The Chinese University of Hong Kong
Department of Computer Science and Engineering

Final Year Project
2014 Spring - KY1302

User-adjustable Automatic Piano Reduction

Supervisor: Prof. Kevin Yip
Co-supervisor: Prof. Lucas Wong

Student Name: Ng Chiu Yuen
Student ID: 1155015247

Table of Contents

1 Background

Creating a piano reduction is the process of both realising and simplifying full score for multiple instruments into a two-staff piano score. There are several motivations for piano reduction, for example, piano may be used to mimic one or more instruments in rehearsal in orchestra, or simply for home enjoyment.

Piano reduction is a subjective process that there are usually no best solution for any use case. Different reduction result may be better or worse than others depending on the situation and pianist's judgement.

In this project, we want to develop a software tool that can perform piano reduction automatically. A user may apply different parameters to the music score, for example, putting foreground matericals such as bass line, melody and harmony into focus, redistributing notes among two or four hands, octave-displacing or eliminating particular instruments for pianistic practically. These functinos are analogous to tuning brightness and contrast of pictures provided by standard image processing software.

This is a joint project with Professor Lucas Wong at Soochow University School of Music. Professor Wong has provide many examples to describe and explain different parameters and methods to perform piano reduction.

2 Introduction

2.1 Previous Work

In the last semeseter, serveral parameters were implemented and the final program was able to classify notes based on users' input parameter and a threshold set by them.

The following summarize all work done in previous semester:

1. Manipulate MusicXML file format
2. Implement algorithms for 8 possible reduction parameters
3. Perform piano reduction based on linear combination of parameter weight and a threshold
4. Merge different parts of a score into one part

More details of previous work are described in their related sections.

Previous work was able to create a piano reduction based on users' parameter, however, the result is usually unpleasent as it was extremely difficult for a user to decide the correct weight and threshold and parameters. Also, linear combination of parameters weight may not be sufficient for complex decision making.

2.2 Objectives

There are several objectives in this project, aiming to improve previous result:

1. Error tolerance for MusicXML input format
2. Implement more possible reduction parameter
3. Try a machine learning approach to learn how a users perform piano reduction with training examples as actual input
4. Generate a playable two-staff piano score with less user interaction

3 Input Manipulation

3.1 Previous Work

Previous report has discussed about data structures used to represent a musical score.

In this project, MusicXML format is used as both input and output file format for representing musical score. MusicXML consists of several components which includes music information (e.g. measures, notes, chords, etc) and layout of the printed score.

This summarize the previous work regarding MusicXML manipulation:

1. Understanding how to store music information in computer memory.
2. Making use of music21 library to manipulate MusicXML files.

However, there are a few problems about previous implementation, and improvements are made in this semester:

1. Unable to deal with parts consist of more than one voice.
2. Slow input due to ad-hoc solution for error tolerance of input.

3.2 Data Structures in Music Score

Music score consist of a hierachical structure, and it can be modeled as an ordered tree in computer.

3.2.1 Score

A score is the root node of the tree, which contains a list of parts, the order of different parts are preserved after the music contents exceed the width of page and continue in the next section.

3.2.2 Part

Every part is almost independent with each other. Each part contains a list of measures, where the order of each measure is very important and cannot be mixed.

3.2.3 Voice

Separate voices may or may not exist within a measure. However, we may always consider a part would consist of at least one voice. Details about voice is explained in later section.

3.2.4 Measure

A measure may be dependent to the previous or next measure when there are tied notes. Measures divide the music into smaller intervals so that the score is easy to read by musicians. Also, for the internal structure of music score stored in computer memory, measures make the data more organised and easier to query contents at different time intervals.

In some measures, time signautres, key signautres and clef are specified, they affect all measures after these notations until the next one is seen. We need to store the reference of relavent signatures to each measure.

3.2.5 Note

There are three main types of note, where they have different meanings in terms of music. Among three types, all of them contain information about onset and duration, specifying when and how long does a note take effect.

1. Rest
   Rest note means the player need not to do anything within certain time interval. There are different notations for different duration of rest. However, the symbol itself is not important for computational purpose.
2. Note
   A note is the basic component of music, it indicates the pitch that the player should play, which is equivalent to the frequency of sound wave that we can hear. Different duration of notes have different notations. And we only need to know about the pitch and duration for simplicity.
3. Chord
   A chord is a note composed by multiple pitch, and playing a chord gives several pitch at the same time. Usually, chord notes share the same duration but different in pitch.

3.3 Consistent Hierachical Structure

In previous implementation, voices were regarded as optional component for a part or measure during processing.

Though the implementation that consider voices as optional component is correct, it is not convenient to keep it optional. Therefore, this current implementation, the program should always reconstruct the hierachical structure, such that every part consists of at least one voice.

In such case, when the program loops through every part in the score, it need not consider two cases that one part has a voice or not. This simplified the implementation a lot while it does not affect the appearence of output score.

3.4 Error Tolerance of Input

The input module of music21 library did not handle any errors in input file, which is extremely problematic for further processing.

An ad-hoc solution was implemented in last semester. Since the output module of music21 library would prune the errors before output, for example, re-assign offset values of notes and chords to non-negative numbers, the program would first read in a file which may or may not contains errors in any part, and write the same score to a temporary file to let the output module of music21 handles the error. After that, read the temporary file to obtain an error-free strcuture.

The above solution was very slow, especially when the input score is large, and music21 would perform several reconstrction of data strcuture both during input and output. This means the above solution has an disk I/O bottleneck.

In current implementation, after processing the input MusicXML file by the import module of music21, extra process is done to reconstruct the hierachical strcutre and prune errors of the structure in memory.

The final approach saved one write to disk and one read, and it tackles two problems at the same time.

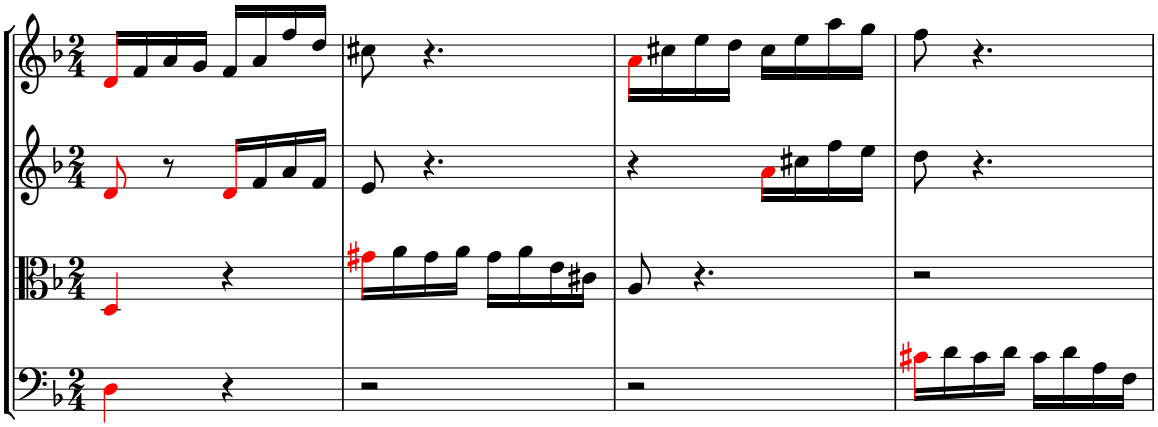
4 Reduction Algorithms

4.1 Basic Parameters

Seven basic reduction parameters were implemented and used to classify notes based on linear combination of these parameters and a threshold.

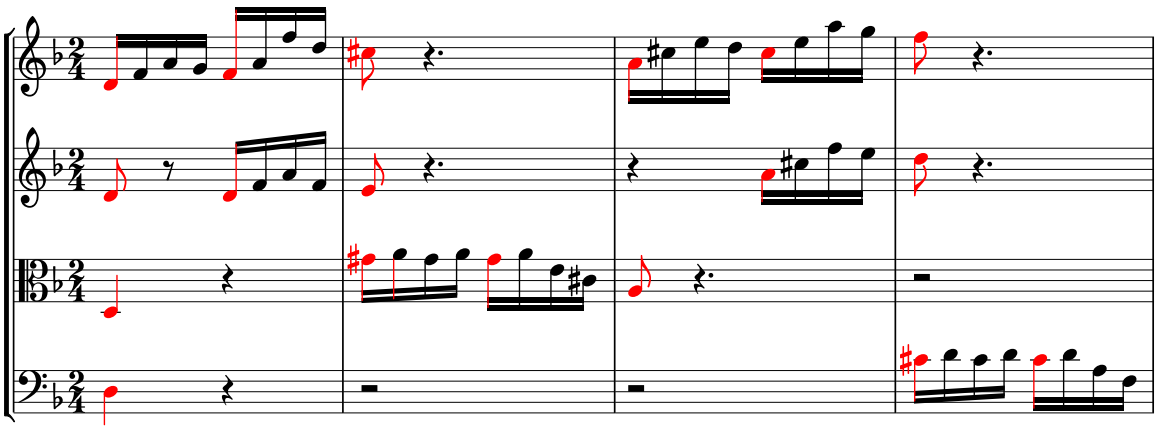
4.1.1 Onset After Rest

A note onset after rest notes are usually important as it can be recognised easily. One parameter we consider is to give bonus to notes at the beginning or sound after rest.



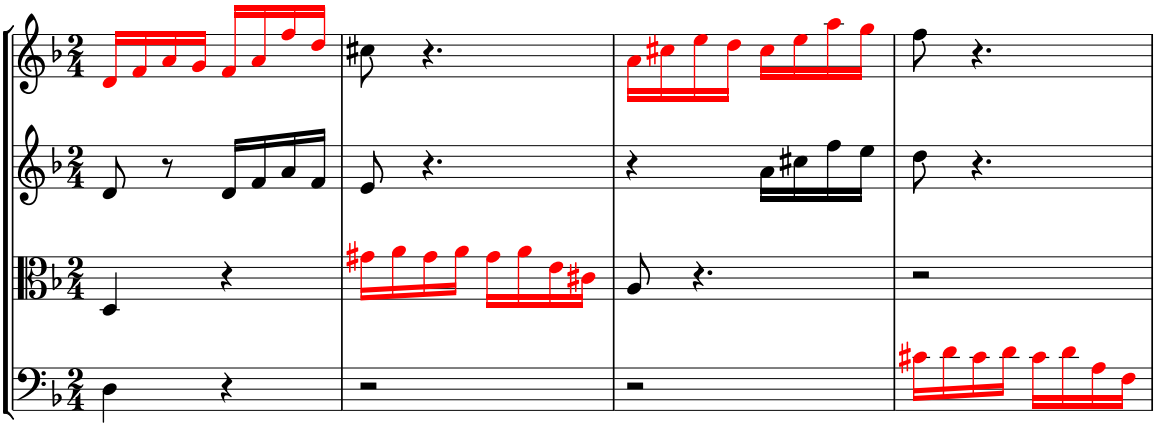
4.1.2 Strong Beats

Beats are usually recognised as important notes since they are easy to count. To find a strong beat, we first need to define the smallest division, e.g. quarter or eighth, every note onset at integral multiple of the division defined.



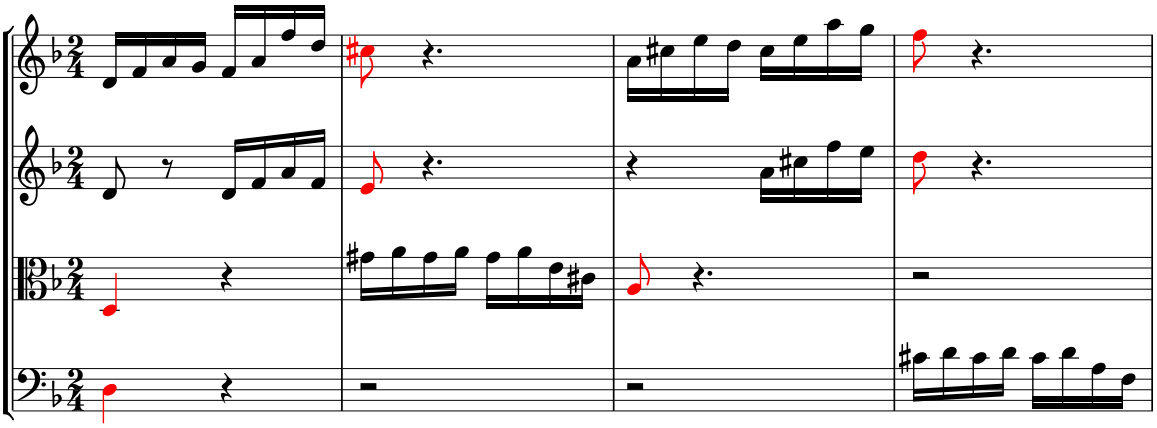
4.1.3 Active Rhythm

When considering only one measure with several parts, we would say the rhythm is active when it has the most number of notes.



4.1.4 Sustained Rhythm

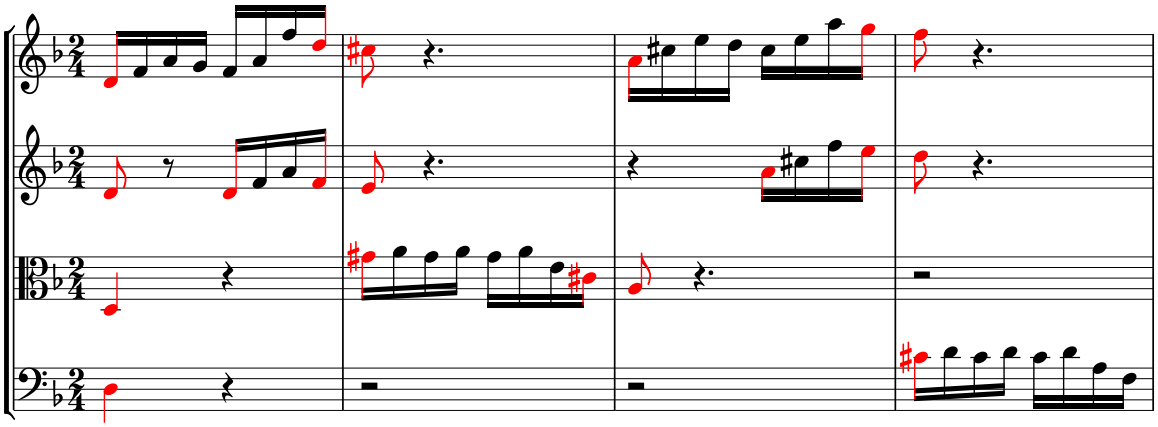
Sustained rhythm is the opposite of active rhythm, instead of finding a part containining the most number of notes within the same measure, it is finding part with the least number of notes.



4.1.5 Rhythm Variety

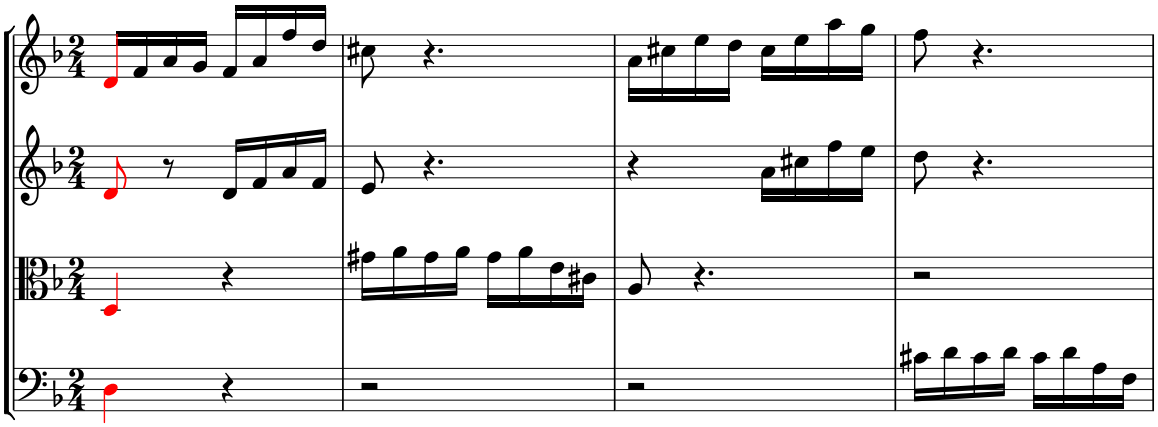
Rhythm variety consider notes with a rhythm different from the previous or next rhythm. And there are two cases for changing rhythm.

1. Rest on the left or right of a note
2. Duration of the note before or after is different



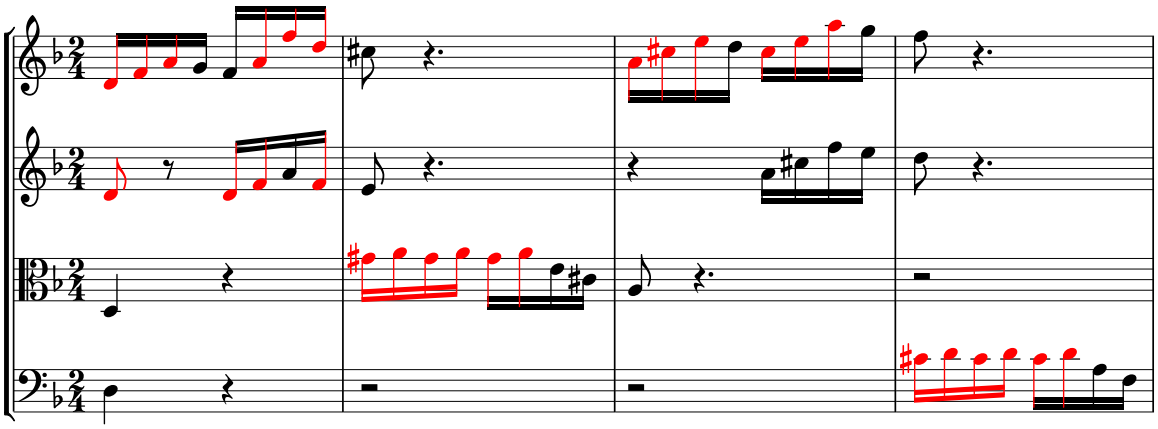
4.1.6 Vertical Doubling

Vertical doubling means doubling of the same pitch-class note at the same onset.



4.1.7 Pitch Occurrence

In each measure, notes of pitch-class with the highest frequency are considered as important under this parameter. If the measure consist of two voices, then the two voices are considered separately. However, if there is only one note in a measure, that note would not be qualified for the bonus score.

