e.g., `cqt`) to dynamically adapt filter banks to match global tuning offset of a particular audio signal.

- Spectral features. Spectral representations – distributions of energy over a set of frequencies – form basis of many analysis techniques in MIR & digital signal processing in general. `librosa.feature` module implements a variety of spectral representations, most of which are based upon short-time Fourier transform.

  Mel frequency scale is commonly used to represent audio signals, as it provides a rough model of human frequency perception [Stevens37]. Both a Mel-scale spectrogram `librosa.feature.melspectrogram` & commonly used Mel-frequency Cepstral Coefficients (MFCC) `librosa.feature.mfcc` are provided. By default, Mel scales are defined to match implementation provided by Slaney's auditory toolbox [Slaney98], but they can be made to match Hidden Markov Model Toolkit (HTK) by setting flag `htk = True` [Young97].

  While Mel-scaled representations are commonly used to capture timbral aspects of music, they provide poor resolution of pitches & pitch classes. Pitch class (or *chroma*) representations are often used to encode harmony while suppressing variations in octave height, loudness, or timbre. 2 flexible chroma implementations are provided: one uses a fixed-window STFT analysis `chroma_stft` [`chroma_stft` is based upon reference implementation provided at http://labrosa.ee.columbia.edu/matlab/chroma-ansyn/] (dead link) & other uses variable-window constant-Q transform analysis `chroma_cqt`. An alternative representation of pitch & harmony can be obtained by `tonnetz` function, which estimates tonal centroids as coordinates in a 6D interval space using method of Harte et al. [Harte06]. Fig. 1: 1st: short-time Fourier transform of a 20-sec audio clip `librosa.stft`. 2nd: corresponding Mel spectrogram, using 128 Mel bands `librosa.feature.melspectrogram`. 3rd: corresponding chromagram `librosa.feature.chroma_cqt`. 4: Tonnetz features `librosa.feature.tonnetz` illustrates difference between STFT, Mel spectrogram, chromagram, & Tonnetz representations, as constructed by following code fragment: [For display purposes, spectrograms are scaled by `librosa.logamplitude`. Refer readers to accompanying IPython notebook for full source code to reconstruct figures.]

```
>>> filename = librosa.util.example_audio_file()
>>> y, sr = librosa.load(filename, offset=25.0, duration=20.0)
>>> spectrogram = np.abs(librosa.stft(y))
>>> melspec = librosa.feature.melspectrogram(y=y, sr=sr)
>>> chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
>>> tonnetz = librosa.feature.tonnetz(y=y, sr=sr)
```

  In addition to Mel & chroma features, `feature` submodule provides a number of spectral statistic representations, including `spectral_centroid, spectral_bandwidth, spectral_rolloff` [Klapuri07], & `spectral_contrast` [Jiang02]. [`spectral_*` functions are derived from MATLAB reference implementations provided by METLab at Drexel University http://music.ece.drexel.edu/.]

  Finally, `feature` submodule provides a new functions to implement common transformations of time-series features in MIR. This includes `delta`, which provides a smoothed estimate of time derivative; `stack_memory`, which concatenates an input feature array with time-lagged copies of itself (effectively simulating feature n-grams); & `sync`, which applies a user-supplied aggregation function, e.g., `numpy.mean` or `median`, across specified column intervals.

- Display. `display` module provides simple interfaces to visually render audio data through `matplotlib` [Hunter07]. 1st function, `display.waveplot` simply renders amplitude envelope of an audio signal `y` using matplotlib's `fill_between` function. For efficiency purposes, signal is dynamically down-sampled. Mono signals are rendered symmetrically about horizontal axis; stereo signals are rendered with left-channel's amplitude above axis & right-channel's below. An example of `waveplot` is depicted in Fig. 2: Top: a waveform plot for a 20-sec audio clip `y`, generated by `librosa.display.waveplot`. Middle: log-power short-time Fourier transform (STFT) spectrum for `y` plotted on a logarithmic frequency scale, generated by `librosa.display.specshow`. Bottom: onset strength function `librosa.onset.onset_strength`, detected onset events `librosa.onset.onset_detect`, & detected beat events `librosa.beat.beat_track` for `y`.

2nd function, `display.specshow` wraps matplotlib's `imshow` function with default settings (`origin, aspect`) adapted to expected defaults for visualizing spectrograms. Additionally, `specshow` dynamically selects appropriate colormaps (binary, sequential, or diverging) from data type & range. [If `seaborn` package [Waskom14] is available, its version of *cubehelix* is used for sequential data.] Finally, `specshow` provides a variety of acoustically relevant axis labeling & scaling parameters. Examples of `specshow` output are displayed in Figs. 1–2 (middle).

- Onsets, tempo, & beats. While spectral feature representations described above capture frequency information, time information is equally important for many applications in MIR. E.g., it can be beneficial to analyze signals indexed by note or beat events, rather than absolute time. `onset, beat` submodules implement functions to estimate various aspects of timing in music.

