14

PCA and *K*-Means Decipher Genome

Alexander N. Gorban[1,3] and Andrei Y. Zinovyev[2,3]

[1] University of Leicester, University Road, Leicester, LE1 7RH, UK,
ag153@le.ac.uk
[2] Institut Curie, 26, rue d'Ulm, Paris, 75248, France,
andrei.zinovyev@curie.fr
[3] Institute Of Computational Modeling of Siberian Branch of Russian Academy of Science, Krasnoyarsk, Russia

Summary. In this paper, we aim to give a tutorial for undergraduate students studying statistical methods and/or bioinformatics. The students will learn how data visualization can help in genomic sequence analysis. Students start with a fragment of genetic text of a bacterial genome and analyze its structure. By means of principal component analysis they "discover" that the information in the genome is encoded by non-overlapping triplets. Next, they learn how to find gene positions. This exercise on PCA and K-Means clustering enables active study of the basic bioinformatics notions. The Appendix contains program listings that go along with this exersice.

Key words: Bioinfomatics; Data visualization; Cryptography; Clustering; Principal component analysis

14.1 Introduction

When it is claimed in newspapers that a new genome is deciphered, it usually means that the sequence of the genome has been read only, so that a long sequence using four genetic letters: A, C, G and T is known. The first step in complete deciphering of the genome is identification of the positions of elementary messages in this text or *detecting genes* in the biological language. This is imperative before trying to understand what these messages mean for a living cell.

Bioinformatics – and genomic sequence analysis, in particular – is one of the hottest topics in modern science. The usefulness of statistical techniques in this field cannot be underestimated. In particular, a successful approach to the identification of gene positions is achieved by statistical analysis of genetic text composition.

In this exercise, we use Matlab to convert genetic text into a table of short word frequencies and to visualize this table using principal component analy-

sis (PCA). Students can find, by themselves, that the sequence of letters is not random and that the information in the text is encoded by non-overlapping triplets. Using the simplest K-Means clustering method from the Matlab Statistical toolbox, it is possible to detect positions of genes in the genome and even to predict their direction.

14.2 Required Materials

To follow this exercise, it is necessary to prepare a genomic sequence. We provide a fragment of the genomic sequence of *Caulobacter Crescentus*. Other sequences can be downloaded from the Genbank FTP-site [1]. Our procedures work with the files in the *Fasta* format (the corresponding files have a *.fa* extension) and are limited to analyzing fragments of 400–500 kb in length.

Five simple functions are used:

LoadFreq.m	loads Fasta-file into a Matlab string
CalcFreq.m	converts a text into a numerical table of short word frequencies
PCAFreq.m	visualizes a numerical table using principal component analysis
ClustFreq.m	is used to perform clustering with the K-Means algorithm
GenBrowser.m	is used to visualize the results of clustering on the text

All sequence files and the m-files should be placed into the current Matlab working folder. The sequence of Matlab commands for this exercise is the following:

```
str = LoadSeq('ccrescentus.fa');
xx1 = CalcFreq(str,1,300);
xx2 = CalcFreq(str,2,300);
xx3 = CalcFreq(str,3,300);
xx4 = CalcFreq(str,4,300);
PCAFreq(xx1);
PCAFreq(xx2);
PCAFreq(xx3);
PCAFreq(xx4);
fragn = ClustFreq(xx3,7);
GenBrowser(str,300,fragn,13000);
```

All the required materials can be downloaded from [2].

14.3 Genomic Sequence

14.3.1 Background

The information that is needed for a living cell functioning is encoded in a long molecule of DNA. It can be presented as a text with an alphabet that has only four letters A, C, G and T. The diversity of living organisms and their complex properties is hidden in their genomic sequences. One of the most exciting problems in modern science is to understand the organization of living matter by reading genomic sequences.

One distinctive message in a genomic sequence is a piece of text, called *a gene*. Genes can be oriented in the sequence in the forward and backward directions (see Fig. 14.1). This simplified picture with unbroken genes is close to reality for bacteria. In the highest organisms (humans, for example), the notion of a gene is more complex.

It was one of many great discoveries of the twentieth century that biological information is encoded in genes by means of triplets of letters, called *codons* in the biological literature. In the famous paper by Crick *et al.* [3], this fact was proven by genetic experiments carried out on bacteria mutants. In this exercise, we will analyze this by using the genetic sequence only.