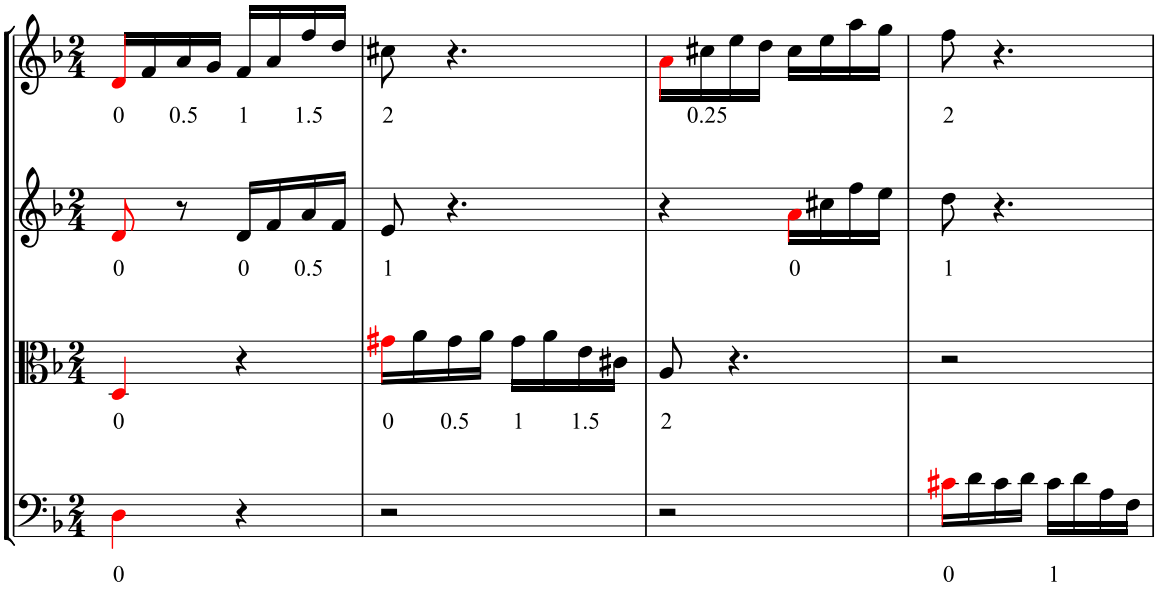


4.2 Additional Parameters

In addition to the parameters implemented in last semester, three parameters are added to provide more options for the user, and helpful for parameter learning, which will be covered in next section.

4.2.1 Important Entrances

Important entrance is a variation to the 'onset after rest' parameter implemented in last semester. Serveral notes after rest may considered as important entrance, and a numerical value is given by calcuating the offset of a note from the last rest in the same part.



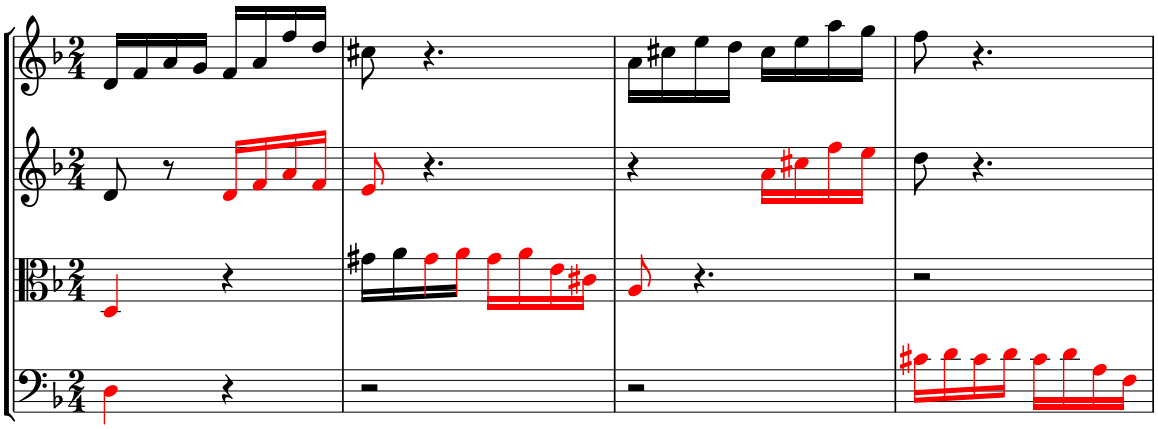
The above diagram shows some value assigned to notes. Red notes are notes with 'onset after rest' property. Not all values are shown because of space.

The effect of this parameter is non-trivial, and is determined later by training with examples provided by Lucas.

4.2.2 Bass Line

As a foundation of harmony, bass line is another important parameter that may help piano reduction. It is determined by the lowest pitch at any time, considering the whole score.

The following score shows which notes are bass line, and they are colored in red.



To simplify the implementation, this parameter is implemented in a different way and it is not all correct.

In each part, consider notes' pitch in each measure:

```
P[1] = [ [ D4, F4, A4, G4, F4, A4, F5, D5 ], [ C#5 ],
         [ A4, C#5, E5, D5, C#5, E5, A5, G5 ], [ F5 ] ]
P[2] = [ [ D4, D4, F4, A4, F4 ], [ E4 ], [ A4, C#5, F5, E5 ], [ D5 ] ]
P[3] = [ [ E3, D3, A3, A3 ], [ D3 ], [ G#4, A4, G#4, A4, E4, C#4 ], [ A3 ] ]
P[4] = [ [ D3 ], [], [], [ C#4, D4, C#, D4, C#4, A3, F3 ] ]
```

Convert the pitches to a number representing pitch steps:

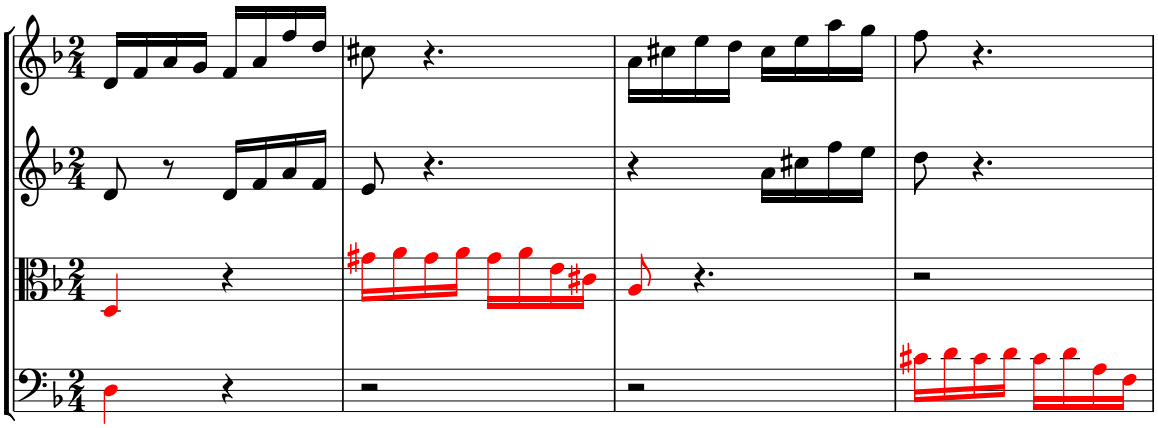
```
P[1] = [ [ 62, 65, 69, 67, 65, 69, 77, 74 ], [ 73 ],
         [ 69, 73, 76, 74, 73, 76, 81, 79 ], [ 77 ] ]
P[2] = [ [ 62, 62, 65, 69, 63 ], [ 64 ], [ 69, 73, 77, 76 ], [ 74 ] ]
P[3] = [ [ 52, 50, 57, 57 ], [ 50 ], [ 68, 69, 68, 69, 64, 61 ], [ 57 ] ]
P[4] = [ [ 50 ], [], [], [ 61, 62, 61, 62, 57, 53 ] ]
```

Find the meadian of pitch steps for each part and each measure:

```
median(P[1]) = [ 68, 73, 75, 77 ]
median(P[2]) = [ 65, 64, 74.5, 74 ]
median(P[3]) = [ 54.5, 50, 68, 57 ]
median(P[4]) = [ 50, inf, inf, 61 ]
```

For each measure, find a part with the smallest median, and mark all notes inside become the bass line. Median of pitch steps for measure without any notes are defined as infinity.

The result is shown in the following score, which is slightly different from the correct answer.

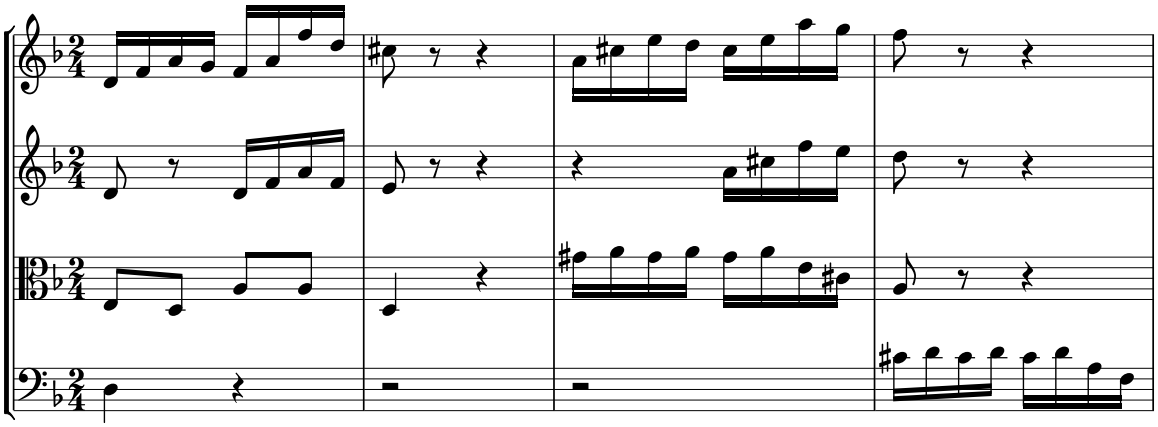


4.2.3 Statistics of Pitch Class

It is hard to determine whether a note belongs to the harmony as there are too many combinations.

An alternative solution is to consider local statistics of a measure, and determine which notes belong to the harmony.

Consider the following score:



For each measure, construct a histogram of pitch class:

```
Index = [ C, C#/Db, D, ..., B ]
H[1] = [0, 0, 6, 0, 1, 5, 0, 1, 0, 5, 0, 0]
H[2] = [0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
H[3] = [0, 4, 1, 0, 4, 1, 0, 1, 3, 6, 0, 0]
H[4] = [0, 3, 4, 0, 0, 2, 0, 0, 0, 2, 0, 0]
```

Then for each note, append the statistics as 12 parameters.

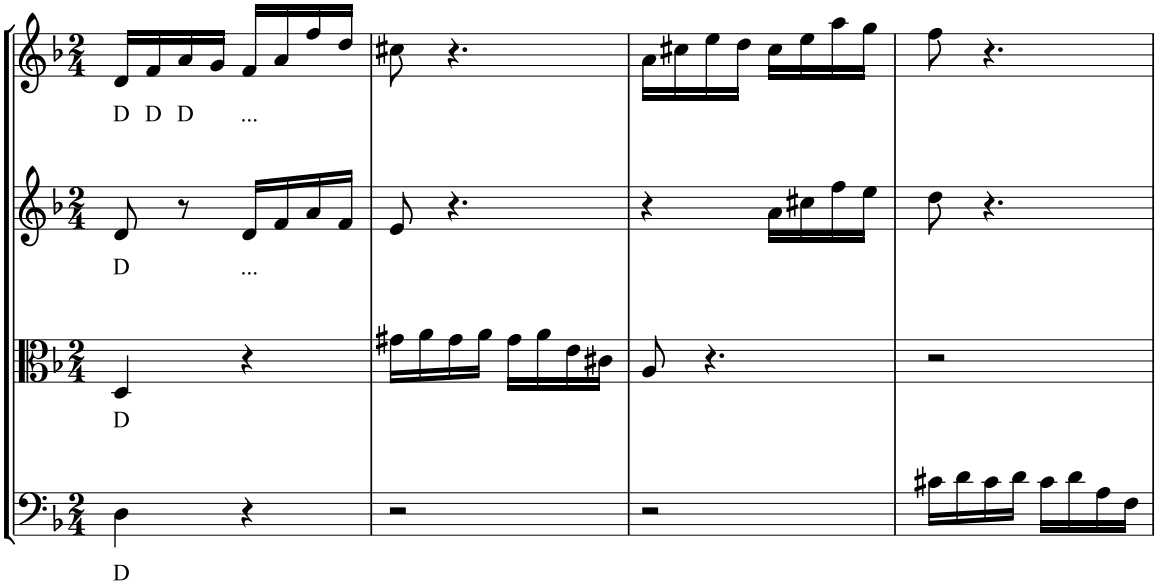
Similar to the 'bass line' parameter, the effect of this statistics is non-trival, the underlying rules for determining which pitch should belongs to the harmony is determined by the nerual network discussed in the next section.

4.2.4 Simulating Consonance

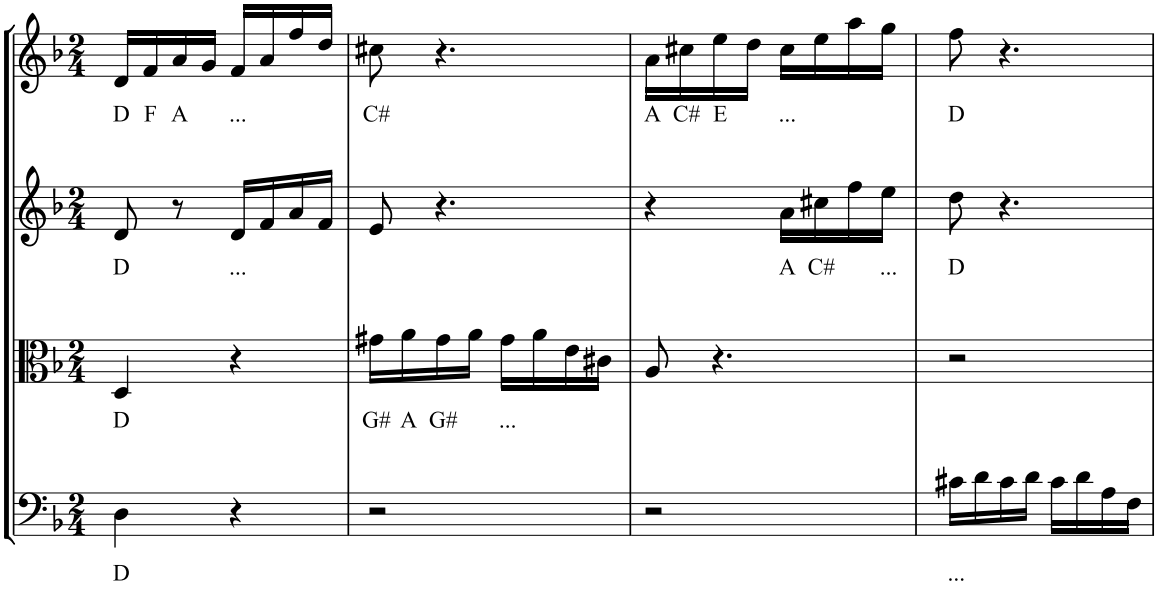
To provide sufficient information for the learning algorithm to guess which note belongs to the harmony, we need to provide two more parameter in addition to 'bass line' and 'statistics of pitch class'.

For each note, find the pitch class of corresponding bass line and the pitch class of itself. These two parameters is added to each note as enumerable.

- Mark the pitch class of the bass line to every note



- Mark the pitch class of the note itself



5 Parameter Learning

In this section, we are going to discuss a computational model that is capable of machine learning and pattern recognition, neural network.

After building the learning model and provide sufficient training to the model, we expect that we can estimate users' actual parameter based on a small portion of piano reduction created by user manually.

If this is successful, we may ease the work of users in two ways:

1. Only creating a small portion of piano reduction will allow the program to generate acceptable result for the whole score.
2. Musician users may not be able to understand how numerical weight of different parameters will affect the outcome, though they know how to make decision on different notes, and the learning model will try to understand how they make decision.

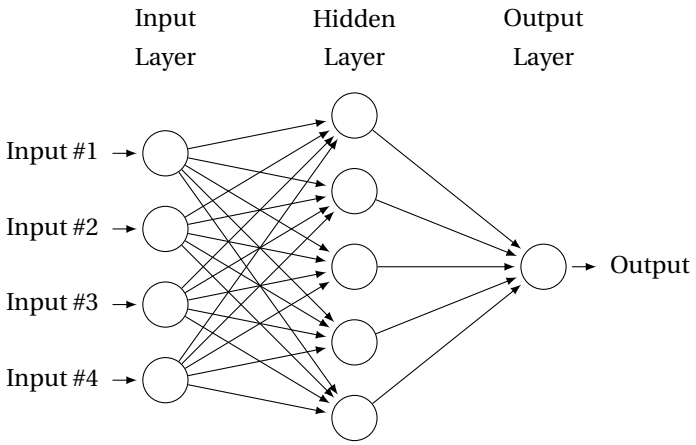
5.1 Nerual Network

5.1.1 Network Structure

Neural network is composed of a number of nodes (or units), where they are connected by link with a numerical weight.

Learning takes place by updating link weights, and to bring the network behaviour more into line with the input provided.

In general, a neural network consists of serveral layers, one input layer, one output layer and zero or more hidden layers. The following diagram illustrates the structure.



In each node, it accepts the output from previous layer as input, and generate output that will be fed to the next layer.

Before going to the details about the neural network, we first define the following symbols:

H_k	number of units in k-th layer
$h_{k,i}$	activation level of k-th layer i-th unit
$w_{k,i,j}$	link weight of the link connecting $(k-1)$-th layer i-th unit to k-th layer j-th unit

5.1.2 Basic Unit

A unit is the basic element of neural network. Each unit consist of input links from other units, output links to other units, an activation level and means of computing the activation level at next step.

Every unit within the network only does a local computation based on inputs from its neighbours, which is independent from any global control over all units.

Input to a unit is computed by weighted sum of output from previous layer. For units in input layer, the input is directly feed in.

$$\text{in}_{i,k} = \sum_{j=1}^{H_{i-1}} w_{i-1,j,k} \cdot h_{i-1,j}$$

The above equation shows how the input of an unit is computed, where $\text{in}_{i,j}$ is the input value of the unit placed in i-th layer j-th position, H_{i-1} is the number of units in the previous layer ($(i-1)$-th layer), $w_{i-1,k,j}$ is the weight of link connecting $(i-1)$-th layer k-th unit to i-th layer j-th unit and $h_{i-1,k}$ is the output of k-th unit in $(i-1)$-th layer.

After accepting the input from previous layer, output is then computed by an activating function g, which will be discussed in the next part.

5.1.3 Activating Function

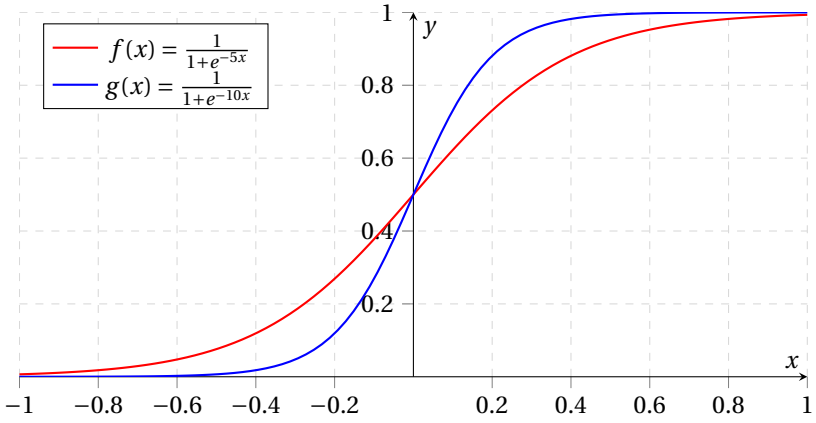
An activating function g is used to update the activation level of a unit based on input value.