  More specially, `onset` module provides 2 functions: `onset_strength, onset_detect`. `onset_strength` function calculates a thresholded spectral flux operation over a spectrogram, & returns a 1D array representing amount of increasing spectral energy at each frame. This is illustrated as blue curve in bottom panel of Fig. 2. `onset_detect` function, on other hand, selects peak positions from onset strength curve following heuristic described by Boeck et al. [Boeck12]. Output of `onset_detect` is depicted as red circles in bottom panel of Fig. 2.

  – Cụ thể hơn, mô-đun `starts` cung cấp 2 hàm: `onset_strength, starts_detect`. Hàm `onset_strength` tính toán 1 phép toán thông lượng phổ ngưỡng trên 1 phổ đồ, & trả về 1 mảng 1D biểu diễn lượng năng lượng phổ tăng dần tại mỗi khung. Điều này được minh họa bằng đường cong màu xanh lam ở bảng dưới cùng của Hình 2. Mặt khác, hàm `onset_detect` chọn các vị trí đỉnh từ đường cong cường độ khởi đầu theo phương pháp tìm kiếm được mô tả bởi Boeck et al. [Boeck12]. Đầu ra của `onset_detect` được mô tả bằng các vòng tròn màu đỏ ở bảng dưới cùng của Hình 2.

  `beat` module provides functions to estimate global tempo & positions of beat events from onset strength function, using method of Ellis [Ellis07]. More specifically, beat tracker 1st estimates tempo, which is then used to set target spacing between peaks in an onset strength function. Output of beat tracker is displayed as dashed green lines in Fig. 2 (bottom).

  Typing this all together, tempo & beat positions for an input signal can be easily calculated by following code fragment:

```
>>> y, sr = librosa.load(FILENAME)
>>> tempo, frames = librosa.beat.beat_track(y=y, sr=sr)
>>> beat_times = librosa.frames_to_time(frames, sr=sr)
```

  Any of default parameters & analyzes may be overridden. E.g., if user has calculated an onset strength envelope by some other means, it can be provided to beat tracker as follows:

```
>>> oenv = some_other_onset_function(y, sr)
>>> librosa.beat.beat_track(onset_envelope=oenv)
```

  All detection functions (beat & onset) return events as frame indices, rather than absolute timing. Downside of this: left to user to convert frame indices back to absolute time. However, in our opinion, this is outweighed by 2 practical benefits: it simplifies implementations, & it makes results directly accessible to frame-indexed functions e.g. `librosa.feature.sync`.

- Structural analysis. Onsets & beats provide relatively low-level timing cues for music signal processing. Higher-level analyses attempt to detect larger structure in music, e.g., at level of bars or functional components e.g. *verse, chorus*. While this is an active area of research that has seen rapid progress in recent years, there are some useful features common to many approaches. `segment` submodule contains a few useful functions to facilitate structural analysis in music, falling broadly into 2 categories.

  – Phân tích cấu trúc. Các nhịp khởi đầu & cung cấp tín hiệu thời gian ở mức tương đối thấp để xử lý tín hiệu âm nhạc. Các phân tích ở mức cao hơn cố gắng phát hiện cấu trúc lớn hơn trong âm nhạc, ví dụ, ở mức ô nhịp hoặc các thành phần chức năng ví dụ *verse, chorus*. Mặc dù đây là 1 lĩnh vực nghiên cứu tích cực đã chứng kiến sự tiến bộ nhanh chóng trong những năm gần đây, nhưng có 1 số tính năng hữu ích chung cho nhiều phương pháp tiếp cận. Mô-đun con `segment` chứa 1 số hàm hữu ích để tạo điều kiện thuận lợi cho phân tích cấu trúc trong âm nhạc, về cơ bản được chia thành 2 loại.

  1st, there are functions to calculate & manipulate *recurrence* or *self-similarity* plots. `segment.recurrence_matrix` constructs a binary k-nearest-neighbor similarity matrix from a given feature array & a user-specified distance function. As displayed in Fig. 3: left: recurrence plot derived from chroma features displayed in Fig. 1., repeating sequences often appear as diagonal bands in recurrence plot, which can be used to detect musical structure. Sometimes more convenient to operate in *time-lag* coordinates, rather than *time-time*, which transforms diagonal structures into more easily detectable horizontal structure Fig. 3: right: corresponding time-lag plot. [Serra12]. This is facilitated by `recurrence_to_lag` (& `lag_to_recurrence`) functions.

  2nd, temporally constrained clustering can be used to detect feature change-points without relying upon repetition. This is implemented in librosa by `segment.agglomerative` function, which uses `scikit-learn`'s implementation of Ward's agglomerative clustering method [Ward63] to partition input into a user-defined number of contiguous components. In practice, a user can override default clustering parameters by providing an existing `sklearn.cluster.AgglomerativeClustering` object as an argument to `segment.agglomerative()`.

- Decompositions. Many applications in MIR operate upon latent factor representations, or other decompositions of spectrograms. E.g., common to apply nonnegative matrix factorization (NMF) [Lee99] to magnitude spectra, & analyze statistics of resulting time-varying activation functions, rather than raw observations.

