УЧЕНЫЕ ЗАПИСКИ КАЗАНСКОГО УНИВЕРСИТЕТА
Том 155, кн. 4 Физико-математические науки *2013*

UDK 004.93

MID-LEVEL FEATURES FOR AUDIO CHORD RECOGNITION USING A DEEP NEURAL NETWORK

N. Glazyrin

Abstract

Deep neural networks composed of several pre-trained layers have been successfully applied to various tasks related to audio processing. Some configurations of deep neural networks (including deep recurrent networks) which can be pretrained with the help of stacked denoising autoencoders are proposed and examined in this paper in application to feature extraction for audio chord recognition task. The features obtained from an audio spectrogram using such network can be used instead of conventional chroma features to recognize the actual chords in the audio recording. Chord recognition quality that was achieved using the proposed features is compared to the one that was achieved using conventional chroma features which do not rely on any machine learning technique.

Keywords: audio chord recognition, autoencoder, recurrent network, deep learning.

Introduction

One of the important areas of music information retrieval is the estimation of elements related to musical concepts. The concepts such as note, chord, melody, key can be determined from musical score, but their extraction from an audio signal is a very difficult task even for the human.

The ultimate goal of estimating automatically all musical concepts of a given musical audio, or obtaining its transcription, can hardly be solved. Instead, many researchers work on different aspects of this task, such as audio melody extraction, beat detection or audio chord estimation. The information about chords contained in a recording can be useful for other tasks related to music processing, for example structural segmentation. This information is valuable per se, as it can be used to index musical recordings by their content, to help musicologists to analyse harmony of a song or to let amateur musicians play their favourite songs when chords are not available from other sources.

More formally, the task of audio chord recognition consists in following. Given a digital musical audio recording one needs to determine the sequence of chords that were played in this recording together with their boundaries: for each chord the name and the times of its beginning and end must be specified. Chord names are usually chosen from a fixed set of 12 major and 12 minor chords plus a N symbol for a missing chord (e.g. when no pitched musical instrument is playing or when there is silence).

During chord recognition various properties of music are exploited, such as the presence of rhythm, the presence of harmonic frequencies of pitched instruments, repeated sections, polyphony. But other factors, such as noises, unpitched musical instruments, a voice singing out-of-tune, can bring negative influence and need to be dealt with.

A common approach to the task of chord recognition can be divided into 3 steps. At first, audio recording is pre-processed (beat detection is often performed here) and transformed to the time-frequency domain (e.g. using fast Fourier transform or constant-Q

transform [1], resulting in a so-called spectrogram. Each spectrogram column represents the spectrum of a short fragment of the original recording.

Then a series of transforms is applied to the spectrogram to consider the music properties and factors mentioned above and transform it to a sequence of feature vectors. Some widely used feature types are pitch class profile vectors [2] and chroma DCT-reduced log-pitch [3]. They are generally called *chroma* vectors, because they have 12 components, one component per pitch class (a pitch class unites all frequencies that correspond to equally named notes that belong to different octaves).

Finally, the sequence of feature vectors is converted to the sequence of chord symbols. Chord boundaries appear naturally, because of initial division of the recording into a set of fragments with fixed boundaries. Probabilistic models such as hidden Markov models (e.g. [4]) and dynamic Bayesian networks [5] are often used here because of their ability to model series of sequential events and include various factors (bass note, musical key). The target sequence of chord labels can be obtained from the sequence of observed feature vectors by application of the Viterbi algorithm to the model.

Simpler algorithms that only compare the feature vectors with predefined template vectors for chords [6] perform a little worse, but do not require training and therefore cannot be overfitted to concrete chord sequences or music style. We believe that a good chord recognition quality can be achieved without probabilistic models, using better features with simpler classifier instead.

Promising results in this direction were obtained in [7] with the help of deep convolutional neural network. In this paper we investigate another type of deep neural networks in application to chord recognition – multi-layer perceptrons pre-trained as stacked denoising autoencoders. Unlike convolutional neural networks they process spectrogram as a sequence of columns, not as a composition of rectangles. Networks of this type have been successfully applied to automatic speech recognition [8]. With the introduction of recurrent connections it becomes possible to model the dependency on the chord playing during previous fragments of a recording.

This paper continues the work started in [9] and gives better understanding of some important facets of the network construction and the system at large. It is also shown how the configuration of the network impacts its training and test times.