In this project, sigmoid function is used as the activating function for every unit except input layer. For input layer's units, input is directly copied to their output.

The following equations is the sigmoid function used, a constant a is added to the exponentiation term for adjusting the hardness of the function.

$$g(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-ax}}$$

The following graph shows the sigmoid function, its output converges to 0 for negative input, and converges to 1 for positive input.



The sigmoid function instead of a 0-1 step function is chosen as sigmoid function gives a smooth curve that is differentiable. This is essential for the later part discussing how link weights are updated.

5.1.4 Back-Propagation Learning

After initializing the network and choosing an appropriate activating function for each unit, the network immediately has the ability to produce one or more output by based on input parameters.

However, this does not make any sense because the network may not be able to produce correct output. Therefore, we need to train the network, and a popular method for learning in multi-layer network is called back-propagation.

The general idea of back-propagation is every link weight and unit in previous layer is responsible for part of the error of the output. As the error term for output layer can be easily computed by comparison and propagate back to previous layers.

For each output $O_k = g\left(\sum_{j=1}^{H_{i-1}} h_{i-1,j} \cdot w_{i-1,j,k}\right)$ and its error term $E_k = \frac{1}{2}(T_k - O_k)^2$, the link weight of previous layer contributes some error to the output O_k:

$$\begin{aligned}\frac{\partial E}{\partial w_{i,j,k}} &= \frac{\partial E}{\partial O_k} \cdot \frac{\partial O_k}{\partial w_{i,j,k}} \\ &= (O_k - T_k) \cdot g'\left(\sum_{j=1}^{H_{i-1}} h_{i-1,j} \cdot w_{i-1,j,k}\right) \cdot h_{i-1,j} \\ &= (O_k - T_k) \cdot (h_{i,j} \cdot (1 - h_{i,j})) \cdot h_{i-1,j}\end{aligned}$$

Also, output from the i-th layer will cause to the error of output in $(i+1)$-th layer, and all error will be propagated to next layer:

$$\begin{aligned}\frac{\partial E}{\partial h_{i,j}} &= \sum_{j=1}^{H_{i+1}} \frac{\partial E}{\partial h_{i+1,j}} \cdot \frac{\partial h_{i+1,j}}{\partial h_{i,k}} \\ &= \sum_{j=1}^{H_{i+1}} \frac{\partial E}{\partial h_{i+1,j}} \cdot (h_{i+1,j}(1 - h_{i+1,j}) w_{i,j,k}) \\ &= \sum_{j=1}^{H_{i+1}} \left[\frac{\partial E}{\partial h_{i+1,j}} \cdot h_{i+1,j}(1 - h_{i+1,j})\right] w_{i,j,k} \\ &= \sum_{j=1}^{H_{i+1}} \Delta_{i+1,j} \cdot w_{i,j,k}\end{aligned}$$

Errors in link weights also contribute to the error of final output and propagate to next layer:

$$\begin{aligned}\frac{\partial E}{\partial w_{i,j,k}} &= \frac{\partial E}{\partial h_{i+1,k}} \cdot \frac{\partial h_{i+1,k}}{\partial w_{i,j,k}} \\ &= \frac{\partial E}{\partial h_{i+1,k}} \cdot (h_{i+1,k}(1 - hi+1,k) \cdot h_{i,j}) \\ &= \left[\frac{\partial E}{\partial h_{i+1,k}} \cdot h_{i+1,k}(1 - h_{i+1,k})\right] h_{i,j} \\ &= \Delta_{i+1,k} \cdot h_{i,j}\end{aligned}$$

Based on the above equations, we may update link weights layer-by-layer, from output layer to input layer.

The following pseudocode shows how link weights are updated in each learning epoch, and for each update, a learning rate α is added to the update term to avoid the problem of overshotting.

Calculate $\frac{\partial E}{\partial w_{K,j,k}}$

for $i \leftarrow K-1$ **to** 0 **do**

Calcuate $\frac{\partial E}{\partial h_{i+1,k}}$ and $\frac{\partial E}{\partial w_{i,j,k}}$

$w_{i,j,k} \leftarrow w_{i,j,k} - \alpha \frac{\partial E}{\partial w_{i,j,k}}$

end for

5.2 Modelling Parameter

In order to feed our examples to the neural network for training and classification, we need to model different kinds of parameter in input and output to numerical values that the network can understand.

5.2.1 Input Parameters

According to all the reduction parameters implemented in both semesters, we may conclude that we have three main types of parameters:

1. Boolean Value
2. Enumerable or Set
3. Statistics or Histogram

Each boolean parameter can be modelled as $\{0,1\}$ input and pass into the network as one input unit. Examples of boolean parameters implemented are 'onset after rest', 'strong beats', etc.

Enumerable can be modelled as a list of $\{0,1\}$ values. For example, to indicate which pitch class does a note belongs to, we have a list of length 12, with 1 one and 11 zeros where the 1 is placed at the position that labeled as certain pitch.

Set is similar to enumerable, however, in this case we allow multiple properties co-exist with each other. We may consider multiple boolean parameters together as a set.

For statistical parameter like simulating consonance, for columns in a histogram, we may model each column in the histogram as one input unit, and the numerical value of the histogram becomes the input value fed into the network.

5.2.2 Output with Threshold

In our implementation for classifying which music note should be kept or not, we are using a threshold and compare it with the marks given to a particular note.

For the single-layer neural network, where the input layer is directly connected to the output layer. Consider the case that we have several boolean attributes with different weights passed to the network, and we want to classify them with a threshold.

$$\text{in}_{i,k} = \sum_{j=1}^{H_{i-1}} w_{i-1,j,k} \cdot h_{i-1,j} > t$$

However, when we consider the activating function $g(x) = \dfrac{1}{1+e^{-ax}}$, which gives positive output with positive input value and vice versa, meaning that we need to guarantee a positive input to the output layer and negative input to the output layer for notes that we want to keep or not to keep with respectively.

To model output singal with a threshold, we may add a bias unit connecting to the output layer, where bias is a special input unit that its activation level is always 1, and the weight is the negative of original threshold.

6 Learning Results

6.1 Examples with Known Answer

To verify that the neural network constructed in previous section is able to estimate parameters applied and corresponding weight, test data is generated by applying known parameter, random weight and threshold.

Generated test data is then used as training sample, also model answer for comparison with estimated reduced score.

6.1.1 Generating Test Data

The following table shows a list of test data generated, each of them are given a label. Only parameter weights and threshold is provided here, scores generated for this section are shown in appendix.

Reduced score is generated from segments extracted from 'Beethoven String Quartet Op.18 No.1 4th Movement'. Three segments are selected from measure 1-18, 34-48 and 136-150.

Parameter \ Label	A	B
Onset After Rest	1	1
Strong Beats	1	0
Active Rhythm	1	0
Sustained Rhythm	0	1
Rhythm Variety	0	1
Vertical Doubling	0	1
Occurrence	1	1
Threshold	2	2

The above examples are generated using the program implemented in last semester, where notes are selected based on a linear combination of score given to different parameters.

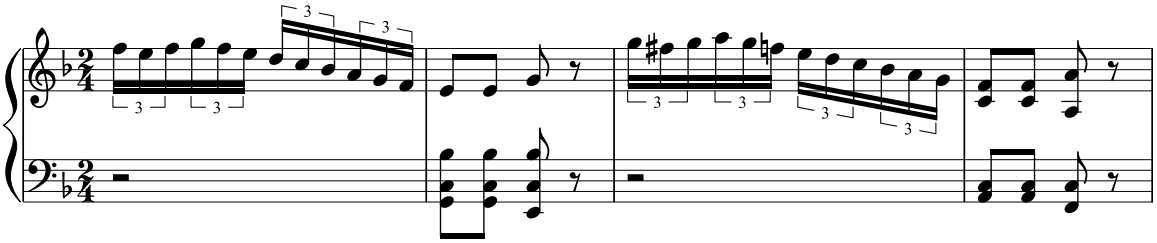
In such case, we should expect a network constrcuted as single-layer perceptron should be able to estimate weights given by different parameter correctly.

6.1.2 Training with Expected Result of Same Segment

In this section, training is done by providing expected result from the same segment as the target segment. The first value in network parameters for all results is the threshold predicted by the trained network.

The following example is the result of group A, measure 1-18, using the expected result of this segment as training example.

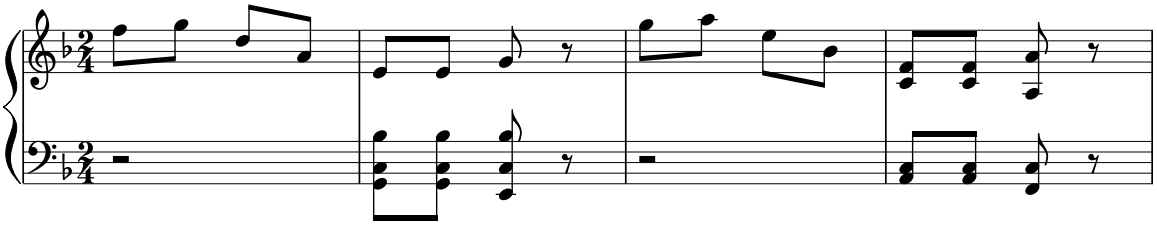
- Training with 10 iterations
  - First 4 Measures of Resulting Score



  - Network Parameters

```
[ 1.41326752 0.34904355 0.83589726 2.00467845 1.83465518 -1.26802504 0.93463713 -0.21044327 ]
```

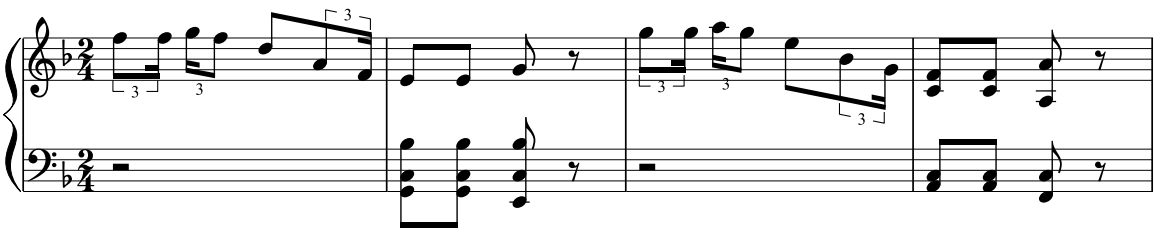
- Training with 50 iterations
  - First 4 Measures of Resulting Score



  - Network Parameters

```
[ 0.2098871 0.43186398 0.5911689 1.34800722 1.4345822 -2.04762829 0.53815176 0.20569448 ]
```

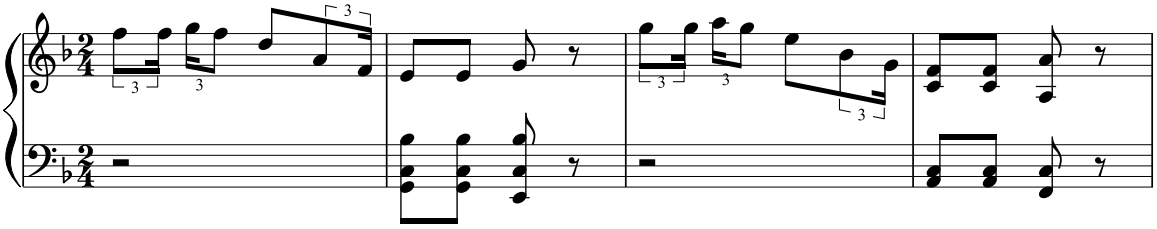
- Training with 300 iterations
  - First 4 Measures of Resulting Score



  - Network Parameters

```
[ -2.06727232 0.86608202 1.75250521 1.682594 -0.03932087 -0.89469602 -0.01904543 1.73729456 ]
```

The expected result of group A, measure 1-4 is shown in the following diagram.



After training for 300 iterations, the resulting score is exactly the same but similar to the expected result.

When looking at the network parameters, we may see that parameters originally scored 1 have positive weight while other two have negative weight. This is normal because we are using sigmoid output, where positive weight gives an output closer to 1 while negative weight gives an output closer to 0. Therefore the result shows that the network weight is getting closer to the correct classification function.

There are several possible reason for the network unable to find the correct answer:

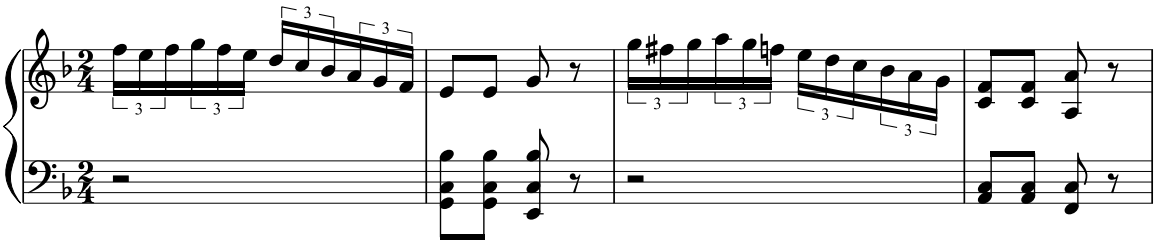
1. Training iterations are not enough.
2. The network reached local optimum and unable to escape from it.

6.1.3 Training with Expected Result of Different Segments

To show that the trained network is applicable to new input and able to generate result in similar manner, more training is done by providing expected result from segments different from the target segment.

The following example is the result of group A, measure 1-18, using the other two segments (measure 34-48 and 136-150) as training example.

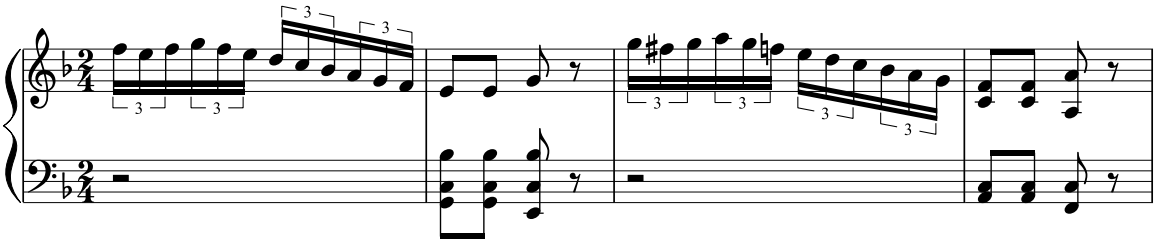
- Training with 10 iterations
  - First 4 Measures of Resulting Score



  - Network Parameters

```
[ 1.43407118 0.42803036 0.74140836 2.1928701 1.83016774 -1.21960974 0.93224384 -0.1151848 ]
```

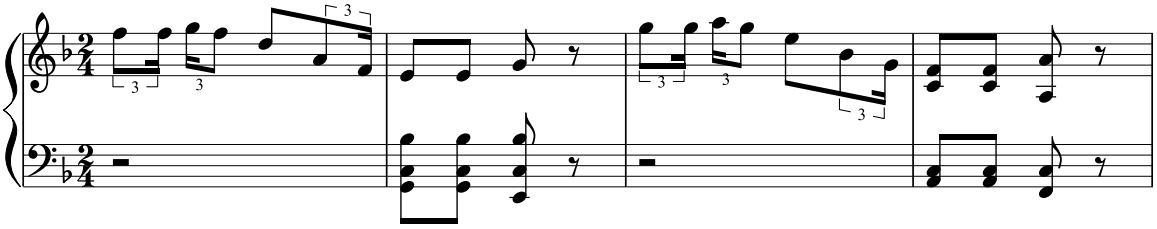
- Training with 50 iterations
  - First 4 Measures of Resulting Score



  - Network Parameters

```
[ 0.43416572 0.92570388 0.27562057 1.9563466 1.35633178 -1.75565586 0.7371129 0.55913304 ]
```

- Training with 300 iterations
  - First 4 Measures of Resulting Score



  - Network Parameters

```
[ -2.04535655 2.81184558 1.46985364 2.14204271 0.11961855 -0.89256043 0.14584181 2.78053821 ]
```

Given enough training epochs, even if the training samples did not contain the target score, the network is also able to predict parameter weights and threshold for classification.

The above result has another implications that only segments of the original score and reduced score are given as training samples, the network can already predict the correct parameter and generate correct result for other parts.