  `decompose` module provides a simple interface to factor spectrograms (or general feature arrays) into *components & activations*:

```
>>> comps, acts = librosa.decompose.decompose(S)
```

By default, `decompose()` function constructs a `scikit-learn` NMF object, & applies its `fit_transform()` method to transpose of S. Resulting basis components & activations are accordingly transposed, so that `comps.dot(acts)` approximates S. If user wishes to apply some other decomposition technique, any object fitting `sklearn.decomposition` interface may be substituted:

```
>>> T = SomeDecomposer()
>>> librosa.decompose.decompose(S, transformer=T)
```

In addition to general-purpose matrix decomposition techniques, librosa also implements harmonic-percussion source separation (HPSS) method of Fitzgerald [Fitzgerald10] as `decompose.hpss`. This technique is commonly used in MIR to suppress transients when analyzing pitch content, or suppress stationary signals when detecting onsets or other rhythmic elements. An example application of HPSS is illustrated in Fig. 4: Top: separated harmonic & percussive waveforms. Middle: Mel spectrogram of harmonic component. Bottom: Mel spectrogram of percussive component.

- Effects. `effects` module provides convenience functions to applying spectrogram-based transformations to time-domain signals. E.g., rather than writing

```
>>> D = librosa.stft(y)
>>> Dh, Dp = librosa.decompose.hpss(D)
>>> y_harmonic = librosa.istft(Dh)
```

one may simply write

```
>>> y_harmonic = librosa.effects.harmonic(y)
```

Convenience functions are provided for HPSS (retaining harmonic, percussive, or both components), time-stretching & pitch shifting. Although these functions provide no additional functionality, their inclusion results in simpler, more readable application code.

- Output. `output` module includes utility functions to save results of audio analysis to disk. Most often, this takes form of annotated instantaneous event timings or time intervals, which are saved in plain text (comma- or tab-separated values) via `output.times_csv, output.annotation`, resp. These functions are somewhat redundant with alternative functions for text output (e.g., `numpy.savetxt`), but provide sanity checks for length agreement & semantic validation of time intervals. Resulting outputs are designed to work with other common MIR tools, e.g. `mir_eval` [Raffel14] & `sonic-visualiser` [Cannam10].

  `output` module also provides `wirte_wav` function for saving audio in `.wave` format. `write_wav` simply wraps built-in `scipy` wave-file writer `scipy.io.wavfile.write` with validation & optional normalization, thus ensuring: resulting audio files are well-formed.

- Caching. MIR applications typically require computing a variety of features (e.g., MFCCs, chroma, beat timings, etc.) from each audio signal in a collection. Assuming application programmer is content with default parameters, simplest way to achieve this: call each function using audio time-series input, e.g.:

```
>>> mfcc = librosa.feature.mfcc(y=y, sr=sr)
>>> tempo, beats = librosa.beat.beat_track(y=y, sr=sr)
```

However, because there are shared computations between different functions – `mfcc` & `beat_track` both compute log-scaled Mel spectrograms, e.g. – this results in redundant (& inefficient) computation. A more efficient implementation of above example would factor out redundant features:

```
>>> lms = librosa.logamplitude(librosa.feature.melspectrogram(y=y, sr=sr))
>>> mfcc = librosa.feature.mfcc(S=lms)
>>> tempo, beats = librosa.beat.beat_track(S=lms, sr=sr)
```

Although it is more computationally efficient, above example is less concise, & it requires more knowledge of implementations on behalf of application programmer. More generally, nearly all functions in librosa eventually depend upon STFT calculation, but rare: application programmer will need STFT matrix as an end-result.

1 approach to eliminate redundant computation: decompose various functions into blocks which can be arranged in a computation graph, as is done in Essentia [Bogdanov13]. However, this approach necessarily constrains function interfaces, & may become unwieldy for common, simple applications.

Instead, librosa takes a lazy approach to eliminating redundancy via *output caching*. Caching is implemented through an extension of `Memory` class from `joblib` package https://github.com/joblib/joblib, which provides disk-backed memoization of function outputs. Cache object `librosa.cache` operates as a decorator on all non-trivial computations. This way, a user can write simple application code (i.e., 1st example above) while transparently eliminating redundancies & achieving speed comparable to more advanced implementation (2nd example).

Cache object is disabled by default, but can be activated by setting environment variable `LIBROSA_CACHE_DIR` prior to importing package. Because `Memory` object does not implement a cache eviction policy (as of version 0.8.4), recommended: users purge cache after processing each audio file to prevent cache from filling all available disk space [Cache can be purged by calling `librosa.cache.clear()`.] Note: this can potentially introduce race conditions in multi-processing environments (i.e., parallel batch processing of a corpus), so care must be taken when scheduling cache purges.