The remainder of the paper is organized as follows. Section 1. introduces autoencoders and their modifications. In section 2. the experimental setup and evaluation methods are described. In section 3. the results are presented and discussed. Section 4. provides the directions for future work.

1. Autoencoders and recurrent networks

1.1. Theoretical background. The definitions here are given in concordance with [10].

An *autoencoder* (also often called autoassociator) can be represented as a pair of transforms

$$y = f_\theta(x) = s(Wx + b), \tag{1}$$

$$z = g_{\theta'}(y) = s(W'y + b'). \tag{2}$$

Here x is an input vector, z is the reconstructed output vector, y is the internal representation of x, $\theta = \{W, b\}$ and $\theta' = \{W', b'\}$ are parameters (with a typical restriction $W' = W^T$), s is a non-linear activation function, natural choice for it is the sigmoid function or the hyperbolic tangent function. In (2) a linear function s is sometimes used. A convenient representation for an autoencoder is a neural network with one hidden layer.

During autoencoder training the cost function $L(X, Z(X))$ is minimized, where X is a set of possible inputs. A natural choice in case when X is a set of spectrogram columns is the squared error objective function $L(x, z) = \|x - z\|^2$.

To avoid learning the identity mapping, the internal representation often has less dimensions than the input vector. Alternative way is to make the internal representation having more dimensions than the input vector, and introduce the sparsity restriction on the hidden layer, so that most of its activations are close to zero. In this case the internal representation becomes a sparse representation of the input. If we denote as $f_\theta^j(x)$ the activation of hidden unit j for an input x, then we can define the average activation of this hidden unit over the whole training set of size m:

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} f_\theta^j \left(x^{(i)} \right). \tag{3}$$

We would like to enforce $\hat{\rho}_j = \rho$, where ρ is a sparsity parameter, typically close to zero, we choose $\rho = 0.05$. To achieve this, an extra penalty term L_ρ is added to the cost function L. Many choices for this term are possible, we apply the one proposed in [11]:

$$L_\rho = \beta \left[\sum_{j=1}^{h} \left(\rho \log \frac{\rho}{\widehat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \widehat{\rho}_j} \right) \right], \tag{4}$$

where h is the number of units in the hidden layer.

A *denoising autoencoder* is trained to reconstruct the input vector x from its corrupted version $\widetilde{x}$. It was shown (see [12]) that internal representations obtained from a denoising autoencoder are more stable and robust and capture better the internal structure of the input distribution. Two often used noise types are additive isotropic Gaussian noise $\widetilde{x}|x \sim \mathcal{N}(x, \sigma^2 I)$ and masking noise, that forces a fraction ν of elements chosen at random to be 0.

Denoising autoencoders can be *stacked* by connecting the hidden layer outputs of one autoencoder to the inputs of another autoencoder, thus forming more complex data representations in the deeper layers. The layers are then pre-trained one by one in an unsupervised manner.

Recurrent denoising autoencoder can be constructed from the conventional denoising autoencoder by adding a recurrent connection from its hidden layer to itself (which results essentially in the Elman network [13]). Then the output of this layer is computed as

$$y(x_t) = s\left(W x_t + b + U y(x_{t-1})\right). \tag{5}$$

Output of a recurrent layer depends both on current input vector and on the hidden representation of previous input vectors. Any layer (and even several layers) can be turned into a recurrent one.

1.2. Adaptation to audio chord recognition. A serious shortcoming of neural networks is the lack of an evident interpretation of values in hidden layers. So an additional logistic regression layer with 12 outputs can be added to make network output interpretable. This additional layer is connected to the hidden layer of the innermost autoencoder, and its output will be treated as chroma vector. Reconstruction layers of autoencoders are then discarded, and the whole network is fine-tuned in a supervised manner. In fact, autoencoders are used to pre-train the network layers and put their parameters in the area where their best values probably reside. Fine tuning is needed to adjust parameters of the last logistic regression layer.

The training data is obtained from known chord labels. Each of them can be represented as an ideal 12-dimensional vector which has 1 on the positions that correspond to the notes of a given chord, and 0 on the other positions. These vectors are often called binary chord templates. No chord (N) corresponds to null vector, and therefore can be detected when the element of the resulting vector with maximum absolute value is no greater than Δ – the parameter, which is adjusted empirically.