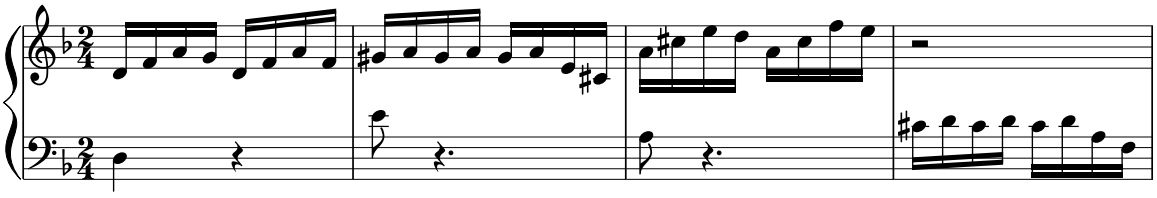
6.2 Examples Created by Pianist

This section will show the training result based on two examples provided by Lucas as training example:

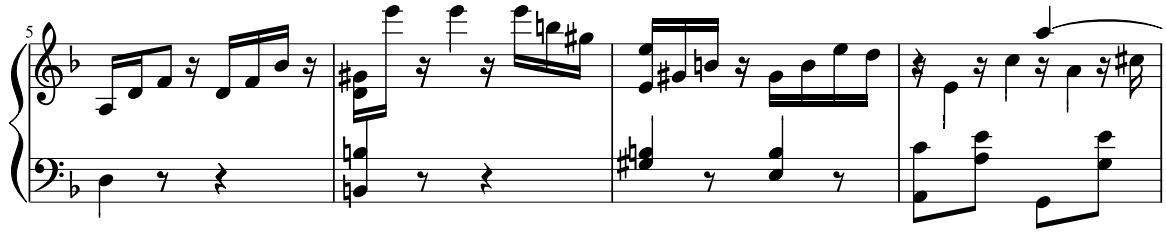
1. Keeping Important Entrance and Bass Line
2. Reduction by Removing Dissonances

6.2.1 Expected Results

- Important Entrance and Bass Line

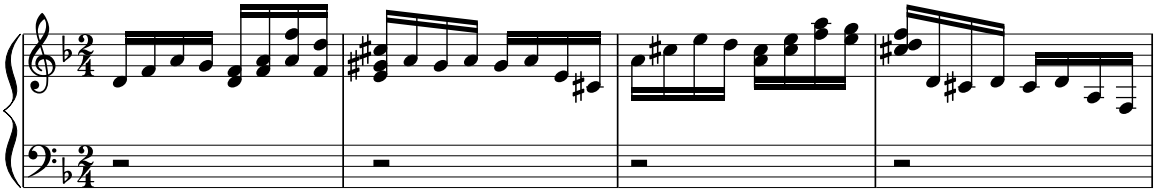


- Reduction by Removing Dissonances

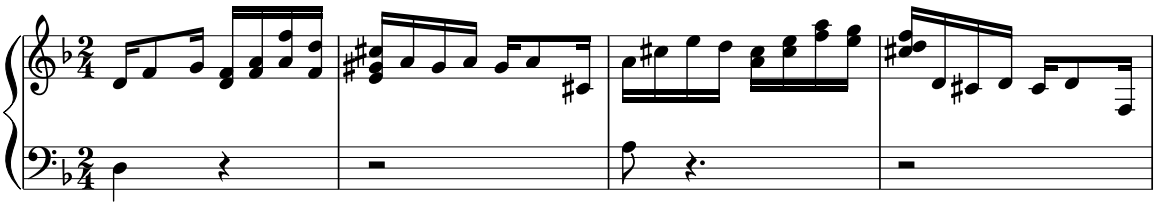


6.2.2 Training with Single-layer Perceptron

- Important Entrance and Bass Line

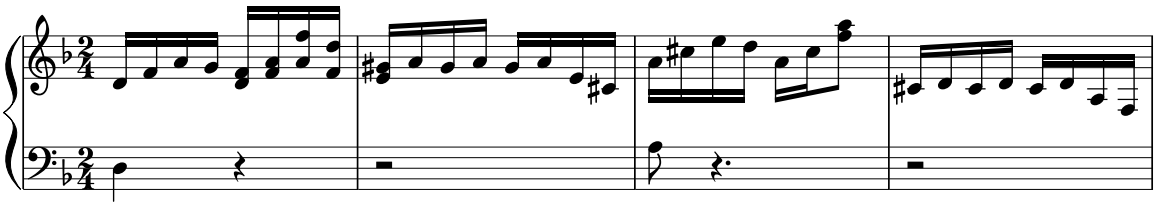


- Reduction by Removing Dissonances

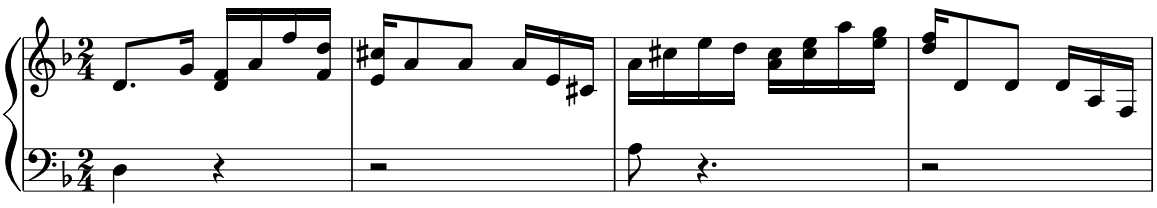


6.2.3 Training with Multi-layer Perceptron

- Important Entrance and Bass Line



- Reduction by Removing Dissonances



6.2.4 Summary

As shown in the result, single-layer perceptron is not sufficient for estimating parameter weights and threshold even after 1000 training epochs.

Better result can be achieved by using multi-layer perceptron, although it still cannot get the exactly same result, it is much better than single-layer.

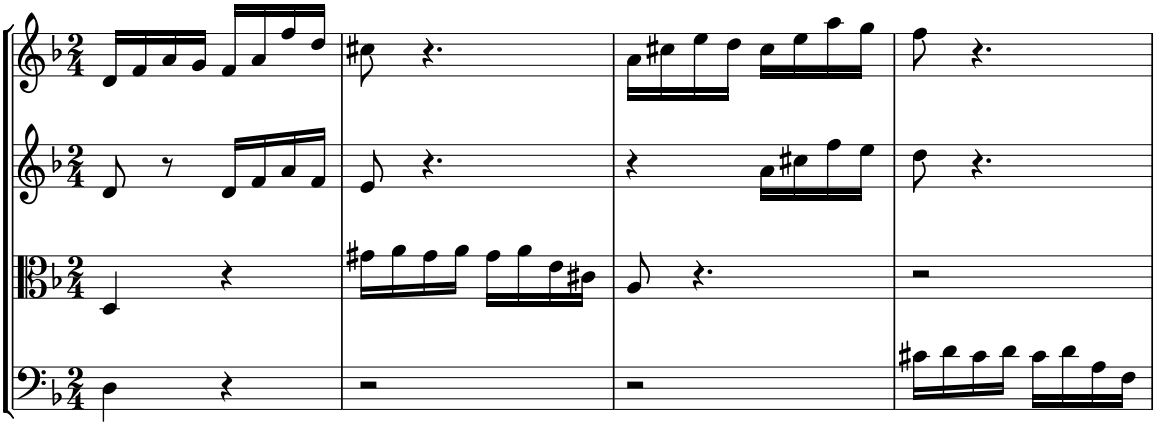
7 Generate Playable Piano Score

Generating a playable piano score is the final step for creating a piano reduction, and there are several criteria for the generated score:

1. Generated score should be a two-staff score.
2. Whether a note should be played by right hand or left hand should be determined based on its octave and other information available.
3. Notes from different parts should be merged into chords if they share the same rhythm.
4. Range of any resultant chord should not be too large, and a chord should not contains too many notes to be played simultaneously.

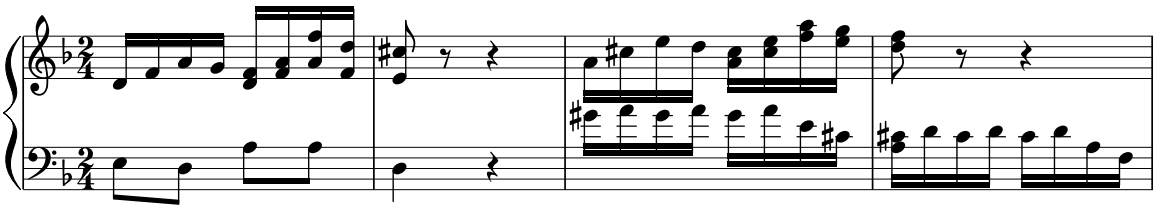
7.1 Previous Work

In previous implementation, users may choose two or more parts from the reduced score and merge them into one. For example, consider the following score which consist of four parts.



Users were asked to decide which parts should be merged into one, and there was no limit on how many parts that the final score may contain.

Based on above piece of score, one user may intuitively choose merge the first two parts originally played by violin to one part, and merge the last two parts played by viola and cello into another part.



This will generally gives a two-staff score as the user want. However, this requires user explicitly state which parts should be merged into one. While the program does not allow, sometimes the user may want to arrange notes to two staves by considering the range of each sentence separately.

There were several problems unresolved in previous implementation, some of them are solved in current implementation. Also, improvements were made to generate a better and more reasonable piano score automatically.

7.2 Redistribute Notes to Staves

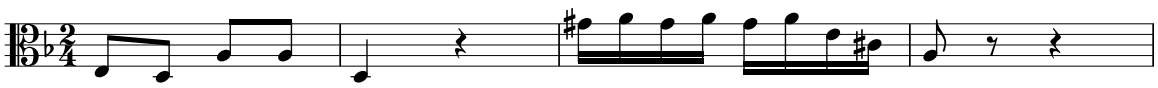
After classifying notes to be kept or not, the remaining score may still consists of more than two parts and we need to redistribute notes to two staves.

Generally, treble clef and bass clef are used in upper and lower staves of a piano score respectively, where the upper staff is played by right hand and lower staff is played by left hand.

In this section, we are going to discuss how to separate note into two staves.

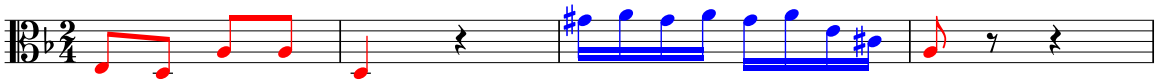
7.2.1 Basic Idea

Consider the following part originally played by viola.

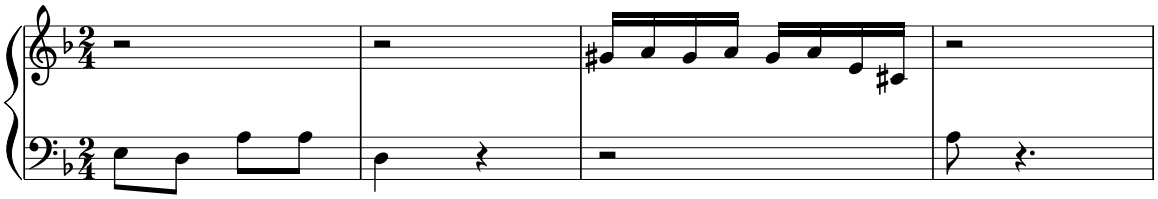


We can easily split notes into two parts by setting a threshold, e.g. middle C. Any note above the threshold will be put to the upper staff and vice versa.

The following diagram shows two classes, blue notes are above middle C and red notes are the opposite.

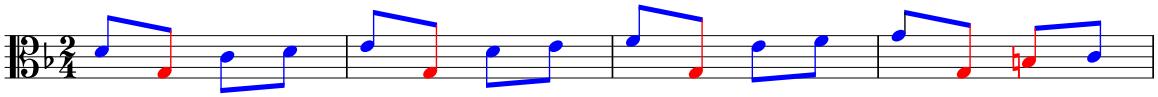


Blue and red notes are then separated into two staves, according to their pitch step, and the following diagram shows the result.

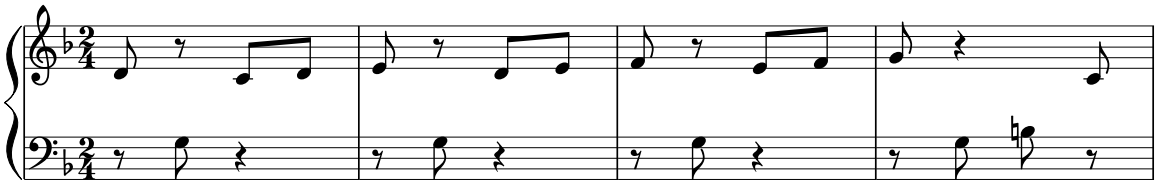


7.2.2 Alternating Sequence

The method proposed above was simple and effective, however, it cannot cope with the situation that the sequence of notes are alternating around middle C.



This is another few measures originally played by viola, note color indicates whether a note is above middle C or not.



The above generated score, though there is no difference to the audience, it is quite awkward for the pianist to play, as all notes belong to the same melody.

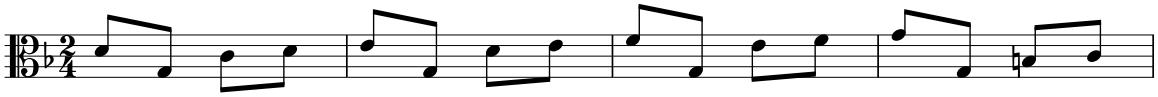
7.2.3 Final Solution

Previous sections described a simple and effective method for distributing notes into two staves, also problems encountered when the sequence alternate around middle C.

Here shows a modified version of the previous method, which is able to cope with the problem of alternating sequences.

Instead of looking at notes one by one, we consider all notes in a measure at one time.

Consider the same score as presented in previous section, where the note sequence is alternating around middle C.



In each measure, construct a list of pitch step as follows:

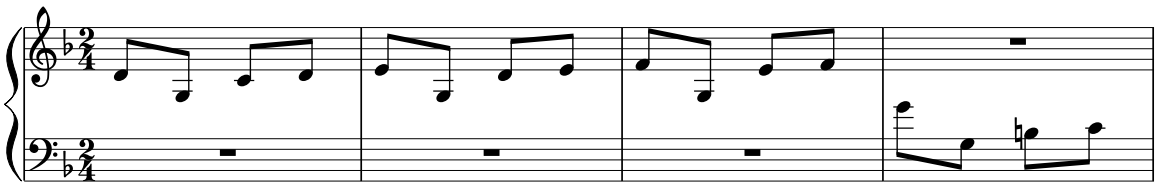
```
M[1] = [ D4(62), G3(55), C4(60), D4(62) ]
M[2] = [ E4(63), G3(55), D4(62), E4(63) ]
M[3] = [ E4(63), G3(55), D4(62), E4(63) ]
M[4] = [ G4(67), G3(55), B3(59), C4(60) ]
```

Note that in the actual implementation, only numerical value of pitch step is considered.

Then compute the median of pitch steps for each measure, and we get another value used to determine whether notes inside a measure should be put to upper or lower staff.

```
median(M) = [ 61, 62.5, 62.5, 59.5 ]
```

We are still using middle C as the threshold, compare the median of pitch steps of a measure with middle C (60), if the median is larger than 60, all notes within that measure will be put to upper staff and vice versa.



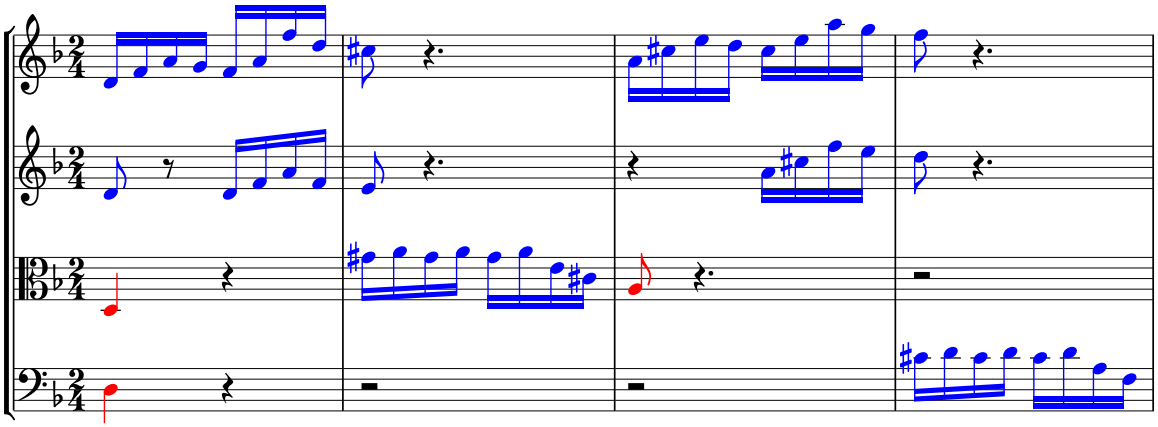
Compare the above result with previous one, the final solution generates better-looking score by considering notes in a measure as a unit for classification.

7.3 Merging Notes

For each part in the reduced score, we may apply the same algorithm for classifying notes into two staves. After that, we obtain a collection of notes that should be played by two hands.

In the same staff and same measure, it may consists of notes originally coming from different parts. Similarly, notes originally from the same part but different measures, they may be separated to two staves based on their median.

Consider the following score, the same piece of score presented in previous work, and notes are already classified into two groups, correspond to right hand and left hand according to the classification algorithm mentioned.

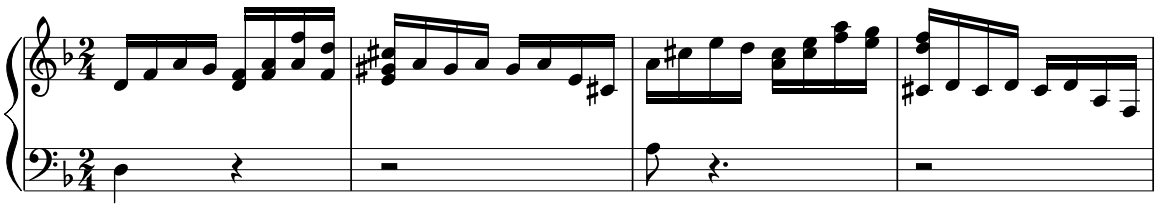


Notes with the same color are grouped together and they will be put into the same staff. This means that for a particular measure within a staff, it contains a collection of notes that are put into it according to the classification algorithm.

Algorithm for merging is the same as previous implementation, the only difference is notes are classified automatically, and notes originally in the same staff may split into two parts. And the following are the steps for merging:

1. Separate notes into two staves using previous method.
2. For the same staff, merge notes having the same offset into a chord.
3. At the same time, truncate the duration of notes that will cover the intervals of the next note.
4. Extend notes or chords to cover all time intervals.
5. Add back rest notes to time interval that are originally at rest in all part that should be merged into the same staff.

The following score shows the final result of auto-generated piano score based on above classification results.

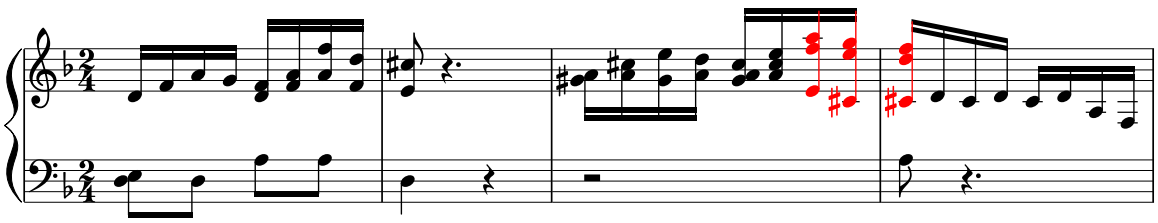


It is obvious that some chords are too hard to play by any human, the following section will vertify the playability of generated score.

7.4 Verify Playability

After merging, although a two-staff piano score is generated, it may not be playable by any human because of unreasonable range of certain chords.

The following piece of score is generated in previous step, range of red chords are too large to be played.



In this section, we are going to discuss about some problems that we may encounter when verifying the playability of auto-generated score, and the possible solution to those problems.

7.4.1 Chords with Large Range

As discussed, one problem of merged score is that some chords are not playable because of large range.

Consider only the third, we may collect the statistics of pitch step of all notes.

```
M[3] = [ A4(69), G#4(68), C#5(73), A4(69), E5(76), G#4(68), D5(74), A4(69), C#5(73), A4(69),
         G#4(68), E5(76), C#5(73), A4(69), E4(64), A5(81), F5(77), E5(76), C#4(61), G5(79) ]
```

The same routine still applied to every measure and every staff, but there would be nothing changed after processing for some of them, the above illustration focus on the third measure in upper staff only to show how the notes will change under this method.