– Đối tượng bộ nhớ đệm bị vô hiệu hóa theo mặc định, nhưng có thể được kích hoạt bằng cách thiết lập biến môi trường `LIBROSA_CACHE_DIR` trước khi nhập gói. Vì đối tượng `Memory` không triển khai chính sách xóa bộ nhớ đệm (kể từ phiên bản 0.8.4), nên khuyến nghị: người dùng xóa bộ nhớ đệm sau khi xử lý từng tệp âm thanh để ngăn bộ nhớ đệm lấp đầy toàn bộ dung lượng đĩa khả dụng [Bộ nhớ đệm có thể được xóa bằng cách gọi `librosa.cache.clear()`.] Lưu ý: điều này có khả năng gây ra tình trạng chạy đua trong môi trường đa xử lý (tức là xử lý hàng loạt song song của 1 ngữ liệu), do đó phải cẩn thận khi lên lịch xóa bộ nhớ đệm.

- Parameter tuning. Some of librosa's functions have parameters that require some degree of tuning to optimize performance. In particular, performance of beat tracker & onset detection functions can vary substantially with small changes in certain key parameters.

  After standardizing certain default parameters – sampling rate, frame length, & hop length – across all functions, optimized beat tracker settings using parameter grid given in Table 1: Parameter grid for beat tracking optimization. Best configuration is indicated in bold. To select best-performing configuration, evaluated performance on a data set comprised of Isophonics Beatles corpus http://isophonics.net/content/reference-annotations & SMC Dataset 2 [Holzapfel12] beat annotations. Each configuration was evaluated using `mir_eval` [Raffel14], & configuration was chosen to maximize Correct Metric Level (Total) metric [Davies14].

  Similarly, onset detection parameters (listed in Table 2: Parameter grid for onest detection optimization. Best configuration is indicated in bold.) were selected to optimize F1-score on Johannes Kepler University onset database https://github.com/CPJKU/onset_db.

  Note: "optimal" default parameter settings are merely estimates, & depend upon datasets over which they are selected. Parameter settings are therefore subject to change in future as larger reference collections become available. Optimization framework has been factored out into a separate repository, which may in subsequent versions grow to include additional parameters. https://github.com/bmcfee/librosa_parameters

- Conclusion. This document provides a brief summary of design considerations & functionality of librosa. More detailed examples, notebooks, & documentation can be found in development repository & project website. Project is under active development, & roadmap for future work includes efficiency improvements & enhanced functionality of audio coding & file system interactions.

3 Wikipedia

3.1 Wikipedia/computer music

"*Computer music* is application of computing technology in music composition, to help human composers create new music or to have computers independently create music, e.g. with algorithmic composition programs. it includes theory & application of new & existing computer software technologies & basic aspects of music, e.g. sound synthesis, digital signal processing, sound design, sonic diffusion, acoustics, electrical engineering, & psychoacoustics. Field of computer music can trace its roots back to origins of electric music, & 1st experiments & innovations with electronic instruments at turn of 20th century.

3.1.1 History

3.1.2 Advances

3.1.3 Research

3.1.4 Machine improvisation

3.1.5 Live coding

" – Wikipedia/computer music

3.2 Wikipedia/octave

"A perfect octave between 2 Cs. In music, an *octave* (Latin: octavus: 8th) or *perfect octave* (sometimes called diapason) is an interval between 2 notes, one having twice frequency of vibration of the other. Octave relationship is a natural phenomenon that has been referred to as "basic miracle of music", use of which is "common in most musical systems". Interval between 1st & 2nd

harmonics of harmonic series in an octave. In Western music notation, notes separated by an octave (or multiple octaves) have same name & are of same pitch class.

To emphasize: 1 of perfect intervals (including unison, perfect 4th, & perfect 5th), octave is designated P8. Other interval qualities are also possible, though rare. Octave above or below an indicated note is sometimes abbreviated 8^a or 8^{va} (Italian: all'ottava), 8^{va} bassa (Italian: all'ottava bassa, sometimes also 8^{vb}), or simply 8 for octave in direction indicated by placing this mark above or below staff.

3.2.1 Explanation & definition

An octave is interval between 1 musical pitch & another with doubel or half its frequency. E.g., if 1 note has a frequency of 440 Hz, note 1 octave above is at 880 Hz, & note 1 octave below is at 220 H. Ratio of frequencies of 2 notes an octave apart is therefore 2:1. Further octaves of a note occur at 2^n times frequency of that note (where $n \in \mathbb{Z}$), e.g., 2, 4, 8, 16, etc. & reciprocal of that series. E.g., 55 Hz & 440 Hz are 1 & 2 octaves away from 110 Hz because they are $\frac{1}{2} = 2^{-1}$ & $4 = 2^2$ times frequency, resp.

Number of octaves between 2 frequencies is given by formula: Number of octaves $= \log_2 \frac{f_2}{f_1}$. Oscillogram of middle C (261.62 Hz). (Scale: 1 square is equal to 1 millisecond). C5, an octave above middle C. The frequency is twice that of middle C (523.25 Hz). C3, an octave below middle C. The frequency is half that of middle C (130.81 Hz).