In nature, there is a special mechanism that is designed to read genes. It is evident that as the information is encoded by non-overlapping triplets, it is important for this mechanism to start reading a gene without a shift, from the first letter of the first codon to the last one; otherwise, the information decoded will be completely corrupted.

An easy introduction to modern molecular biology can be found in [4].

14.3.2 Sequences for the Analysis

The work starts with a fragment of genomic sequence of the *Caulobacter Crescentus* bacterium. A short biological description of this bacterium can be found in [5]. The sequence is given as a long text file (300 kb), and the students are asked to look at the file and ensure that the text uses the alphabet of four letters (a, c, g and t) and that these letters are used without spaces. It is noticeable that, although the text seems to be random, it is well organized, but we cannot understand it without special tools. Statistical methods can help us understand the text organization.

The sequence can be loaded in the Matlab environment by *LoadSeq* function:

```
str = LoadSeq('ccrescentus.fa');
```

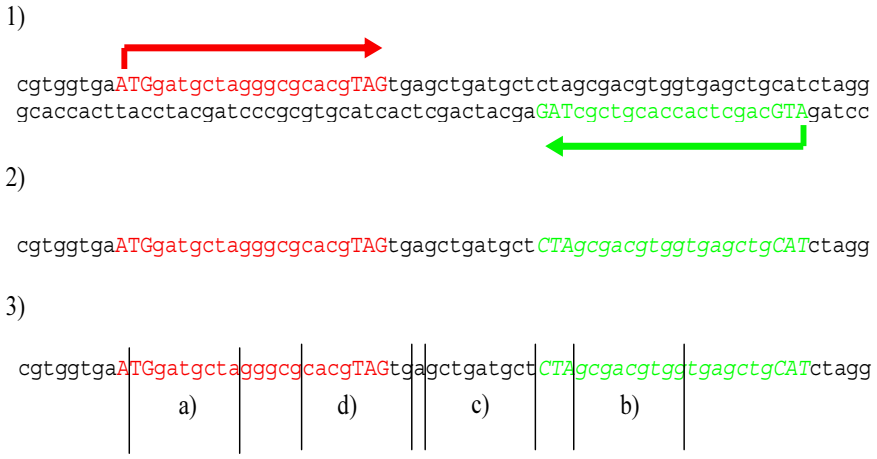

Fig. 14.1. 1) DNA can be represented as two complementary text strings. "Complementary" here means that instead of any letter "a","t","g" and "c" in one string there stands "t","a","c" and "g", respectively, in the other string. Elementary messages or genes can be located in both strings, but in the lower one they are read from right to the left. Genes usually start with "atg" and end with "tag" or "taa" or "tga" words. **2**) In databases one obtains only the upper "forward" string. Genes from the lower "backward" string can be reflected on it, but should be read in the opposite direction and changing the letters, accordingly to the complementary rules. **3**) If we take a randomly chosen fragment, it can be of one of three types: **a**) entirely in a gene from the forward string; **b**) entirely in a gene from the backward string; **c**) entirely outside genes; **d**) partially in genes and partially outside genes

14.4 Converting Text to a Numerical Table

A *word* is any continuous piece of text that contains several subsequent letters. As there are no spaces in the text, separation into words is not unique.

The method we use is as follows. We clip the whole text into fragments of 300 letters[4] in length and calculate the frequencies of short words (of length 1–4) inside every fragment. This will give us a description of the text in the form of a numerical table. There will be four such tables for every short word length from 1 to 4.

As there are only four letters, there are four possible words of length 1 (singlets), $16 = 4^2$ possible words of length 2 (duplets), $64 = 4^3$ possible words of length 3 (triplets) and $256 = 4^4$ possible words of length 4 (quadruplets). The first table contains four columns (frequency of every singlet) and the number of rows equals the number of fragments. The second table has 16 columns and the same number of rows, and so on.