Also, to simplify the explanation, the two list of pitch steps are sorted in accending order of the numerical value representing it, and names of pitch are omitted. Median is also computed as it will used as a reference for modifying notes.

```
M[3] = [ 61, 64, 68, 68, 68, 69, 69, 69, 69, 69, 73, 73, 73, 74, 76, 76, 76, 77, 79, 81 ]
median(M[3]) = 71
```

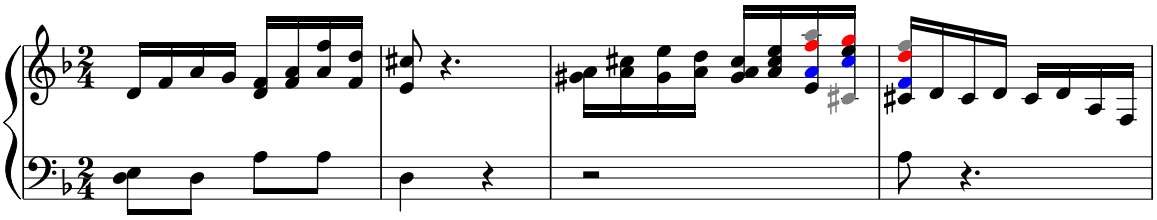
The next step is to find a chord that is unplayable, simply find the range of pitch steps within a chord, and it is easy to check if the range of chord exceeds an octave. And there are two chords with this property in the third measure.

```
c1 = [ E4(64), F5(77), A5(81) ]
c2 = [ C#4(61), E5(76), G5(79) ]
```

In an unplayable chord, find a pitch step which is farthest away from the median, and move exactly one octave towards the median ± 12. The following lines show the modified pitch steps in two chords.

```
c1 = [ E4(64), A4(69), F5(77) ]
c2 = [ C#5(73), E5(76), G5(79) ]
```

Modified chords are shown below, where gray notes are moved to blue notes.



Another problem quicky arise, the melody of this few notes changed. The original melody of colored chords were '↘ ↘', but it is changed to '↗ ↘', and this is an unpleasant effect.

7.4.2 Changing Melody

To avoid melody changed by the above method, the algorithm need to be modified in a way that the highest pitch is always the same.

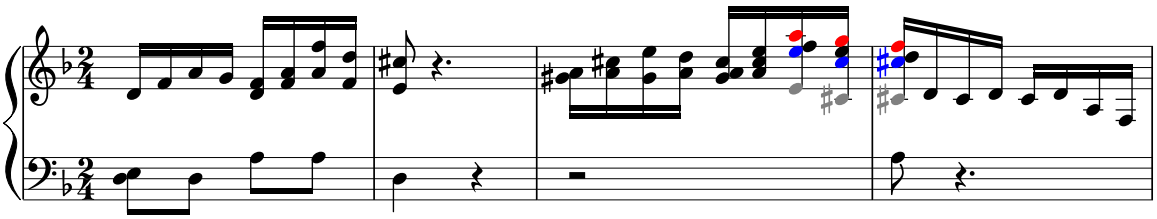
The same piece of score is used again as an example, and the two chords in the third measure with the range problem are as follows:

```
c1 = [ E4(64), F5(77), A5(81) ]
c2 = [ C#4(61), E5(76), G5(79) ]
```

Instead of finding a note that is farthest away from the median, the lowest pitch is moved up by an octave. And the result is shown below:

```
c1 = [ E5(76), F5(77), A5(81) ]
c2 = [ C#5(73), E5(76), G5(79) ]
```

The final score is shown in following diagram. Red notes represent the melody, and it has remains unchanged after modification to chord notes.



8 Conclusion

This project aimed to provide users with different reduction parameters and allow user manually adjust weights and a threshold to perform automatical piano reduction.

Though this apporach did not violate the original target: user-adjustable, as a computer scientist, we could easily imagine this is not feasible to normal user, especially potential users of the final product, musicians.

Another approach was tried to offload more work to computations and allow easier and more inituative input methods for user. By using some simple machine learning techniques, we allow the computer to guess how the user make decision on different notes, deciding which note should be kept and which should not.

As discussed learning results, while the program was able to estimate reduction parameters if we know there must be an answer in the given example, while it cannot perform well for training examples created by pianist, we may conclude that if our program has sufficient parameters, the program is able to estimate different weights or even more complex decision making strategy based on training examples provided by users.

However, the capability for the program to estimate how a user make decisions on differnt notes is limited by the completeness of the possible parameter set for performing piano reduction. And it is also limited by the neural network complexity.

Finally, no matter how well or bad was the trained program was, it may help the user to perform piano reduction for the whole score or even other songs. And finally, a playable piano score is automatically generated without user's assistance.

Although at the final stage of this project, we still cannot achieve good result based on users' examples, at least we could believe that if we have more parameters implemented in the future, and have a more complete set of parameters, the program should be able to learn from examples and generate reduced score which its style should be similar to given examples.

9 Future Works

Some possible future works are listed below:

1. More possible reduction parameters.
2. Identification of chords and harmony before passing statistical information to the training program.
3. Tranformation of sheet music by retaining only musical notes in harmony with chords, or replacing non-chord tones with chord tones.
4. Interactive graphical user interface for input and output, and possibly intermediate actions.

10 References

10.1 Acknowledgements

This project was co-supervised by Professor Lucas Wong. As a professor of piano, he is able to propose many reduction paraemters that actually shows how musicians perform piano reduction. This project cannot be done without his assistance.

10.2 Open-source Libraries

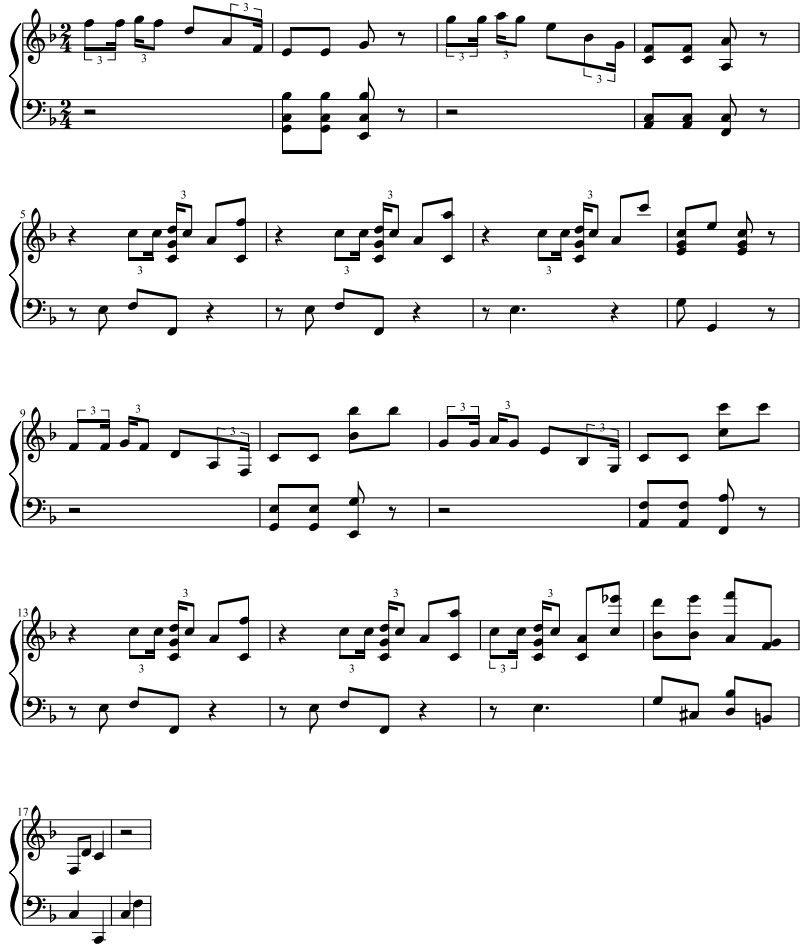
This project is implemeted with the help of several open-source libraries:

- MusicXML: the standard open format for exchanging digital sheet music.
  `http://www.musicxml.com`
- music21: a toolkit for computer-aided musicology
  `http://web.mit.edu/music21`
- pybrain: a modular machine learning library for Python
  `http://pybrain.org`
- numpy: the fundamental package for scientific computing with Python
  `http://www.numpy.org`