To compensate for uneven distribution of chords in training data, the following trick can be applied. Note that binary templates for all chords of a given type (e.g. major chords) can be obtained from the C major chord template by cyclic permutations of its components. So for each spectrogram column it is possible to generate 12 input vectors (and 12 corresponding output vectors from the corresponding chord label) by applying cyclic permutation in one direction 12 times. This way, equal counts of all chords of one type in the training data can be achieved. But cyclic permutation of spectrum components is unnatural, because highest spectral components are moved before lowest ones. So it is better to replace it with a sliding window and let the initial spectrum span 1 octave more, thus providing 12 additional values.

Cyclic permutation of inputs or chroma vector components is a common trick when learning model parameters in chord recognition algorithms. We propose here to use it both when training a stacked denoising autoencoder and when testing it on unknown inputs. Chroma vectors for a new spectrogram can be calculated 12 times with 12 different roots, then rotated to the same root and averaged. This can potentially result in better recognition quality.

To compensate for the difference in total numbers of major and minor chords within the training set, we restricted this difference to be no more than 100 (which turns to 1200 due to cyclic permutation) in the generated set of examples for neural network training.

2. Experimental setup

2.1. Pre-processing and spectrogram. A thorough description of the spectrogram calculation process can be found in [14]. We will further assume that the initial spectrogram spans 6 octaves: from C2 (65.41 Hz) to B7 (3951 Hz) and has 1 row per note. Therefore each spectrogram column consists of 72 values that represent the intensity of the corresonding note on a given sound fragment. When less components are required for calculations, the lowest ones are always used. A logarithmic transformation is applied to each value to mimic the human perception of sound intensity: each value v is replaced with $\log_{10}(1000v + 1)$, as proposed in [3]. Each spectrogram column was normalized to fit its values into $[0, 1]$ before passing it to the neural network.

2.2. Feature vectors and neural networks. Twelve-dimensional chroma feature vectors are obtained from the neural network trained as described above. We experimented with various network layouts. The network in all experiments has 12 outputs, but the number of inputs, the number of hidden layers and their size varied across experiments. The following options were considered:

- the described network with 48 or 60 inputs, 1, 2 or 3 hidden layers and no recurrent connections (SDA);
- the described network with 48 or 60 inputs, 1, 2 or 3 hidden layers and recurrent connection from the innermost hidden layer to itself (RSDA).

2.3. Post-processing. At first, each feature vector is replaced with a linear combination of 10% of most similar vectors, where each vector's weight is its similarity to

the source feature vector. Similarity is calculated as Euclidean distance between 2 vectors. This makes it possible to take into account the repetition of music phrases. Similar technique was employed in [5], but they only preserved the diagonals in self-similarity matrix which are parallel to the main diagonal. We do not put such restriction on this matrix.

Additionally, two heuristics were implemented to correct chord detection errors of certain types. First heuristic consists in searching for all intervals where multiple chords of the same root but different type are arranged next to each other, and then replacing each interval with a single chord label. This chord is chosen as the nearest one to the sum of all feature vectors within the interval. Another heuristic was introduced to fix chord sequences like (A, B, C), where 3 successive beats are marked with 3 different chords. Each similar sequence is replaced with the one of (A, A, C), (A, C, C), (B, B, C), (A, B, B) for which the sum the of distances from successive feature vectors to corresponding chord labels is minimal.

2.4. Evaluation. The experiments were conducted on the combination of *Isophonics* [15] and *RWC* [16] datasets. The former consists of 218 songs by *The Beatles*, *Queen* and *Zweieck*. The latter consists of 100 Japanese pop songs. This dataset of 318 tracks was randomly divided into 2 groups of same size, which were treated by turns as train and test sets.

Given the reference sequence of chord labels and the estimated sequence of chord labels for a recording the evaluation was performed as follows. At first, the recording is separated into segments using all chord boundaries from both reference and estimated sequences. Then on each segment a "Triads" score was calculated as described in [17]. Its value s equals to 1 on a segment when estimated and reference chord on this segment reduced to their corresponding triads (first 3 notes) are equal or are both no chord symbols. Otherwise $s = 0$. Then the overlap ratio is calculated for this recording as:

$$OR = \sum_{i=1}^{N_{\text{segm}}} s_i \ell_i \Big/ \sum_{i=1}^{N_{\text{segm}}} \ell_i, \tag{6}$$

where s_i is the score on ith segment, ℓ_i is the length of ith segment, N_{segm} is the total number of segments in the recording. *Weighted average overlap ratio* is defined for the whole dataset as:

$$WAOR = \sum_{i=1}^{N_{\text{tracks}}} OR_i \cdot L_i \Big/ \sum_{i=1}^{N_{\text{tracks}}} L_i, \tag{7}$$

where OR_i is the overlap ratio for ith recording, L_i is the total length of ith recording and N_{tracks} is the size of the dataset (318 recordings).