[4] Mean gene size in bacteria is about 1000 genetic letters, the fragment length in 300 letters corresponds well to detect genes on this scale, with some resolution

To calculate the tables, students use the *CalcFreq.m* function. The first input argument for the function *CalcFreq* is the string containing the text, the second input argument is the length of the words to be counted, and the third argument is the fragment length. The output argument is the resulting table of frequencies. Students use the following set of commands to generate tables corresponding to four different word lengths:

```
xx1 = CalcFreq(str,1,300);
xx2 = CalcFreq(str,2,300);
xx3 = CalcFreq(str,3,300);
xx4 = CalcFreq(str,4,300);
```

14.5 Data Visualization

14.5.1 Visualization

PCAFreq.m function has only one input argument, the table obtained from the previous step. It produces a PCA plot for this table (PCA plot shows distribution of points on a principal plane, with the x axis corresponding to the projection of the point on the first principal component and the y axis corresponding to the projection on the second one). For an introduction to PCA, see [6].

By typing the commands below, students produce four plots for the word lengths from 1 to 4 (see Fig. 14.2).

```
PCAFreq(xx1);
PCAFreq(xx2);
PCAFreq(xx3);
PCAFreq(xx4);
```

14.5.2 Understanding Plots

The main message in these four pictures is that the genomic text contains information that is encoded by non-overlapping *triplets*, because the plot corresponding to the triplets is evidently highly structured as opposed to the pictures of singlets, duplets and quadruplets. The triplet picture evidently contains 7 clusters.

It is important to explain to students how 7-cluster structure in Fig. 14.1 occurs in nature.

Let us suppose that the text contains genes and that information is encoded by non-overlapping subsequent triplets (codons), but we do not know where the genes start and end or what the frequency distribution of codons is.

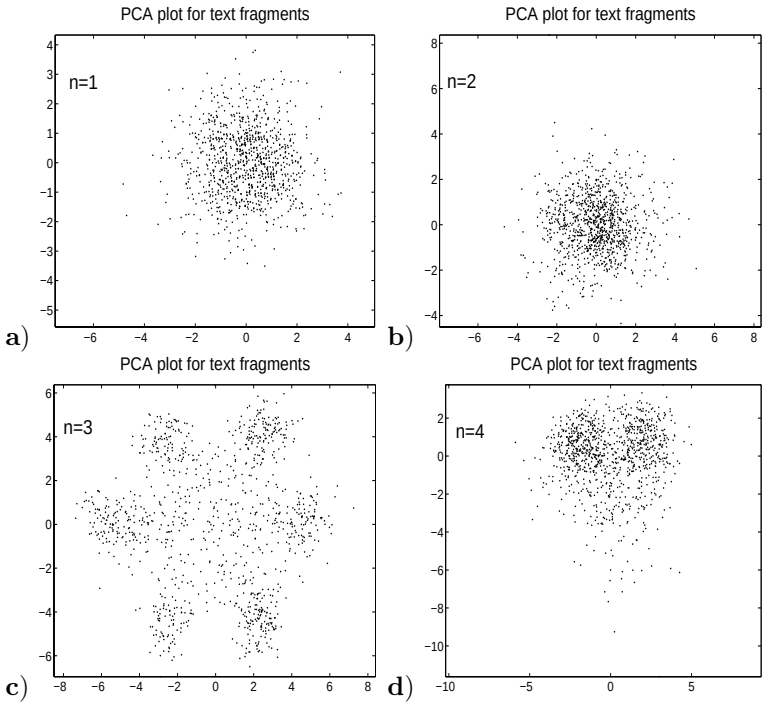

Fig. 14.2. PCA plots of word frequencies of different length. **c**) Shows the most structured distribution. The structure can be interpreted as the existence of a non-overlapping triplet code

Let us blindly cut the text into fragments. Any fragment can contain: a) a piece of a gene in the forward direction; b) a piece of a gene in the backward direction; c) no genes (non-coding part); d) or a mix of coding and non-coding parts.

Consider the first case (a). The fragment can overlap with a gene in three possible ways, with three possible shifts (mod3) of the first letter of the fragment with respect to the start of the gene. If we enumerate the letters in the gene form the first one, 1-2-3-4-..., then the first letter of the fragment can be in the sequence 1-4-7-10-... (=1(mod(3)) (a "correct" shift), the sequence 2-5-8-11-... (=2(mod(3)), or 3-6-9-12-... (=0(mod(3)). If we start to read the information triplet by triplet, starting from the first letter of the fragment, we can read the gene correctly only if the fragment overlaps with it with a correct shift (see Fig. 14.1). In general, if the start of the fragment is not chosen deliberately, then we can read the gene in three possible ways. Therefore, this first case (a) generates three possible frequency distributions, each one of which is "shifted" in relation to another.