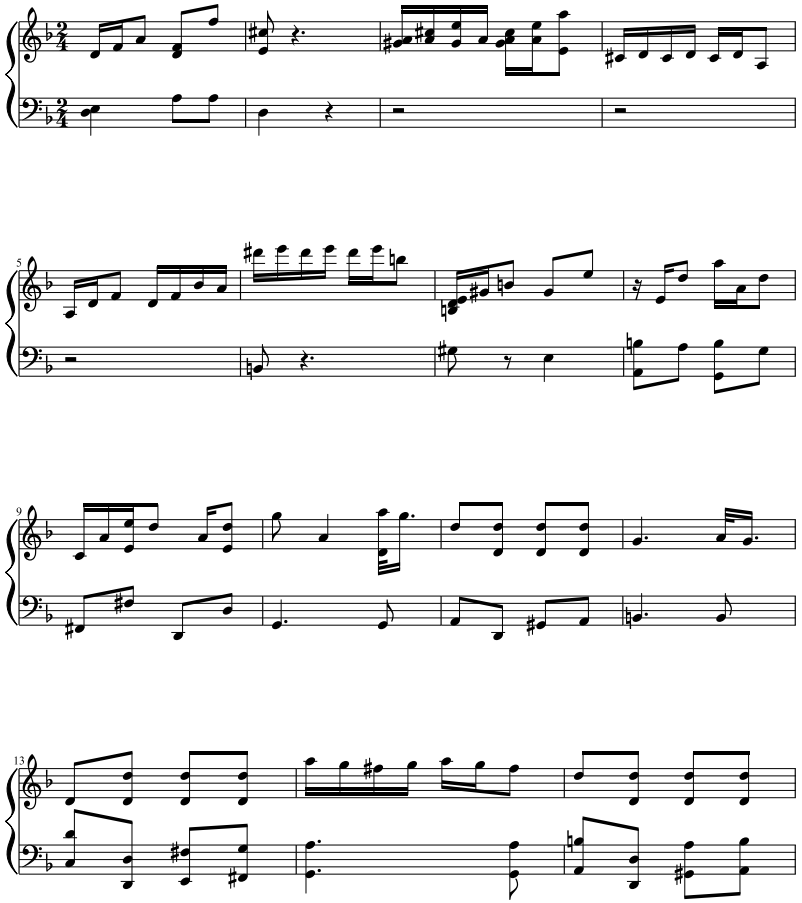
A Generated Test Data

A.1 Group A

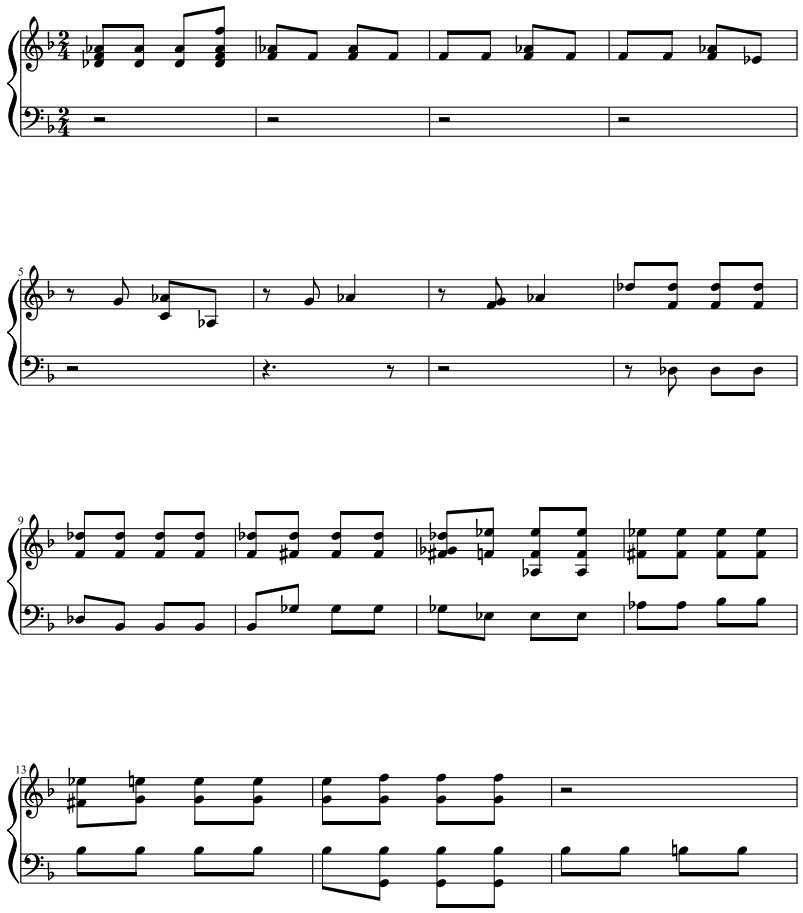
A.1.1 Measure 1-18



A.1.2 Measure 34-48

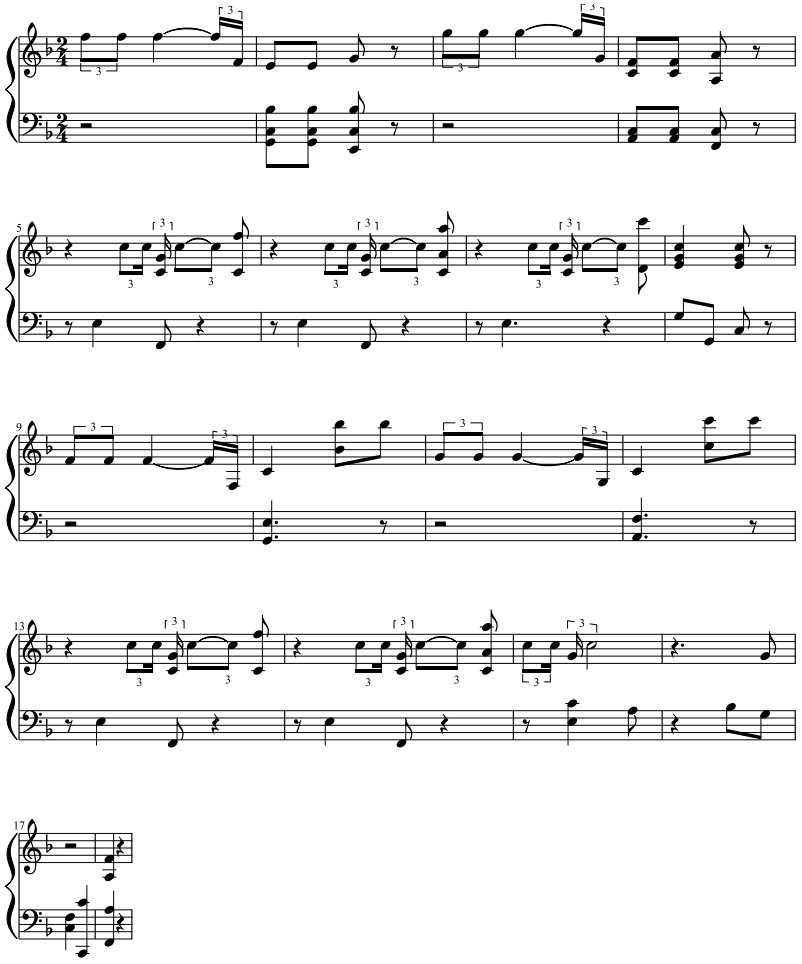


A.1.3 Measure 136-150

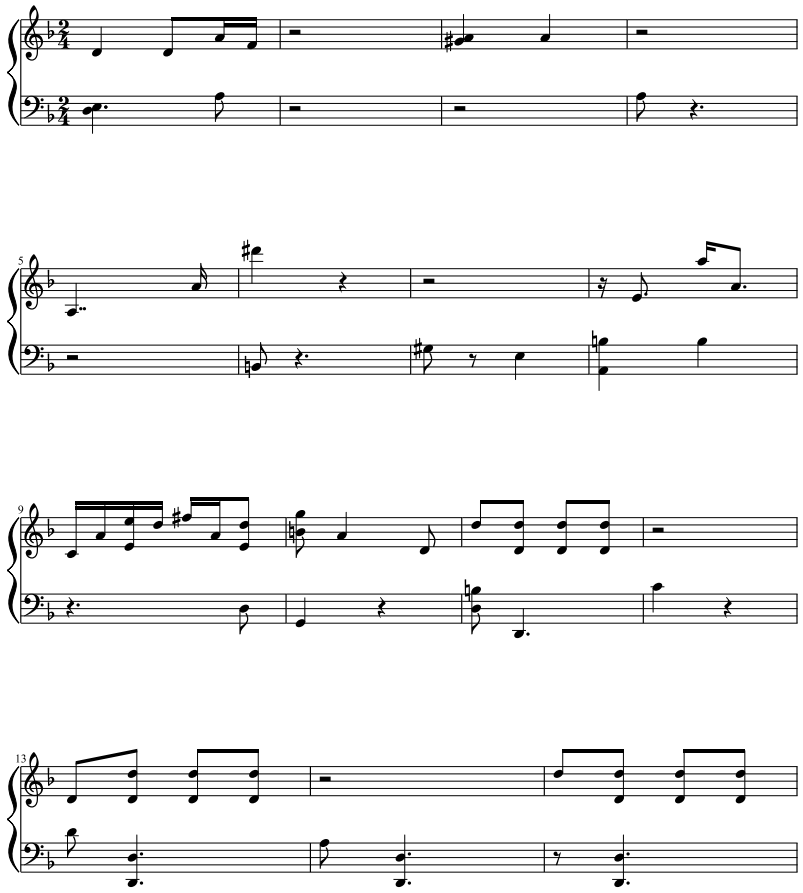


A.2 Group B

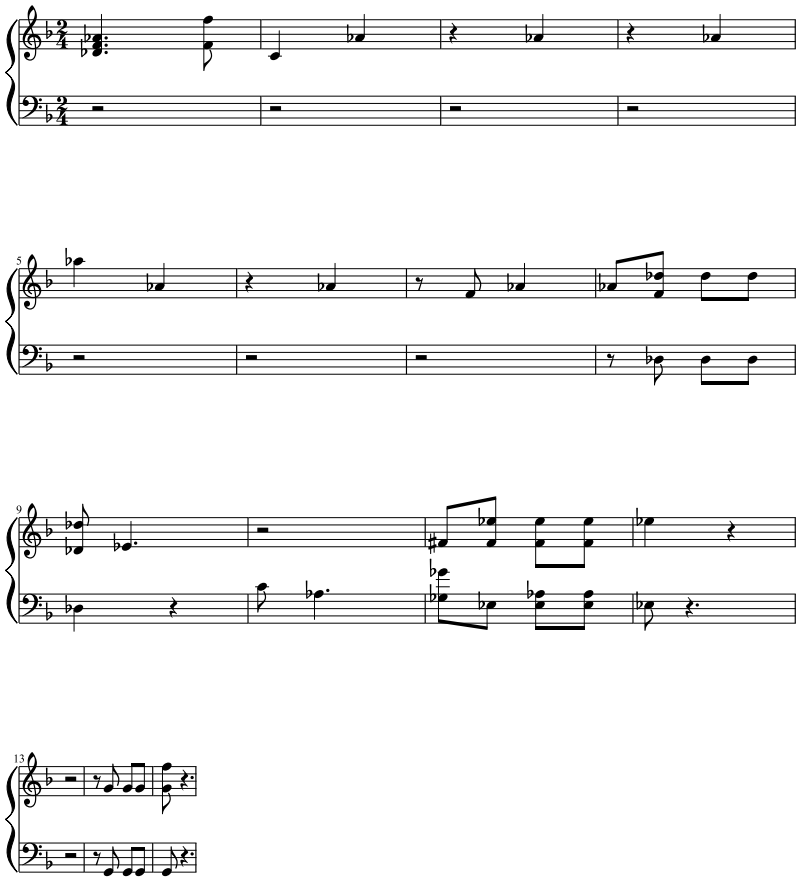
A.2.1 Measure 1-18



A.2.2 Measure 34-48



A.2.3 Measure 136-150

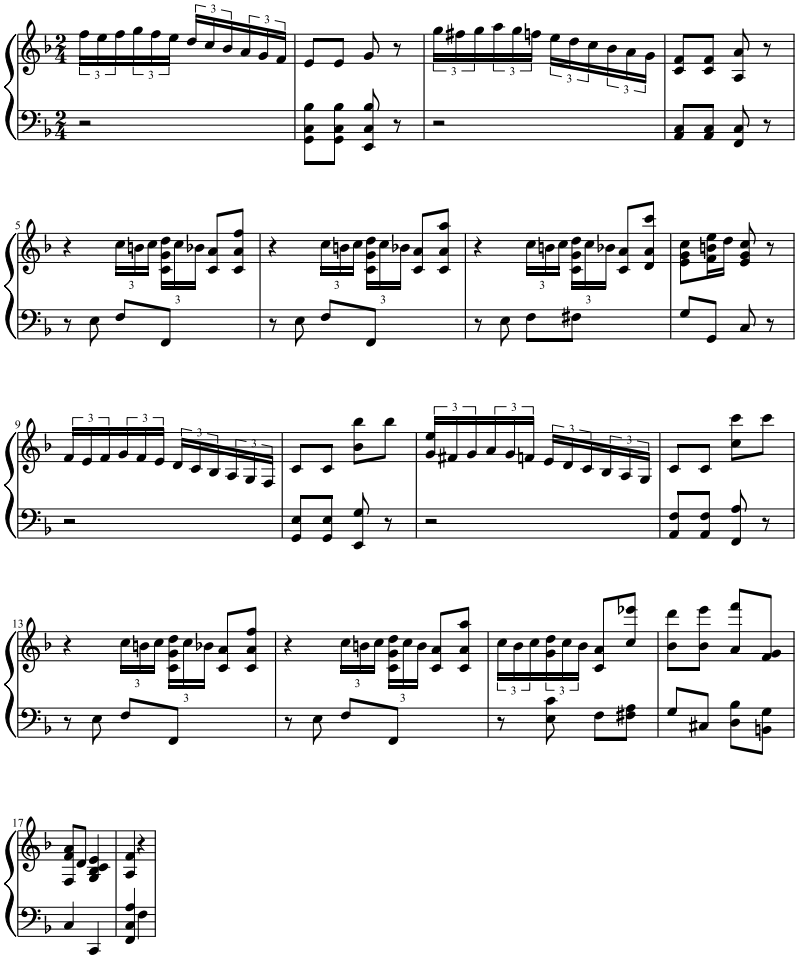


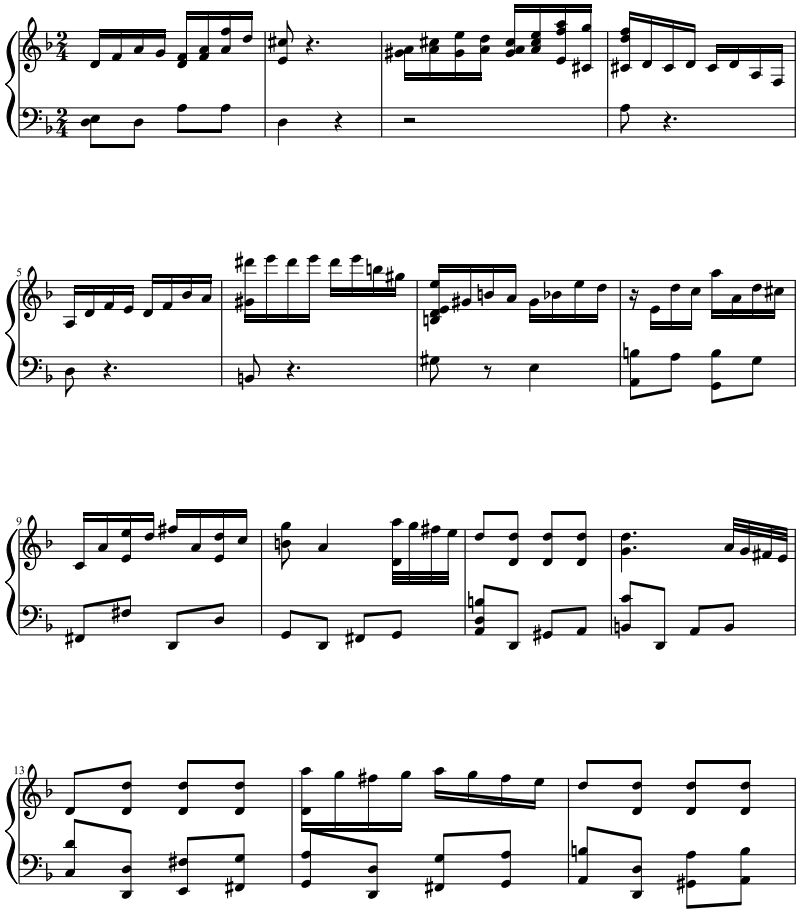
B Detailed Training Results

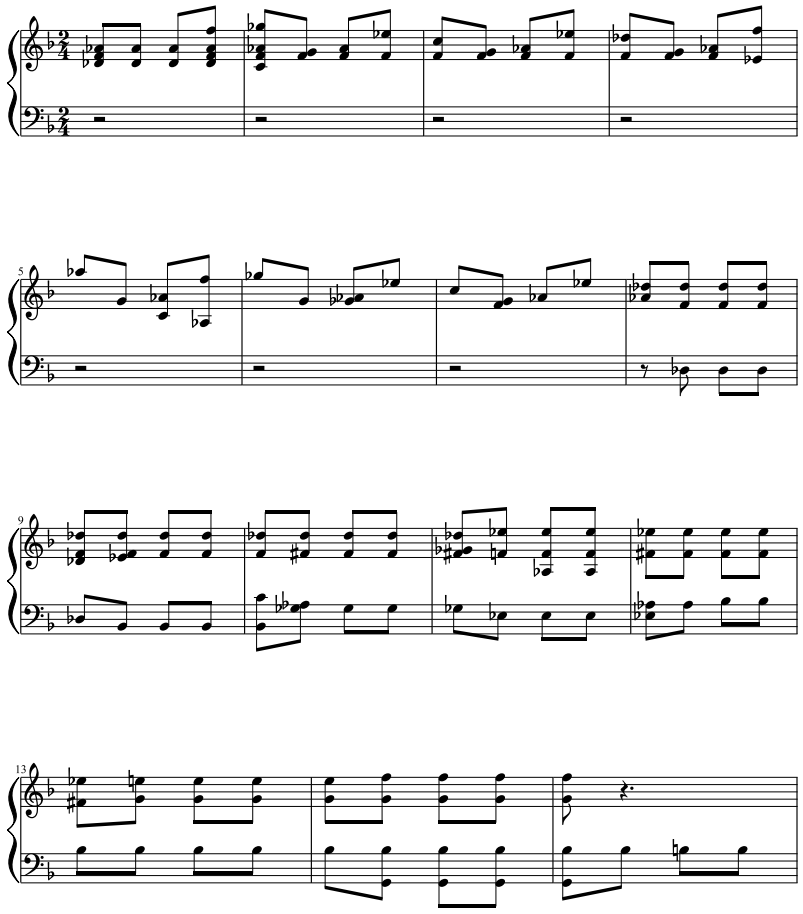
B.1 Examples with Known Answer

B.1.1 Group A

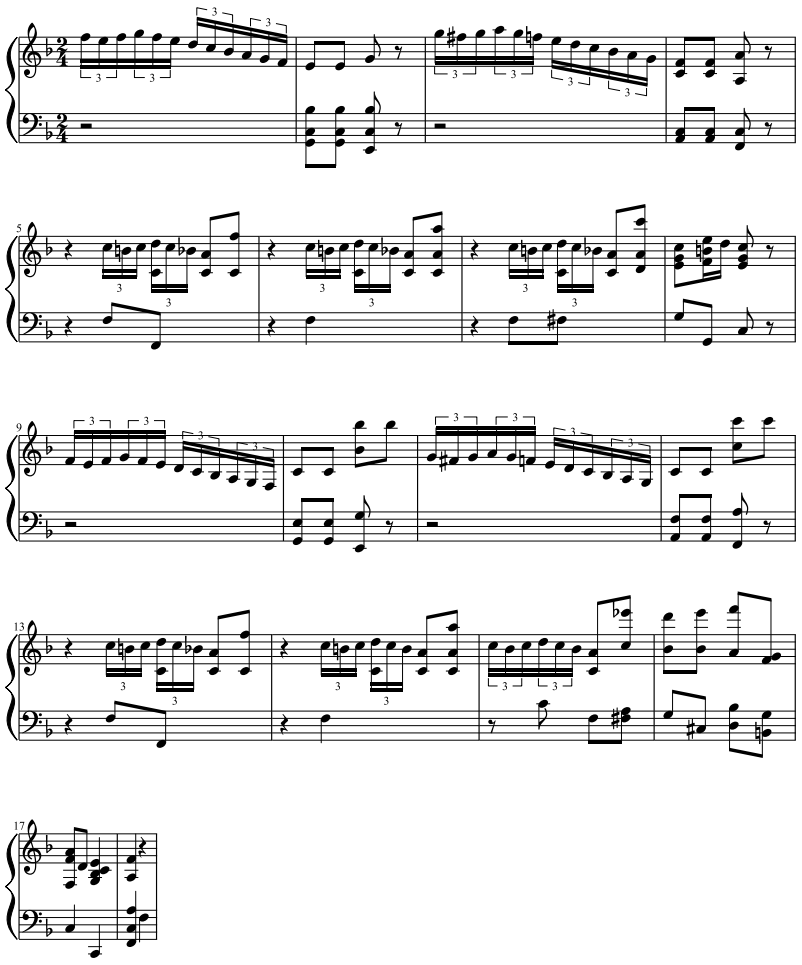
- Training with 10 iterations

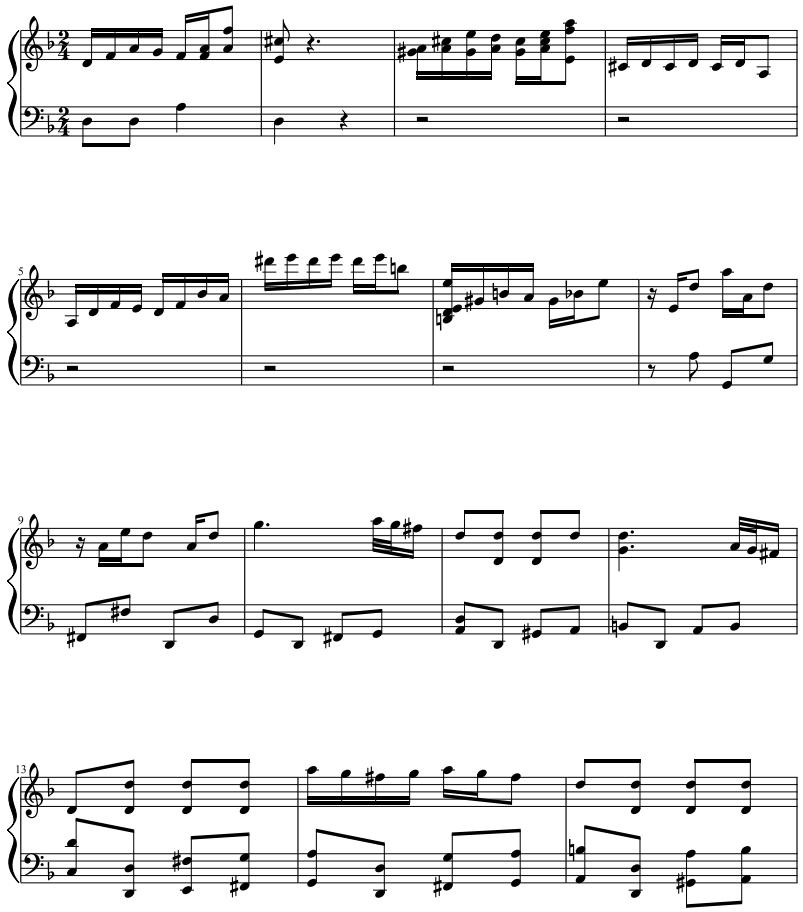


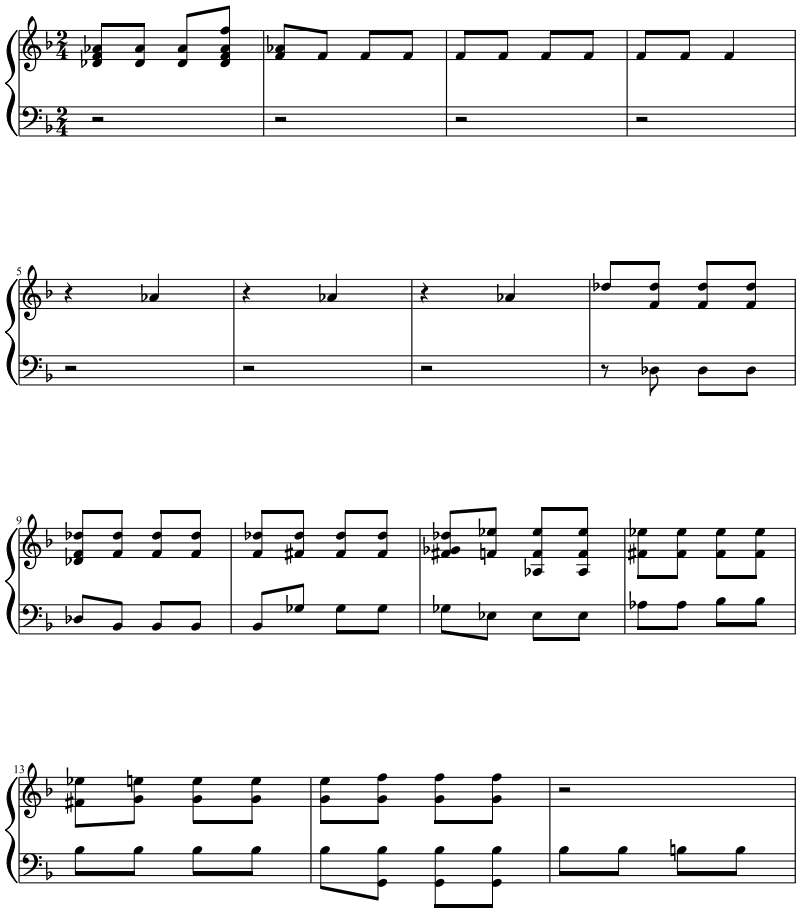




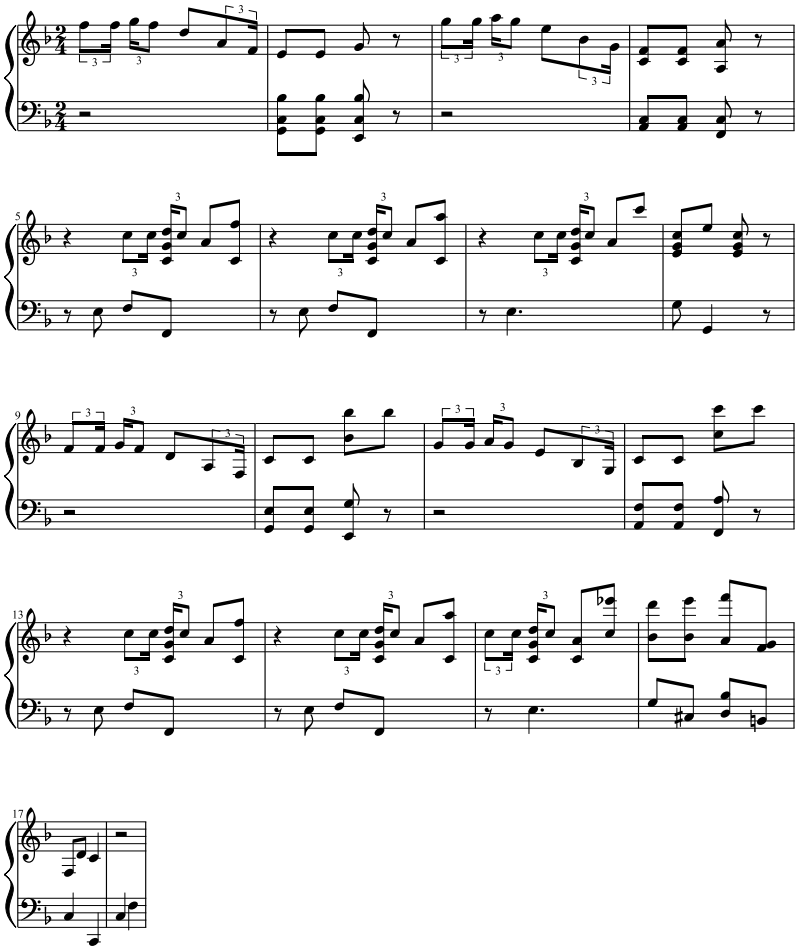
- Training with 50 iterations

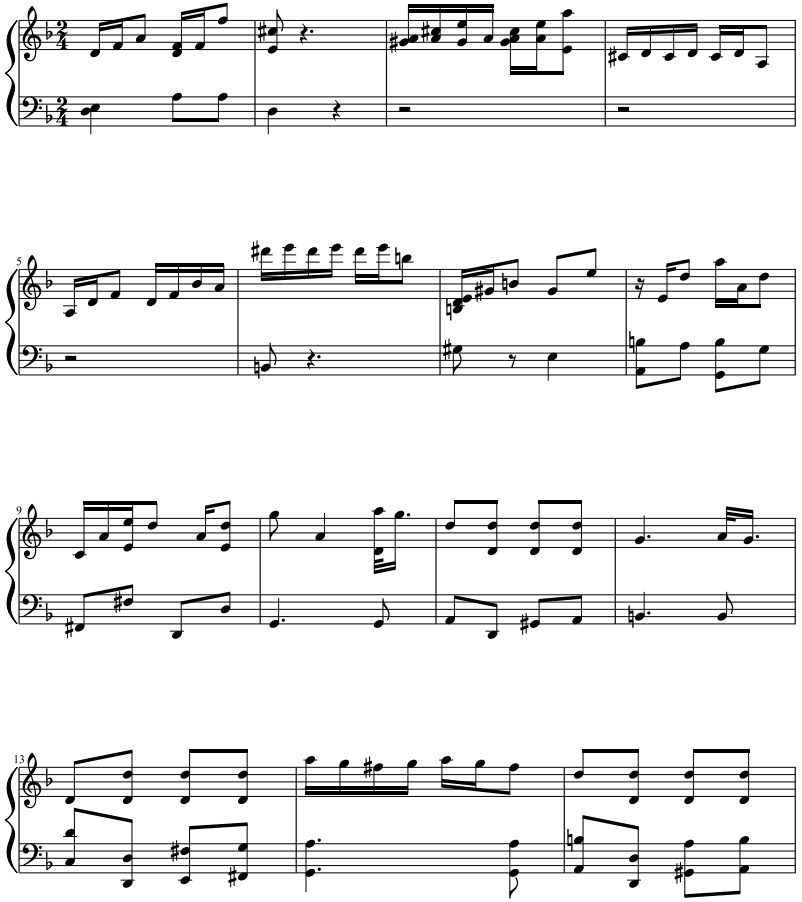