3.2.2 Music theory

Most musical scales are written so that they begin & end on notes that are octave apart. E.g., C major scale is typically written C D E F G A B C, initial & final Cs being an octave apart.

– Hầu hết âm nhạc được viết sao cho chúng bắt đầu & kết thúc bằng các nốt cách nhau 1 quãng tám. E.g., âm giai C trưởng thường được viết là C D E F G A B C, nốt C đầu & cuối cách nhau 1 quãng tám.

Because of octave equivalence, notes in a chord that are 1 or more octaves apart are said to be *doubled* (even if there are > 2 notes in different octaves) in chord. Word is also used to describe melodies played in parallel 1 or more octaves apart (see example under Equivalence, below).

– Do sự tương đương quãng tám, các nốt trong 1 hợp âm cách nhau 1 hoặc nhiều quãng tám được gọi là *được nhân đôi* (kể cả khi có > 2 nốt trong các quãng tám khác nhau) trong hợp âm. Từ này cũng được dùng để mô tả giai điệu được chơi trong parallel cách nhau 1 hoặc nhiều quãng tám (xem ví dụ trong mục Tương đương bên dưới).

While octaves commonly refer to perfect octave (P8), interval of an octave in music theory encompasses chromatic alterations within pitch class, i.e., G♮ to G♯ (13 semitones higher) is an Augmented octave (A8), & G♮ to G♭ (11 semitones higher) is a diminished octave (d8). Use of such intervals is rare, as there is frequently a preferable enharmonically-equivalent notation available (minor 9th & major 7th resp.), but these categories of octaves must be acknowledged in any full understanding of role & meaning of octaves more generally in music.

– Trong khi quãng tám thường đề cập đến quãng tám hoàn hảo (P8), khoảng cách của 1 quãng tám trong lý thuyết âm nhạc bao gồm các thay đổi về sắc độ trong lớp cao độ, tức là, G♮ đến G♯ (cao hơn 13 nửa cung) là 1 Quãng tám tăng (A8), & G♮ đến G♭ (cao hơn 11 nửa cung) là 1 quãng tám giảm (d8). Việc sử dụng các khoảng như vậy rất hiếm, vì thường có 1 ký hiệu tương đương enharmonically thích hợp hơn (minor 9th & major 7th tương ứng), nhưng các loại quãng tám này phải được thừa nhận trong bất kỳ sự hiểu biết đầy đủ nào về vai trò & ý nghĩa của quãng tám nói chung trong âm nhạc.

3.2.3 Notation

- **Octave of a pitch.** Octaves are identified with various naming systems. Among most common are scientific, Helmholtz, organ pipe, & MIDI note systems. In scientific pitch notation, a specific octave is indicated by a numerical subscript number after note name. In this notation, middle C is C_4, because of note's position as 4th C key on a standard 88-key piano keyboard, while C an octave higher is C_5.

  – Các quãng tám được xác định bằng nhiều hệ thống đặt tên khác nhau. Trong số những hệ thống phổ biến nhất là scientific, Helmholtz, organ pipe, & hệ thống nốt MIDI. Trong ký hiệu cao độ khoa học, 1 quãng tám cụ thể được chỉ ra bằng 1 số chỉ số dưới dạng số sau tên nốt. Trong ký hiệu này, middle C là C_4, vì vị trí của nốt là phím C thứ 4 trên bàn phím piano 88 phím tiêu chuẩn, trong khi C cao hơn 1 quãng tám là C_5. An 88-key piano, with octaves numbered & Middle C (turquoise) & A440 (yellow) highlighted.

- **Ottava alta & bassa.** Example of same 3 notes expressed in 3 ways: (1) regularly, (2) in an 8^{va} bracket, (3) in a 15^{ma} bracket. Similar example with $8^{vb}, 15^{mb}$. Notation 8^a or 8^{va} is sometimes seen in sheet music, meaning "play this an octave higher than written" (all' ottava: "at octave" or all' 8^{va}). 8^a or 8^{va} stands for *ottava*, Italian word for octave (or "8th"); octave above may be specified as *ottava alta* or *ottava sopra*. Sometimes 8^{va} is used to tell musician to play a passage an octave *lower* (when placed under rather than over staff), though similar notation 8^{vb} (ottava bassa or ottava sotto) is also used. Similarly, 15^{ma} (quindicesima) means "play 2 octaves higher than written" & 15^{mb} (quindicesima bassa) means "play 2 octaves lower than written".

  Abbreviations col 8, coll' 8, and c. 8^{va} stand for coll'ottava, meaning "with octave", i.e., to play notes in passage together with notes in notated octaves. Any of these directions can be canceled with word *loco*, but often a dashed line or bracket indicates extent of music affected.

3.2.4 Equivalence

After unison, octave is simplest interval in music. Human ear tends to hear both notes as being essentially "the same", due to closely related harmonics. Notes are separated by an octave "ring" together, adding a pleasing sound to music. Interval is so natural to humans that when men & women are asked to sing in unison, they typically sing in octave.