The second case (b) is analogous and also gives three possible triplet distributions. They are not independent of the ones obtained in the first cases

for the following reason. The difference is the triplets are read "from the end to the beginning" which produces a kind of mirror reflection of the triplet distributions from the first case (a).

The third case (c) produces only one distribution, which is symmetrical with respect to the 'shifts' (or rotations) in the first two cases, and there is a hypothesis that this is a result of genomic sequence evolution. This can be explained as follows.

Vitality of a bacterium depends on the correct functioning of all biological mechanisms. These mechanisms are encoded in genes, and if something wrong happens with gene sequences (for example there is an error when DNA is duplicated), then the organism risks becoming non-vital. Nothing is perfect in our world and the errors happen all the time, including during DNA duplication. These errors are called *mutations*.

The most dangerous mutations are those that change the reading frame, i.e. letter deletions or insertions. If such a mutation happens inside a gene sequence, the rest of the gene becomes corrupted: the reading mechanism (which reads the triplets one by one and does not know about the mutation) will read it with a shift. Because of this the organisms with such mutations often die before producing offspring. On the other hand, if such a mutation happens in the non-coding part (where no genes are present) this does not lead to a serious problem, and the organism produces offspring. Therefore, such mutations are constantly accumulated in the non-coding part and three shifted triplet distributions are mixed into one. The fourth case (d) also produces a mix of triplet distributions.

As a result, we have three distributions for the first case (a), three for the second case (b) and one, symmetrical distribution for the 'non-coding' fragments (third case (c)). Because of natural statistical deviations and other reasons, we have 7 clusters of points in the multidimensional space of triplet frequencies.

For more illustrative material see [7, 8, 11, 15, 12].

14.6 Clustering and Visualizing Results

The next step is clustering the fragments into 7 clusters. This can be explained to the students that as a classification of fragments of text by similarity in their triplet distributions. This is an unsupervised classification (the cluster centers are not known in advance).

Clustering is performed by the K-Means algorithm using the Matlab Statistical toolbox. It is implemented in the *ClustFreq.m* file. The first argument is the frequency table name and the second argument is the number of clusters proposed. As we visually identified 7 clusters, in this activity we put 7 as the value of the second argument:

```
fragn = ClustFreq(xx3,7);
```

The function assigns different colors onto the cluster points. The cluster that is the closest to the center of the picture is automatically colored in black (see Fig. 14.3).

After clustering, every fragment of the text is assigned a cluster label (a color). The *GenBrowser* function puts this color back to the text and then visualizes it. The input arguments of this function are a string with the genetic text, the fragment size, a vector with cluster labels and a position in the genetic text to read from (we use a value 13000, which can be changed for any other position):

```
GenBrowser(str,300,fragn,13000);
```

This function implements a simple *genome browser* (a program for visualizing genome sequence together with other properties) with information about gene positions.

It is explained to the students that clustering of fragments corresponds to segmentation of the whole sequence into homogeneous parts. The homogeneity is understood as similarity of the short word frequency distributions for the fragments of the same cluster (class). For example, for the following text

aaaaaaaatataaaaaatttatttttttattttttgggggggggaagaggggcccccgcctccccccc

one can try to apply the frequency dictionary of length 1 for a fragment size around 5-10 to separate the text into four homogeneous parts.

14.7 Task List and Further Information

Interested students can continue this activity. They can modify the Matlab functions in order to solve the following problems (the first three of them are rather difficult and require programming):

Determine a cluster corresponding to the correct shift. As it was explained in the "Understanding plots" section, some fragments overlap with genes in such a way that the information can be read with the correct shift, whereas others contain the "shifted" information. The problem is to detect the cluster corresponding to the correct shift. To give a hint, we can say that the correct triplet distribution (probably) will contain the lowest frequency of the stop codons TAA, TAG and TGA (see [16, 17]). Stop codon can appear only once in a gene because it terminates its transcription.