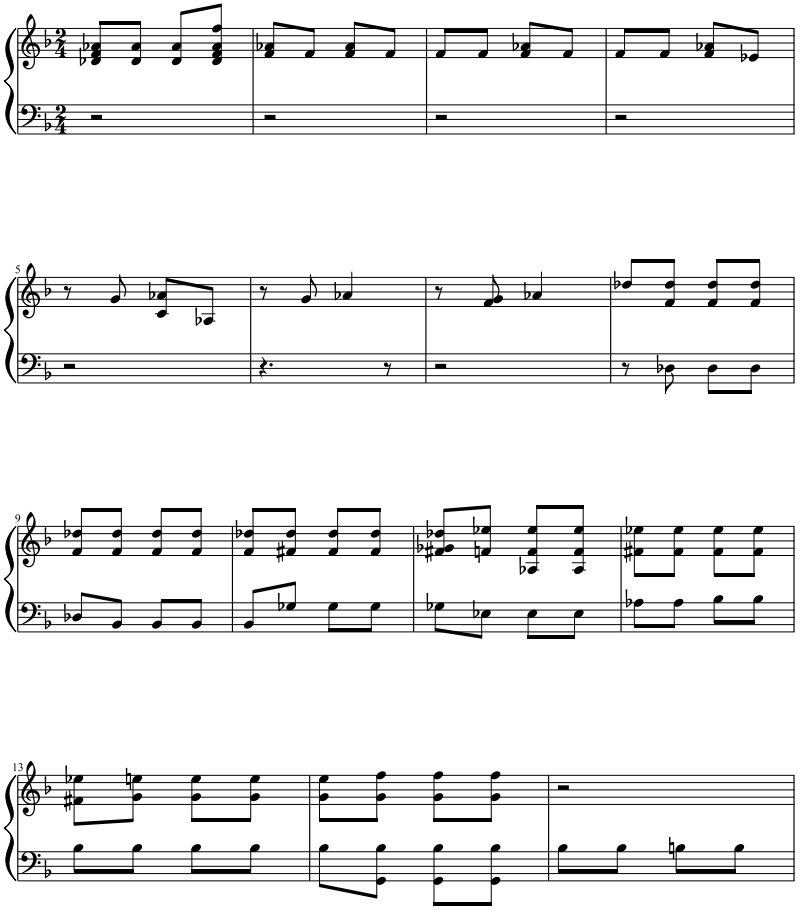




- Training with 300 iterations

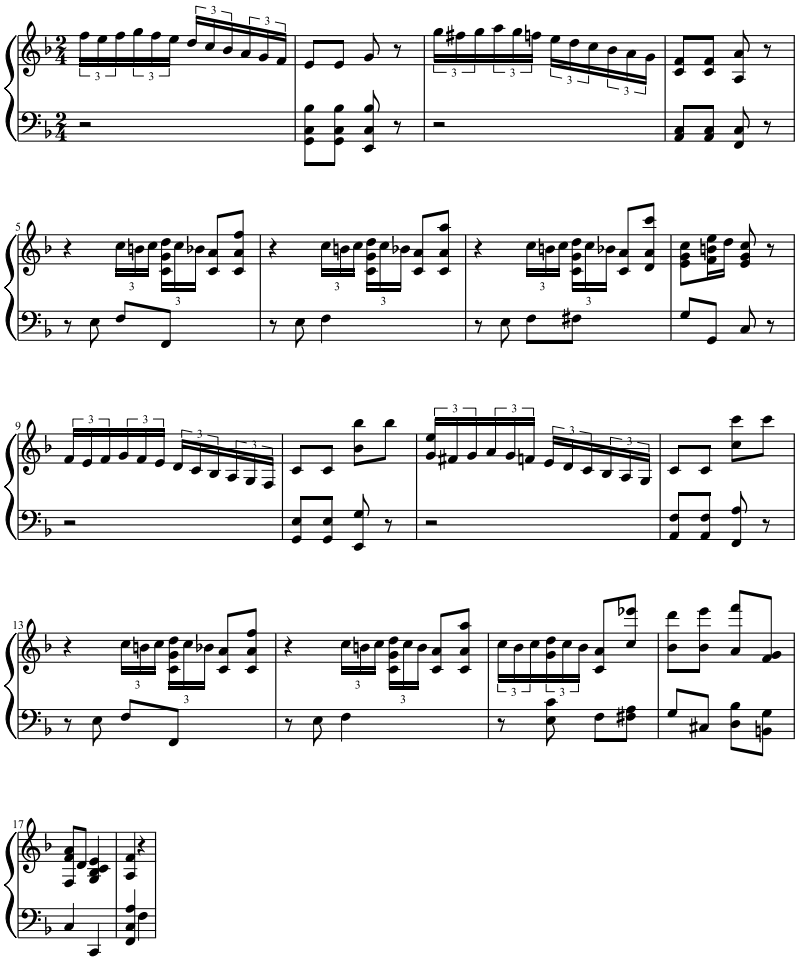


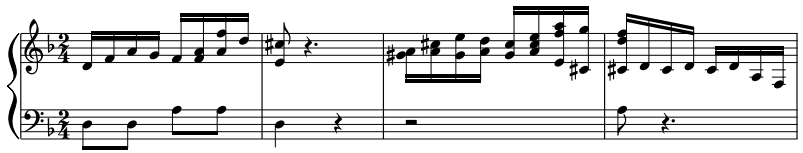


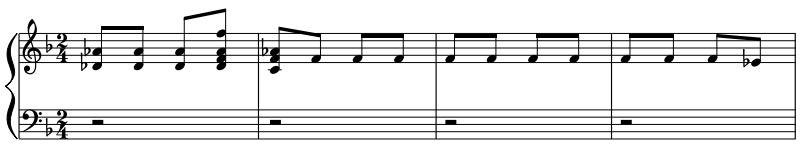


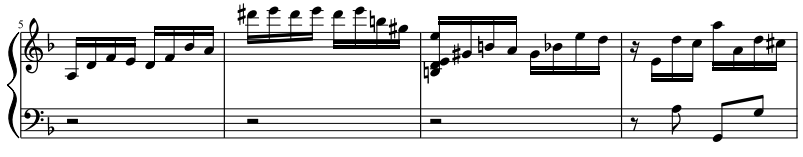
B.1.2 Group B

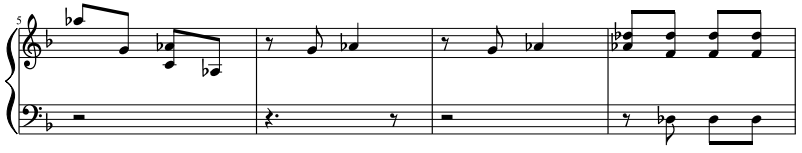
- Training with 10 iterations

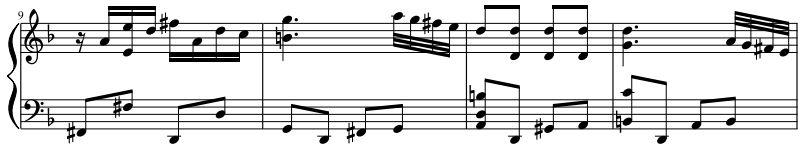


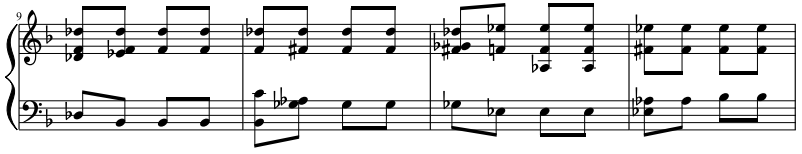


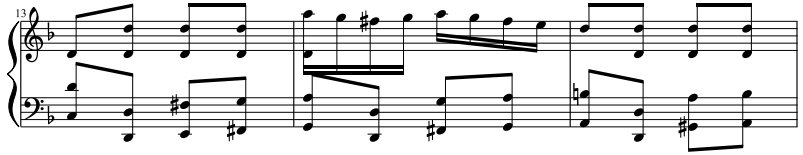


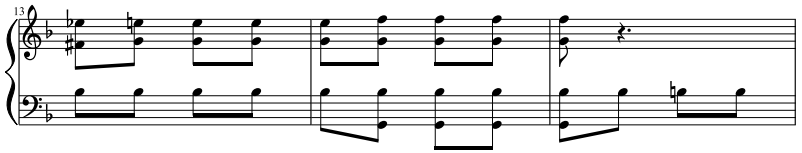




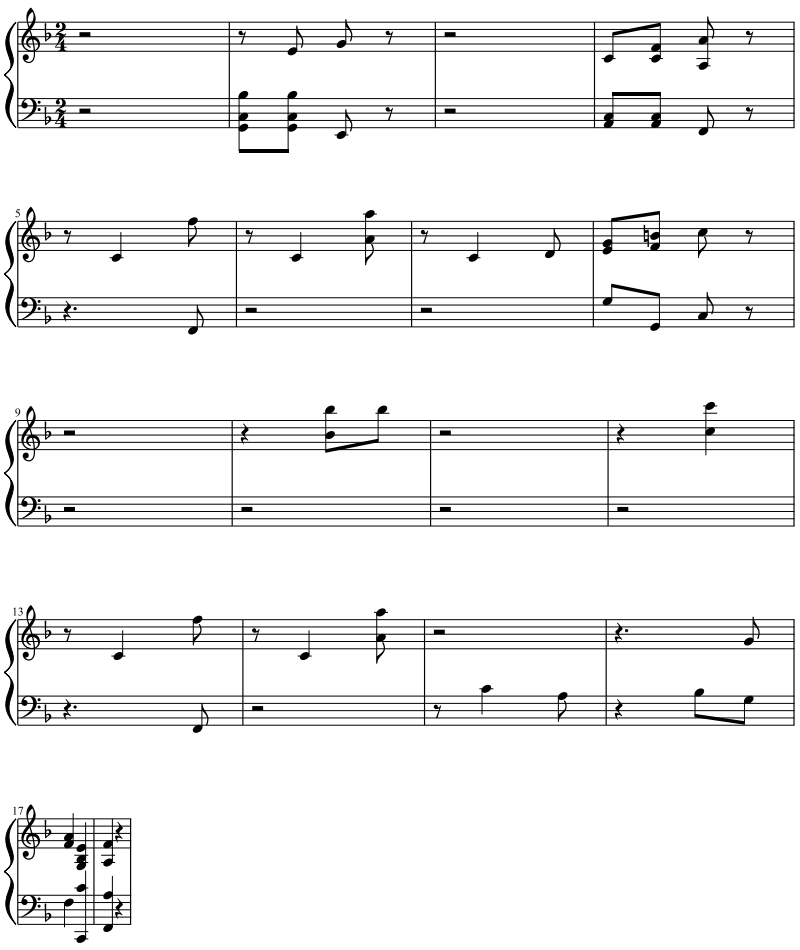


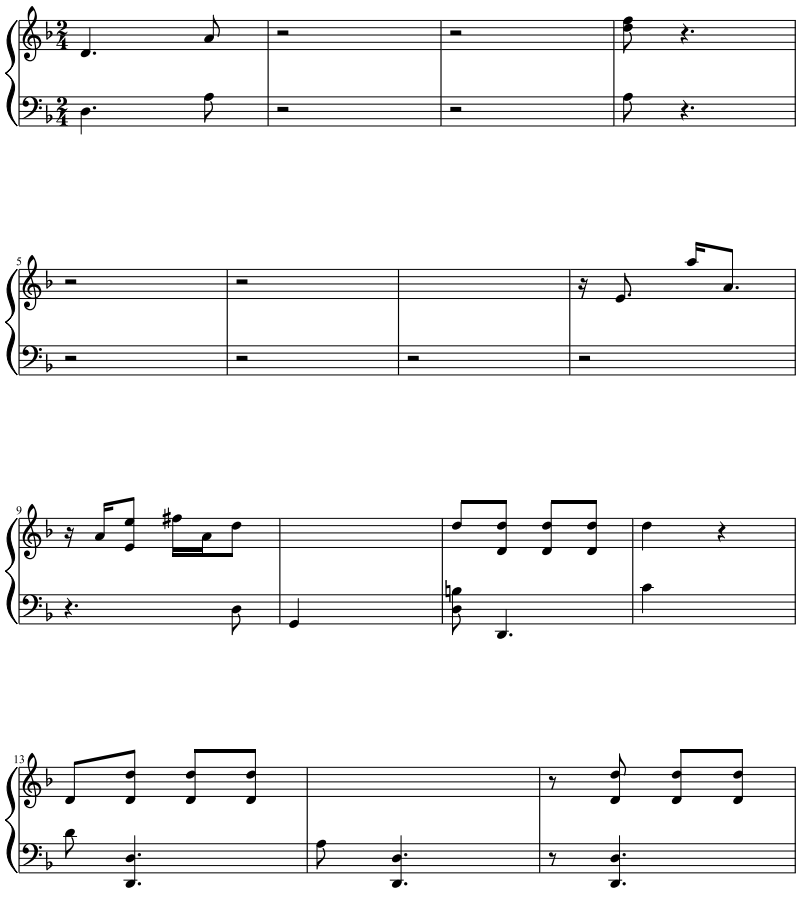


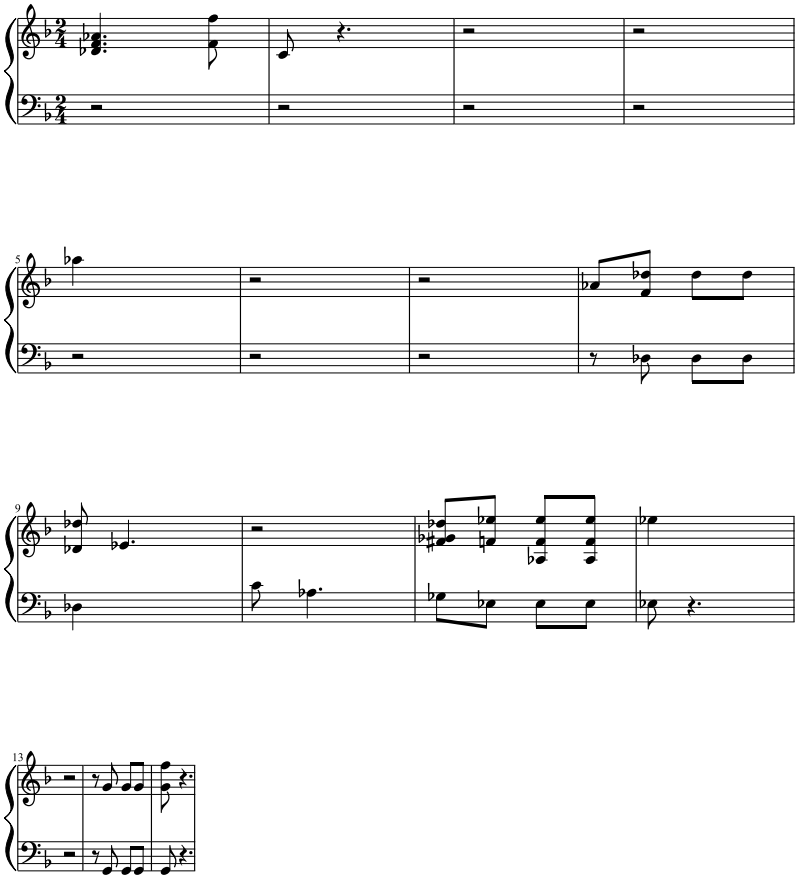




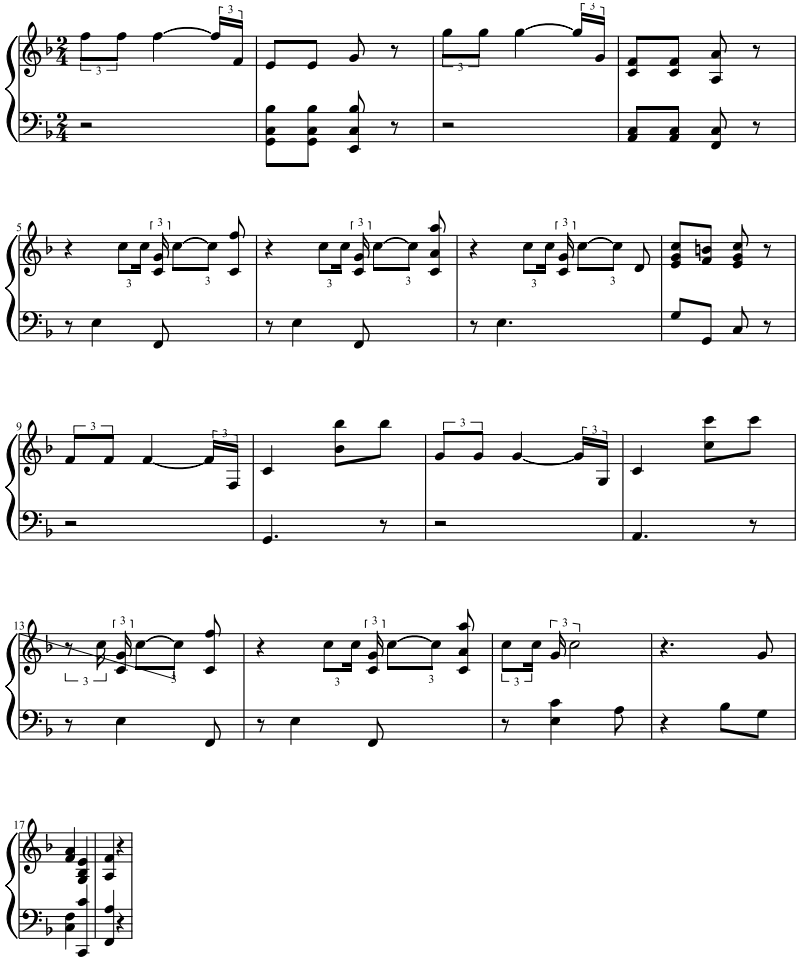
- Training with 50 iterations

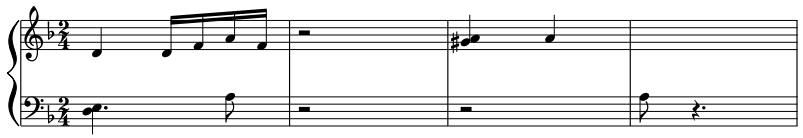


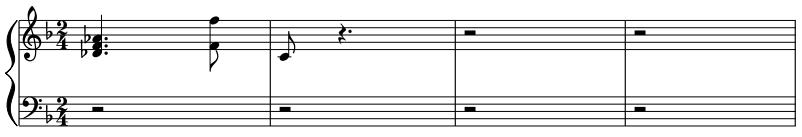


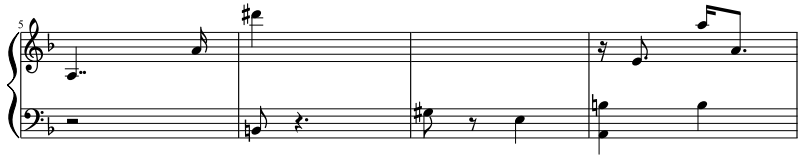


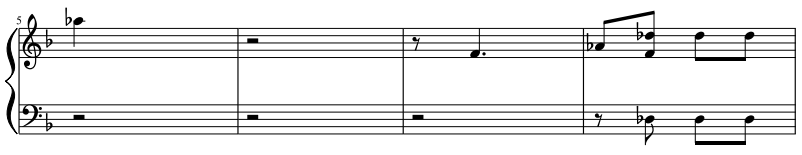