The values of weighted average overlap ratio calculated during the MIREX Audio Chord Estimation 2013 contest on the Isophonics collection [1] are between 0.71 and 0.82. Average segmentation defined in [5] is not commonly used, but we also provide its values here. It characterizes the chord boundaries estimation quality.

We also performed statistical significance tests to determine if there is a significant difference between algorithm variations. Friedman's ANOVA with post-hoc Tukey HSD is used in MIREX Audio Chord Estimation and other MIREX contests [18]. In this work the R [19] implementation [20] was employed using the code by Tal Galili[2].

[1] http://www.music-ir.org/mirex/wiki/2013:Audio_Chord_Estimation_Results_MIREX_2009

[2] http://www.r-statistics.com/2010/02/post-hoc-analysis-for-friedmans-test-r-code)

Table 1. Chord recognition quality achieved with various neural network configurations

Configuration	Overlap	Segmentation
SDA (48, 200)	0.7639	0.8035
SDA (48, 200, 200)	0.7616	0.8009
SDA (60, 100)	0.7616	0.7991
SDA (60, 200)	0.7637	0.8010
SDA (60, 300)	0.7649	0.8011
SDA (60, 200, 100)	0.7664	0.8011
SDA (60, 300, 300)	0.7650	0.8012
SDA (60, 100, 100, 100)	0.7671	0.7996
SDA (60, 300, 300, 300)	0.7674	0.8011
RSDA (48, 200)	0.7616	0.7963
RSDA (48, 200, 200)	0.7660	0.7982
RSDA (60, 300)	0.7613	0.7955
RSDA (60, 300, 300)	0.7672	0.7982
RSDA (60, 300, 300, 300)	0.7686	0.7986

3. Results

The number of inputs in the network is relatively small. Therefore we only experimented with 1, 2 and 3 hidden layers. Learning rate for autoencoder pre-training and network learning were set to 0.03 and 0.01 respectively; both pre-training and network learning were run for 15 epochs. Batch size for batch gradient descent was set to 5. Standard deviation for isotropic Gaussian noise was set to 0.2, and the noise was applied to each spectral component with probability $p = 0.7$. Probability of setting vector component to 0 was set to 0.2 for masking noise.

Below in parentheses are written the numbers of units in input layer and hidden layers of neural networks (SDA or RSDA) or the number of spectral components used to calculate PCP, CLP or CRP features. In the experiments only the last pre-trained layer was made recurrent.

The first experiment is targeted at discovering the best performing network layout. Same layouts were used with and without recurrent connections. The spectrum was calculated from 6 or 5 octaves using sliding 5 or 4 octaves wide window (60 and 48 inputs respectively). Both logarithmic transformation and post-processing were applied. The results are summarized in Table 1.

In spite of the fact that the configurations with more units achieved a little better result, there was no significant differences between the results. Neither the presence of recurrent connections, nor the increase of input vector size improve chord recognition much. Only for some pairs the difference was statistically significant. But it can be seen from Table 2 that the total time of training and testing grows strongly with addition of more layers and recurrent connections. Because of that only the SDA (60, 300, 300) was used in the following experiments.

It was mentioned above that a logarithmic transformation is applied to the spectrogram before passing its values to a neural network. To check whether a network can perform at the same level without this transform, the SDA (60, 300, 300) network was trained using spectral data without logarithmic transformation. The results are shown in Table 3. It can be concluded that this transformation is an important part of data preparation process, as its absence drops the result significantly.