Measure information content for every phase. The information of a triplet distribution with respect to a letter distribution is $I = \sum_{ijk} f_{ijk} \ln \frac{f_{ijk}}{p_i p_j p_k}$, where p_i is a frequency of letter i (a genetic text is characterized by four such frequencies), and f_{ijk} is the frequency of triplet ijk. Each cluster is a collection of fragments. Each fragment F can be divided on triplets starting

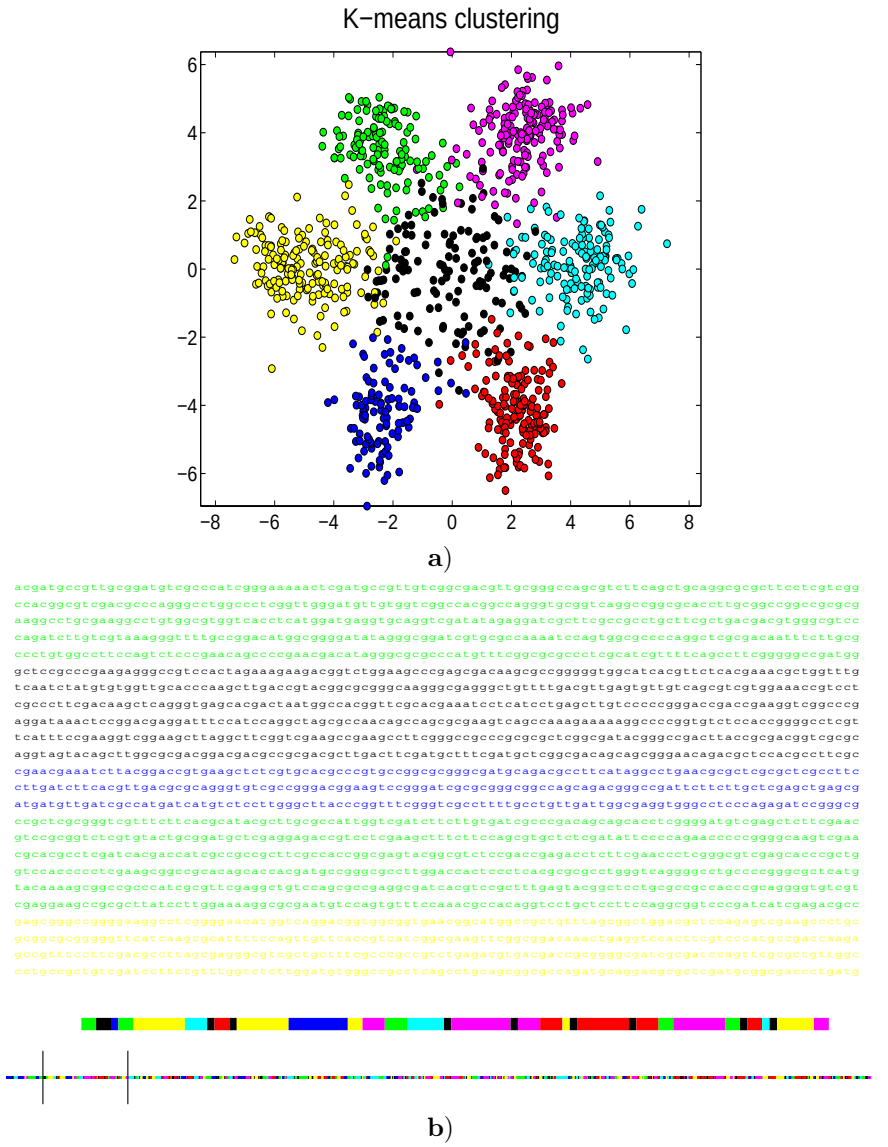

Fig. 14.3. K-Means clustering of triplet frequencies (**a**) and visualizing the clustering results onto the text (**b**). There are three scales in the browser. The bottom color code represents the genetic text as a whole. The second line shows colors of 100 fragments starting from a position in the text specified as an argument of the *GenBrowser.m* function. The letter color code shows 2400 symbols, starting from the same position. The black color corresponds to the fragments with non-coding information (the central cluster); other colors correspond to locations of coding information in different directions and with different shifts with respect to randomly chosen division of the text into fragments

from the first letter. We can calculate the information value of this triplet distribution $I(F)$ for each afragment F . Is the information of fragments in the cluster with a correct shift significantly different from the information of fragments in other clusters? Is the *mean* information value in the cluster with a correct shift significantly different from the *mean* information value of fragments in other clusters? Could you verify the hypothesis that in the cluster with a correct shift the mean information value is higher than in other clusters?