– Sau unison, quãng tám là quãng đơn giản nhất trong âm nhạc. Tai người có xu hướng nghe cả hai nốt nhạc về cơ bản là "giống nhau", do các âm bội có liên quan chặt chẽ. Các nốt nhạc được tách ra bằng 1 "vòng" quãng tám, tạo nên âm thanh dễ chịu cho âm nhạc. Quãng rất tự nhiên đối với con người đến nỗi khi đàn ông & phụ nữ được yêu cầu hát đồng thanh, họ thường hát ở quãng tám.

For this reason, notes an octave apart are given same note name in Western system of music notation – name of a note an octave above A is also A. This is called *octave equivalence*, assumption that pitches 1 or more octaves apart are musically equivalent in many ways, leading to convention "that scales are uniquely defined by specifying intervals within an octave". Conceptualization of pitch as having 2 dimensions, pitch height (absolute frequency) & pitch class (relative position within octave), inherently include octave circularity. Thus all C♯s (or all 1s, if $C = 0$), any number of octaves apart, are part of same pitch class.

– Vì lý do này, các nốt cách nhau 1 quãng tám được đặt cùng tên trong hệ thống ký hiệu âm nhạc của phương Tây – tên của 1 nốt cao hơn A 1 quãng tám cũng là A. Điều này được gọi là *tương đương quãng tám*, giả định rằng các cao độ cách nhau 1 hoặc nhiều quãng tám là tương đương về mặt âm nhạc theo nhiều cách, dẫn đến quy ước "rằng scales được xác định duy nhất bằng cách chỉ định các khoảng trong 1 quãng tám". Khái niệm về cao độ có 2 chiều, độ cao cao độ (tần số tuyệt đối) & lớp cao độ (vị trí tương đối trong quãng tám), vốn bao gồm tính tròn của quãng tám. Do đó, tất cả các nốt C♯ (hoặc tất cả các nốt 1, nếu $C = 0$), cách nhau bất kỳ số quãng tám nào, đều thuộc cùng 1 lớp cao độ.

Octave equivalence is a part of most musical cultures, but is far from universal in "primitive" & early music. Languages in which oldest extant written documents on tuning are written, Sumerian & Akkadian, have no known word for "octave". However, believed: a set of cuneiform tablets that collectively describe tuning of a 9-stringed instrument, believed to be a Babylonian lyre, describe tunings for 7 of strings, with indications to tune remaining 2 strings an octave from 2 of 7 tuned strings. Leon Crickmore recently proposed: "Octave may not have been thought of as a unit in its own right, but rather by analogy like 1st day of a new 7-day week."

– Sự tương đương quãng tám là 1 phần của hầu hết các nền văn hóa âm nhạc, nhưng không phổ biến trong "nguyên thủy" & âm nhạc thời kỳ đầu. Các ngôn ngữ mà các tài liệu viết tay lâu đời nhất còn tồn tại về cách lên dây được viết, tiếng Sumer & Akkad, không có từ nào được biết đến cho "quãng tám". Tuy nhiên, người ta tin rằng: 1 bộ cuneiform các tấm bia mô tả chung cách lên dây của 1 nhạc cụ có 9 dây, được cho là 1 lyre của người Babylon, mô tả cách lên dây cho 7 dây, với chỉ dẫn lên dây 2 dây còn lại 1 quãng tám từ 2 trong số 7 dây đã lên dây. Leon Crickmore gần đây đã đề xuất: "Quãng tám có thể không được coi là 1 đơn vị riêng biệt, mà giống như ngày đầu tiên của 1 tuần 7 ngày mới."

Monkeys experience octave equivalence, & its biological basis apparently is an octave mapping of neurons in auditory thalamus of mammalian brain. Studies have also shown perception of octave equivalence in rats, human infants, & musicians but not starlings, 4–9-year-old children, or non-musicians." – Wikipedia/octave

3.3 Wikipedia/transcription (music)

"A J. S. Bach keyboard piece transcribed for guitar. In music, *transcription* is practice of notating a piece or a sound which was previously unnotated &/or unpopular as a written music, e.g., a jazz improvisation or a video game soundtrack. When a musician is tasked with creating sheet music from a recording & they write down notes that make up piece in music notation, it is said: they created a *musical transcription* of that recording. Transcription may also mean rewriting a piece of music, either solo or ensemble, for another instrument or other instruments than which it was originally intended. Beethoven Symphonies transcribed for solo piano by Franz Liszt are an example. Transcription in this sense is sometimes called *arrangement*, although strictly speaking transcriptions are faithful adaptations, whereas arrangements change significant aspects of original piece.

Further examples of music transcription include ethnomusicological notation of oral traditions of folk music, e.g. Béla Bartók's & Ralph Vaughan Williams' collections of national folk music of Hungary & England resp. French composer Olivier Messiaen transcribed birdsong in wild, & incorporated it into many of his compositions, e.g. his Catalogue d'oiseaux for solo piano. Transcription of this nature involves scale degree recognition & harmonic analysis, both of which transcriber will need relative or perfect pitch to perform.