In paragraph 1.1. various noise types have been mentioned. Noise is added to input vectors to let autoencoder learn better representation of input data in the hidden leayer. Table 4 confirms this assumption. The results obtained with 2 different types of noise are

Table 2. Total training and test times for different network configurations

Configuration	Training time	Test time
SDA (60, 100)	43 min 49 s	4 min 38 s
SDA (60, 200)	1 h 2 min 19 s	4 min 56 s
SDA (60, 300)	1 h 20 min 48 s	5 min 17 s
SDA (60, 200, 100)	2 h 19 min 52 s	7 min 6 s
SDA (60, 300, 300)	6 h 21 min 28 s	8 min 18 s
SDA (60, 300, 300, 300)	15 h 21 min 19 s	13 min 21 s
RSDA (60, 300)	6 h 55 min 43 s	1 h 52 min 01 s

Table 3. The effect of non-linear transformations

Algorithm	Overlap	Segmentation
SDA (60, 300, 300) (no log)	0.7350	0.7890
SDA (60, 300, 300)	0.7672	0.7982

Table 4. The effect of various noise types during autoencoder pre-training

Noise type	Triads	Segmentation
additive	0.7672	0.7982
masking	0.7677	0.8018
no noise	0.7620	0.7989

Table 5. The effect of sliding window application during testing

Configuration	Triads	Segmentation
SDA (60, 300, 300) (no sliding window)	0.7703	0.8020
SDA (60, 300, 300)	0.7672	0.7982

better than the one obtained without noise addition, and this difference is statistically significant. But there is no significant difference between additive noise and masking noise.

In paragraph 1.2. we proposed to use sliding window to imitate cyclic permutations both during training and testing phases. It is definitely important during training to equalize the numbers of training examples per chord. But it needs to be checked whether this procedure is helpful on testing phase. As can be seen from Table 5, its effect is even negative.

Finally, these numbers can be compared to the result obtained under the same configuration, but using one of the best conventional chroma features – DCT-reduced log pitch (CRP) [3] – instead: Triads – 0.7720, Segmentation – 0.8141. The difference between this result and the one obtained using the SDA (60, 300, 300) network is not so big, but statistically significant.

4. Conclusions and future work

It can be concluded, that the features calculated with the proposed deep network make it possible to achieve practically the same chord recognition quality as the best conventional features. But the configuration of the network and the type of noise added during layerwise pre-training are of less importance than the method of data preparation. Better spectrum data can improve both conventional features and neural network based ones.

Another important conclusion is that the addition of layers and units improves the result slightly at the cost of multiplied training time. The same is true for recurrent connections. In addition they make testing process a lot slower, because the output vector at every step depends on all previous steps, and therefore separate input vectors cannot be processed in parallel.

In this paper we have investigated how a deep neural network that processes a spectrogram column by column (as opposed to convolutional neural networks) can produce effective chroma features that can be used for audio chord recognition. In the future we plan to investigate whether aggregating these features with conventional ones can lead to better chord recognition quality. A separate problem is to find another way of inclusion of time information into the model, because recurrent connections do not bring much improvement within the current network configuration.

Резюме

Н.Ю. Глазырин. Промежуточное представление звуковых данных с использованием многослойной нейронной сети для задачи распознавания аккордов.

Глубокие нейронные сети, состоящие из нескольких предварительно обученных слоёв, успешно применяются в задачах, связанных с обработкой звука. В данной работе предложены и рассмотрены применительно к задаче распознавания аккордов некоторые конфигурации глубоких нейронных сетей (в том числе рекуррентных), допускающие предварительное послойное обучение при помощи многослойных очищающих автоассоциаторов. Рассмотренные нейронные сети позволяют преобразовывать спектрограмму звукозаписи в последовательность векторов признаков, по которой затем определяются звучащие аккорды. Качество распознавания аккордов, достигнутое с использованием описанных признаков, сравнивается с качеством распознавания аккордов, достигнутым при помощи часто используемых хроматических признаков, при вычислении которых не используются методы машинного обучения.

Ключевые слова: распознавание аккордов, автоассоциатор, рекуррентная нейронная сеть, глубокое обучение.