- Training with 300 iterations

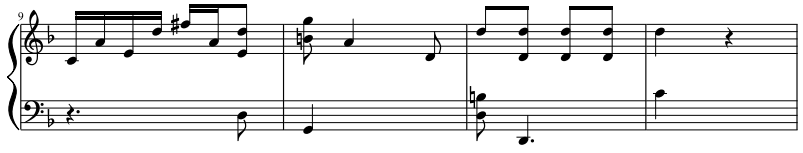


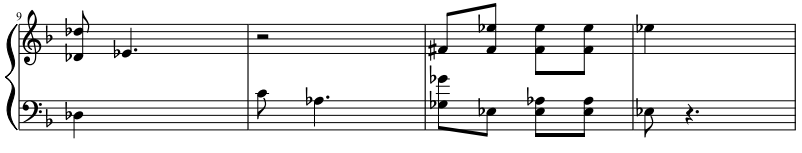


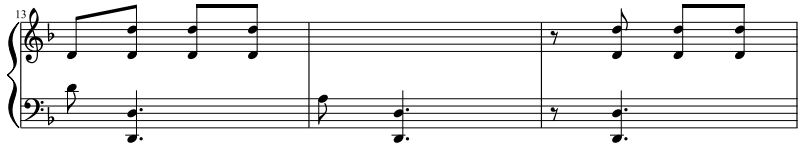


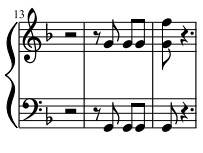

5


5


9


9

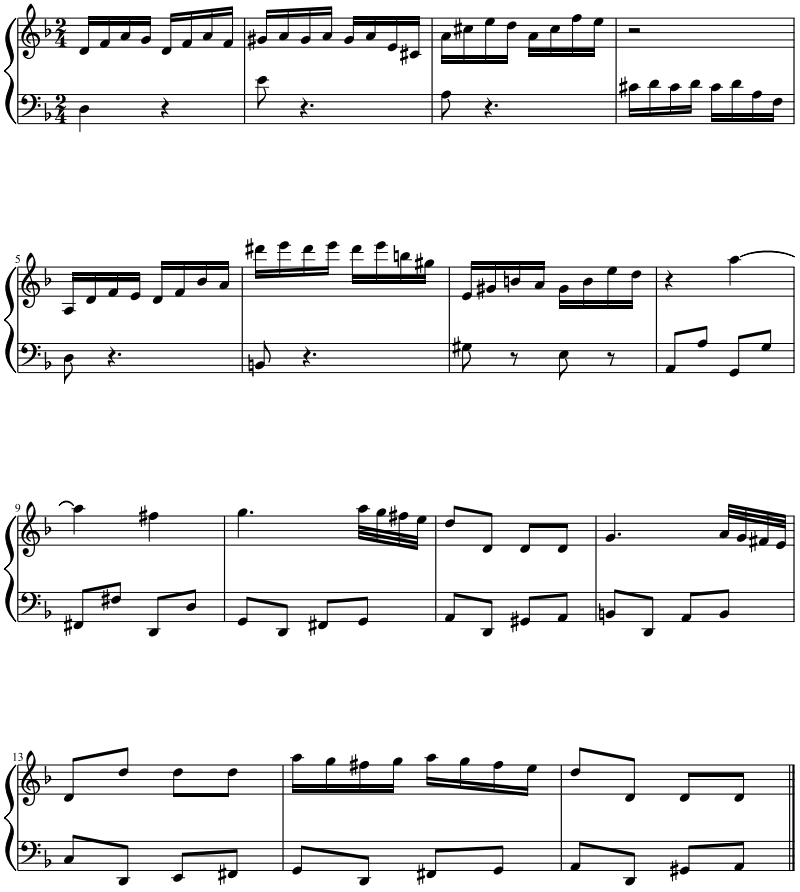

13


13

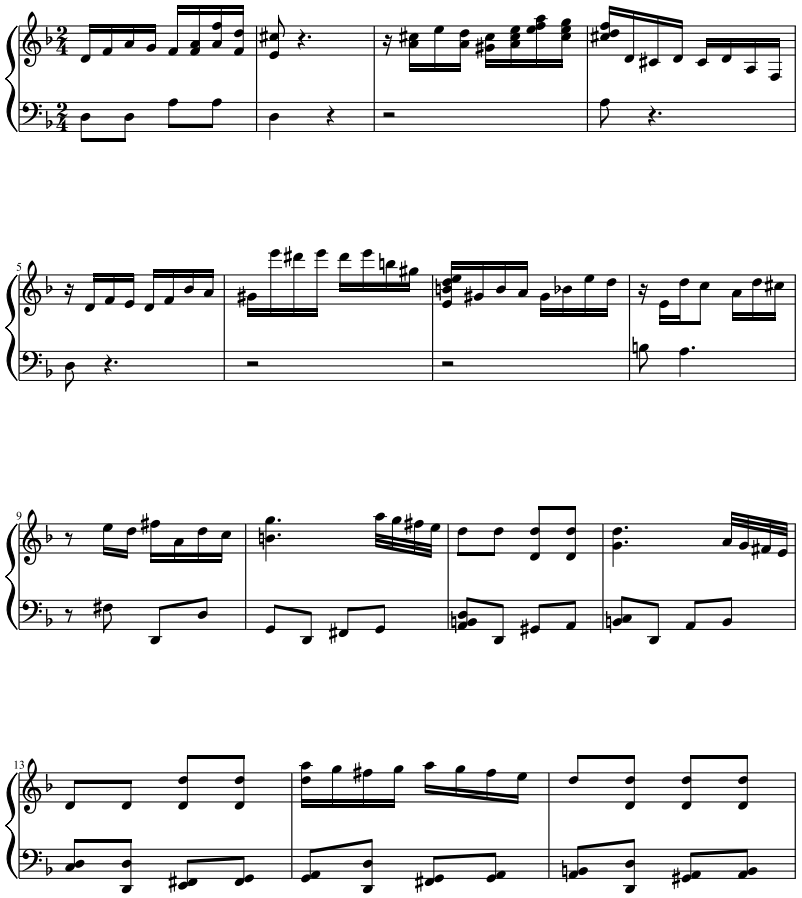
B.2 Examples Created by Pianist

B.2.1 Keeping Important Entrance and Bass Line

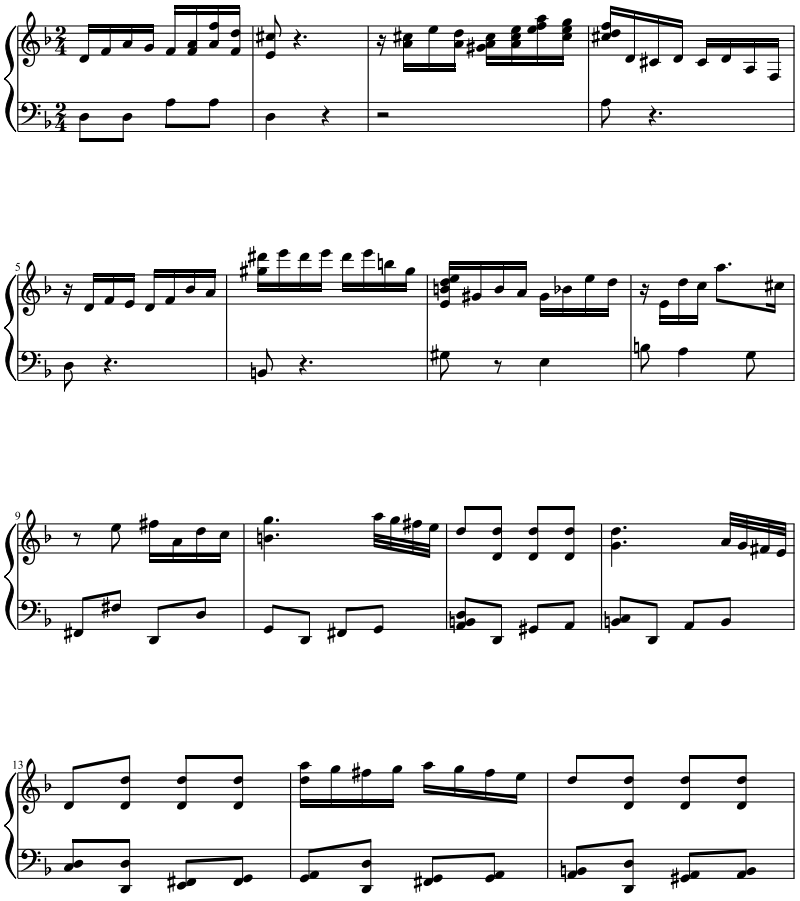
- Expected Result



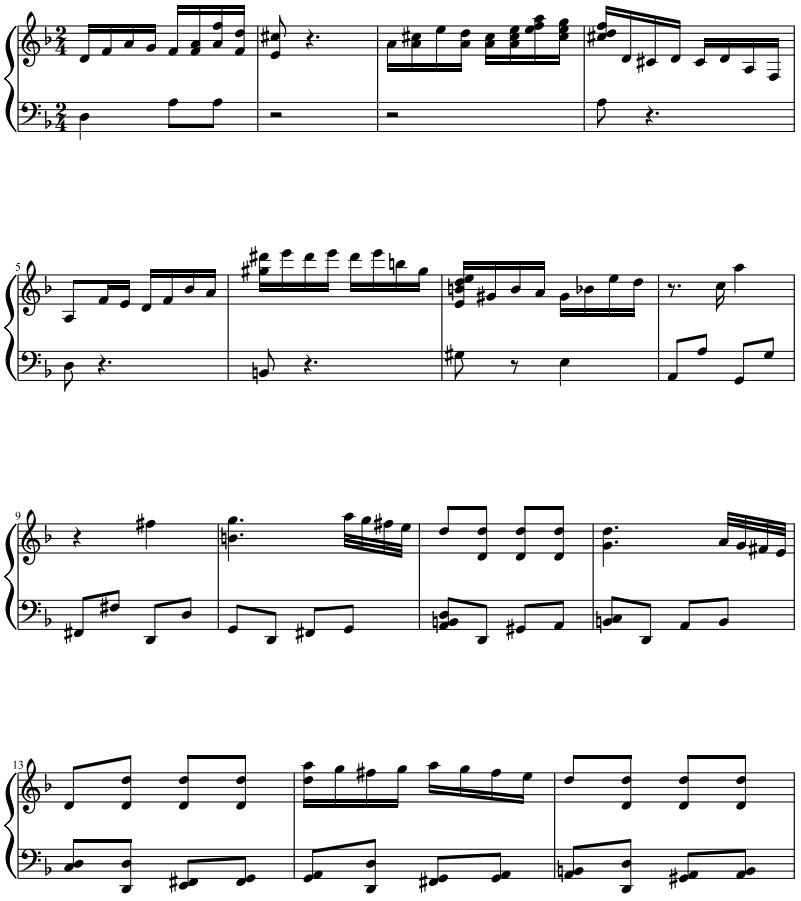
- Training with 10 iterations



- Training with 50 iterations

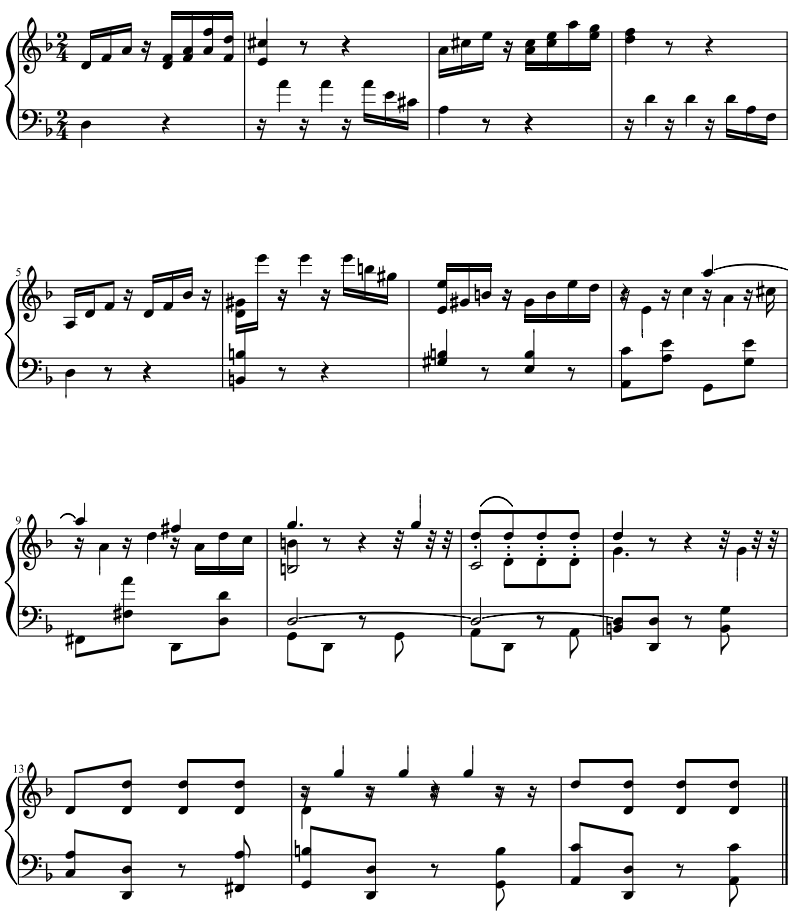


- Training with 300 iterations

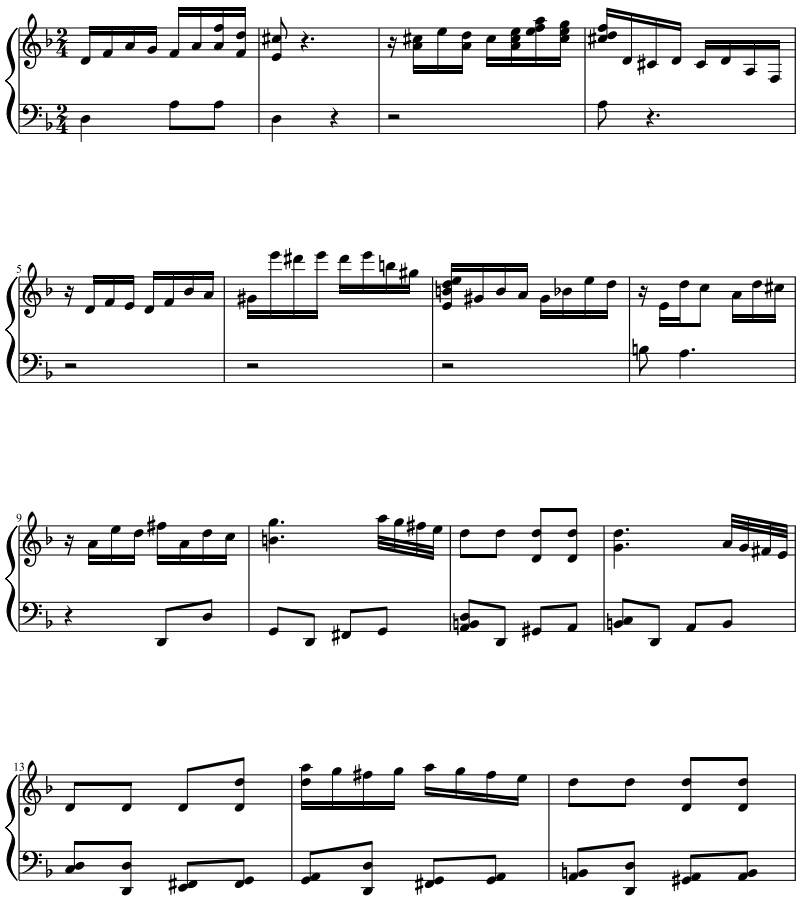


B.2.2 Reduction by Removing Dissonances

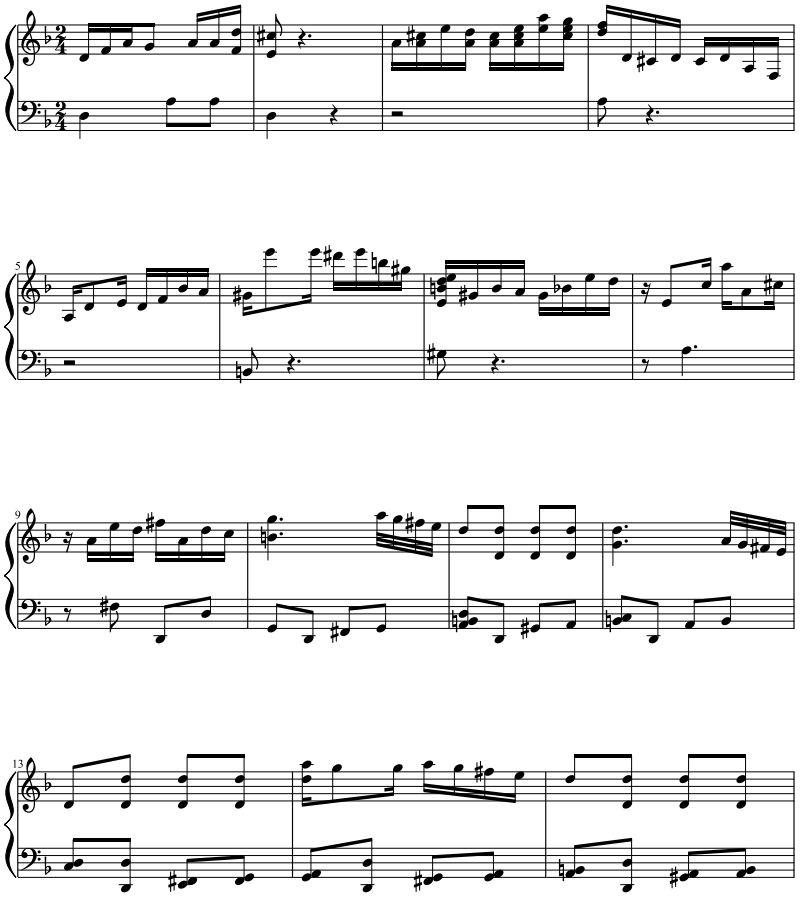
- Expected Result



- Training with 10 iterations



- Training with 50 iterations



- Training with 300 iterations