In popular music & rock, there are 2 forms of transcription. Individual performers copy a note-for-note guitar solo or other melodic line. As well, music publishers transcribe entire recordings of guitar solos & bass lines & sell sheet music in bound books. Music publishers also publish PVG (piano/vocal/guitar) transcriptions of popular music, where melody line is transcribed, & then accompaniment on recording is arranged as a piano part. Guitar aspect of PVG label is achieved through guitar chords written above melody. Lyrics are also included below melody.

3.3.1 Adaptation

Some composers have rendered homage to other composers by creating "identical" versions of earlier composers' pieces while adding their own creativity through use of completely new sounds arising from difference in instrumentation. Most widely known example of this is Ravel's arrangement for orchestra of Mussorgsky's piano piece *Pictures at an Exhibition*. Webern used his transcription for orchestra of 6-part ricercar from Bach's *The Musical Offering* to analyze structure of Bach piece, by using different instruments to play different subordinate motifs of Bach's themes & melodies.

In transcription of this form, new piece can simultaneously imitate original sounds while recomposing them with all technical skills of an expert composer in such a way that it seems: piece was originally written for new medium. But some transcriptions & arrangements have been done for purely pragmatic or contextual reasons. E.g., in Mozart's time, overtures & songs from this popular operas were transcribed for small wind ensemble simply because such ensembles were common ways of providing popular entertainment in public places. MOZART himself did this in his own opera *The Marriage of Figaro*. A more contemporary example is STRAVINSKY's transcription for 4 hands piano of *The Rite of Spring*, to be used on ballet's rehearsals. Today musicians who play in cafes or restaurants will sometimes play transcriptions or arrangements of pieces written for a larger group of instruments.

Other examples of this type of transcription include BACH's arrangement of VIVALDI's 4-violin concerti for 4 keyboard instruments & orchestra; MOZART's arrangement of some Bach fugues from *The Well-Tempered Clavier* for string trio; BEETHOVEN's arrangement of his *Groβe Fuge*, originally written for string quartet, for piano duet, & his arrangement of his Violin Concerto as a piano concerto; Franz Liszt's piano arrangements of works of many composers, including symphonies of Beethoven; TCHAIKOVSKY's arrangement of 4 Mozart piano pieces into an orchestral suite called "Mozartiana"; MAHLER's re-orchestration of SCHUMANN symphonies; & SCHOENBERG's arrangement for orchestra of BRAHMS's piano quintet & BACH's "St. Anne" Prelude & Fugue for organ.

Since piano became a popular instrument, a large literature has sprung up of transcriptions & arrangements for piano of works for orchestra or chamber music ensemble. These are sometimes called "piano reductions", because multiplicity of orchestral parts – in an orchestral piece there may be as many as 2 dozen separate instrumental parts being played simultaneously – has to be reduced to what a single pianist (or occasionally 2 pianists, or 1 or 2 pianos, e.g. different arrangements for GEORGE GERSHWIN's *Rhapsody in Blue*) can manage to play.

Piano reductions are frequently made of orchestral accompaniments to choral works, for purposes of rehearsal or of performance with keyboard alone.

Many orchestral pieces have been transcribed for concert band.

3.3.2 Transcription aids

- **Notation software.** Since advent of desktop publishing, musicians can acquire music notation software, which can receive user's mental analysis of notes & then store & format those notes into standard music notation for personal printing or professional publishing of sheet music. Some notation software can accept a Standard MIDI File (SMF) or MIDI performance as input instead of manual note entry. These notation applications can export their scores in a variety of formats like EPS, PNG, & SVG. Often software contains a sound library that allows user's score to be played aloud by application for verification.

- **Slow-down software.** Prior to invention of digital transcription aids, musicians would slow down a record or a tape recording to be able to hear melodic lines & chords at a slower, more digestible pace. Problem with this approach was: it also changed pitches, so once a piece was transcribed, it would then have to be transposed into correct key. Software designed to slow down tempo of music without changing pitch of music can be very helpful for recognizing pitches, melodies, chords, rhythms, & lyrics when transcribing music. However, unlike slow-down effect of a record player, pitch & original octave of notes will stay same, & not descend in pitch. This technology is simple enough that it is available in many free software applications.

  Software generally goes through a 2-step process to accomplish this. 1st, audio file is played back at a lower sample rate than that of original file. This has same effect as playing a tape or vinyl record at slower speed – pitch is lowered meaning music can sound like it is in a different key. 2nd step: use Digital Signal Processing (or DSP) to shift pitch back up to original pitch level or musical key.

- **Pitch tracking software.** Main article: Wikipedia/pitch tracker. As mentioned in the Automatic music transcription sect, some commercial software can roughly track pitch of dominant melodies in polyphonic musical recordings. Note scans are not exact, & often need to be manually edited by user before saving to file in either a proprietary file format or in Standard MIDI File Format. Some pitch tracking software also allows scanned note lists to be animated during audio playback.