References

1. *Brown J.C.* Calculation of a constant Q spectral transform // J. Acoust. Soc. Am. – 1991. – V. 89, No 1. – P. 425–434.
2. *Fujishima T.* Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music // Proc. Int. Computer Music Conf. (ICMC). – 1999. – P. 464–467.
3. *Müller M., Ewert S., Kreuzer S.* Making chroma features more robust to timbre changes ICASSP'09 Proc. 2009 IEEE Int. Conf. on Acoustics, Speech and Signal Processing. – Washington, DC, USA: IEEE Comput. Soc., 2009. – P. 1877–1880.
4. *Khadkevich M., Omologo M.* Use of Hidden Markov Models and Factored Language Models for Automatic Chord Recognition // Proc. 10th Int. Soc. for Music Information Retrieval Conference (Kobe, Japan, October 26–30, 2009) – 2009. – P. 561–566.
5. *Mauch M.* Automatic Chord Transcription from Audio Using Computational Models of Musical Context: PhD Thesis. – London: University of London, 2010. – 168 p.
6. *Oudre L., Grenier Y., Févotte C.* Template-Based Chord Recognition: Influence of the Chord Types // Proc. 10th Int. Soc. for Music Information Retrieval Conference (Kobe, Japan, October 26–30, 2009). – 2009 – P. 153–158.
7. *Humphrey E.J., Cho T., Bello J.P.* Learning a robust Tonnetz-space transform for automatic chord recognition // Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP-12). – 2012. – P. 453–456.

8. *Maas A.L., Le Q.L., O'Neil T.M., Vinyals O., Nguyen P., Ng A.Y.* Recurrent Neural Networks for Noise Reduction in Robust ASR // INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association (Portland, Oregon, USA, September 9–13, 2012). – 2012. – URL: http://www1.icsi.berkeley.edu/~vinyals/Files/rnn_denoise_2012.pdf/.

9. *Glazyrin N.Yu.* Use of autoencoders for recognition of chord sequences in digital sound recordings // Proc. All-Russ. Sci. Conf. "Analysis of Images, Social Networks, and Texts" (AIST 2013), Ekaterinburg, April 4–6, 2013. – Moscow: INTUIT National Open University, 2013. – P. 211–215. (In Russian)

10. *Vincent P., Larochelle H., Lajoie I., Bengio Y., Manzagol P.-A.* Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion // J. Mach. Learn. Res. – 2010. – V. 11. – P. 3371–3408.

11. *Ng A.* CS294A Lecture Notes. Sparse autoencoder. – URL: http://www.stanford.edu/class/cs294a/sparseAutoencoder.pdf.

12. *Bengio Y., Courville A.C., Vincent P.* Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives. – 2012. – URL: http://arxiv.org/abs/1206.5538.

13. *Elman J.L.* Finding structure in time // Cognitive Sci. – 1990. – V. 36, No 2. – P. 179–211.

14. *Glazyrin N.Yu,* Towards the task of audio chord esitmation // Izv. Irkutsk. Gos. Univ. Ser. Matem. – 2013. – V. 6, No 2. – P. 2–17. (In Russian)

15. *Mauch M., Cannam C., Davies M., Dixon S., Harte C., Kolozali S., Tidhar D., Sandler M.* OMRAS2 Metadata Project 2009 // Late-breaking session at the 10th Int. Conf. on Music Information Retrieval, Kobe, Japan. – 2009. – URL: http://matthiasmauch.de/_pdf/mauch_omp_2009.pdf.

16. *Goto M., Hashiguchi H., Nishimura T., Oka R.* RWC Music Database: Popular, Classical, and Jazz Music Databases // Proc. 3rd Int. Conf. on Music Information Retrieval (ISMIR 2002). – 2002 – P. 287–288.

17. *Pauwels J., Peeters G.* Evaluating automatically estimated chord sequences // Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP-13). – 2013. – P. 749–753.

18. *Downie S.J.* The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research // Acoust. Sci. & Tech. – 2008. – V. 29, No 4. – P. 247–255.

19. *R Core Team* R: A language and environment for statistical computing. – Vienna, Austria: R Foundation for Statistical Computing, 2012.

20. *Hothorn T., Hornik K., Wiel M.A. van de, Zeileis A.* Implementing a Class of Permutation Tests: The Coin Package // J. Stat. Software. – 2008. – V. 28, No 8. – P. 1–23.

Поступила в редакцию
02.08.13

Glazyrin, Nikolai – PhD Student, Department of Algebra and Discrete Mathematics, Ural Federal University named after B.N. Yeltsin, Ekaterinburg, Russia.

Глазырин Николай Юрьевич – аспирант кафедры алгебры и дискретной математики, Уральский федеральный университет им. Б.Н. Ельцина, г. Екатеринбург, Россия.

E-mail: *nglazyrin@gmail.com*