3.3.3 Automatic music transcription (AMT)

Term "automatic music transcription" was 1st used by audio researchers JAMES A. MOORER, MARTIN PISZCZALSKI, & BERNARD GALLER in 1977. With their knowledge of digital audio engineering, these researchers believed: a computer could be programmed to analyze a digital recording of music s.t. pitches of melody lines & chord patterns could be detected, along with rhythmic accents of percussion instruments. Task of AMT concerns 2 separate activities: making an analysis of a musical piece, & printing out a score from that analysis.

This was not a simple goal, but one that would encourage academic research for at least another 3 decades. Because of close scientific relationship of speech to music, much academic & commercial research that was directed toward more financially resourced speech recognitions technology would be recycled into research about music recognition technology. While many musicians & educators insist that manually doing transcriptions is a valuable exercise for developing musicians, motivation for AMT remains same as motivation for sheet music: musicians who do not have intuitive transcription skills will search for sheet music or a chord chart, so that they may quickly learn how to play a song. A collection of tools created by this ongoing research could be of great aid to musicians. Since much recorded music does not have available sheet music, an automatic transcription device could also offer transcriptions that are otherwise unavailable in sheet music. To date, no software application can yet completely fulfill JAMES MOORER;s definition of AMT. However, pursuit of AMT has spawned creation of many software applications that can aid in manual transcription. Some can slow down music while maintaining original pitch & octave, some can track pitch of melodies, some can track chord changes, & others can track beat of music.

Automatic transcription most fundamentally involves identifying pitch & duration of performed notes. This entails tracking pitch & identifying note onsets. After capturing those physical measurements, this information is mapped into traditional music notation, i.e., sheet music.

Digital Signal Processing is branch of engineering that provides software engineers with tools & algorithms needed to analyze a digital recording in terms of pitch (note detection of melodic instruments), & energy content of un-pitched sounds (detection of percussion instruments). Musical recordings are sampled at a given recording rate & its frequency data is stored in any digital wave format in computer. Such format represents sound by digital sampling.

- **Pitch detection.** Pitch detection is often detection of individual notes that might make up a melody in music, or notes in a chord. When a single key is pressed upon a piano, what we hear is not just *1* frequency of sound vibration, but a *composite* of multiple sound vibrations occurring at different mathematically related frequencies. Elements of this composite of vibrations at differing frequencies are referred to as harmonics or partials.

  E.g., if note A_3 (220 Hz) is played, individual frequencies of composite's harmonic series will start at 220 Hz as fundamental frequency: 440 Hz would be 2nd harmonic, 660 Hz would be 3rd harmonic, 880 Hz would be 4th harmonic, etc. These are integer multiples of fundamental frequency (e.g., $2 \cdot 220 = 440$, 2nd harmonic). While only about 8 harmonics are really needed to audibly recreate note, total number of harmonics in this mathematical series can be large, although higher harmonic's numerical weaker magnitude & contribution of that harmonic. Contrary to intuition, a musical recording at its lowest physical level is not a collection of individual notes, but is really a collection of individual harmonics. That is why very similar-sounding recordings can be created with differing collections of instruments & their assigned notes. As long as total harmonics of recording are recreated to some degree, it does not really matter which instruments or which notes were used.

- **Beat detection.**
- **How ATM works.**
- **Detailed computer steps behind AMT.**

" – Wikipedia/transcription (music)

4 Miscellaneous

Tài liệu

[HWR22] Michael S. Horn, Melanie West, and Cameron Roberts. *Introduction to Digital Music with Python Programming: Learning Music with Code*. 1st edition. Focal Press, 2022, p. 262.

[KD06] Anssi Klapuri and Manuel Davy. *Signal Processing Methods for Music Transcription*. Springer, 2006, p. 440.

[Mül15] Meinard Müller. *Fundamentals of music processing*. Audio, analysis, algorithms, applications. Springer, Cham, 2015, pp. xxix+487. ISBN: 978-3-319-21944-8; 978-3-319-21945-5. DOI: 10.1007/978-3-319-21945-5. URL: https://doi.org/10.1007/978-3-319-21945-5.

[Mül21] Meinard Müller. *Fundamentals of music processing—using Python and Jupyter notebooks*. Second edition [of 3382223]. Springer, Cham, [2021] ©2021, pp. xxxi+495. ISBN: 978-3-030-69807-2; 978-3-030-69808-9. DOI: 10.1007/978-3-030-69808-9. URL: https://doi.org/10.1007/978-3-030-69808-9.

[Väl+06] Vesa Välimäki, Jyri Pakarinen, Cumhur Erkut, and Matti Karjalainen. "Discrete-time modelling of musical instruments". In: *Rep. Prog. Phys.* 69.1 (2006), pp. 1–78. DOI: 10.1088/0034-4885/69/1/R01. URL: https://iopscience.iop.org/article/10.1088/0034-4885/69/1/R01.