Increase resolution of determining gene positions. In this exercise, we use the same set of fragments to calculate the cluster centers and to annotate the genome. As a result the gene positions are determined with precision that is equal to the fragment size. In fact, cluster centers can be calculated with non-overlapping fragments and then the genome can be scanned again with a sliding window (this will produce a set of overlapping fragments). Following this, triplet frequencies for each window can be calculated and the corresponding cluster in the 64-dimensional space can be determined. The position in the text is assigned a color corresponding to the content of the fragment centered in this position. For further details of this process, see [9, 10, 11, 13].

Precise start and end positions of genes. The biological mechanism for reading genes (the polymerase molecule) identifies the beginning and the end of a gene using special signals, known as specialized codons. Therefore, almost all genes start with "ATG" start codon and end with "TAG", "TAA" or "TGA" stop codons. Try to use this information to find the beginning and end of every gene.

Play with natural texts. The *CalcFreq* function is not designed specifically for the four-letter alphabet text of genome sequences; it can work with any text. For natural text, it is also possible to construct local frequency dictionaries and segment it into "homogeneous" (in short word frequencies) parts. But be careful with longer frequency dictionaries, for an English text one has $(26 + 1)^2 = 729$ possible duplets (including space as an extra letter)!

Students can also look at visualizations of 143 bacterial genomic sequences at [14]. All possible types of the 7-cluster structure have been described in [15]. Nonlinear principal manifolds were utilized for visualization of the 7-cluster structure in [17]. Principal trees are applied for visualization of the same structure in [18].

14.8 Conclusion

In this exercise on applying PCA and K-means to the analysis of a genomic sequence, the students learn basic bioinformatics notions and train to apply PCA to the visualization of local frequency dictionaries in genetic text.

References

1. Genbank FTP-site: ftp://ftp.ncbi.nih.gov/genbank/genomes
2. An http-folder with all materials required for the tutorial: http://www.ihes.fr/∼zinovyev/pcadg .
3. Crick, F.H.C., Barnett, L., Brenner, S., and Watts-Tobin, R.J.: General nature of the genetic code for proteins. Nature, **192**, 1227–1232 (1961)
4. Clark, D. and Russel, L.: Molecular Biology Made Simple and Fun. Cache River Press (2000)
5. Caulobacter crescentus short introduction at http://caulo.stanford.edu/caulo/ .
6. Jackson, J.: A User's Guide to Principal Components (Wiley Series in Probability and Statistics). Wiley-Interscience, (2003)
7. Zinovyev A.: Hierarchical Cluster Structures and Symmetries in Genomic Sequences. Colloquium talk at the Centre for Mathematical Modelling, University of Leicester. December, (2004) (PowerPoint presentation at http://www.ihes.fr/∼zinovyev/presentations/7clusters.ppt.)
8. Zinovyev, A.: Visualizing the spatial structure of triplet distributions in genetic texts. HES Preprint, M/02/28 (2003) Online: http://www.ihes.fr/PREPRINTS/M02/Resu/resu-M02-28.html
9. Gorban, A.N., Zinovyev, A.Yu., and Popova, T.G.: Statistical approaches to the automated gene identification without teacher, Institut des Hautes Etudes Scientiques. IHES Preprint, M/01/34 (2001) Online: http://www.ihes.fr web-site. (See also e-print: http://arxiv.org/abs/physics/0108016)
10. Zinovyev, A.Yu., Gorban, A.N., and Popova, T.G.: Self-Organizing Approach for Automated Gene Identification. Open Systems and Information Dynamics, **10**(4), 321–333 (2003)
11. Gorban, A., Zinovyev, A., and Popova, T.: Seven clusters in genomic triplet distributions In Silico Biology, **3** 0039 (2003) (Online: http://arxiv.org/abs/cond-mat/0305681 and http://cogprints.ecs.soton.ac.uk/archive/00003077/)
12. Gorban, A.N., Popova, T.G., and Zinovyev, A.Yu.: Codon usage trajectories and 7-cluster structure of 143 complete bacterial genomic sequences. Physica A, **353**, 365-387 (2005)
13. Ou, H.Y., Guo, F.B., and Zhang, C.T.: Analysis of nucleotide distribution in the genome of Streptomyces coelicolor A3(2) using the Z curve method, *FEBS Lett.* **540** (1-3), 188–194 (2003)
14. Cluster structures in genomic word frequency distributions. Web-site with supplementary materials. http://www.ihes.fr/∼zinovyev/7clusters/index.htm
15. Gorban, A.N., Zinovyev, A.Yu., and Popova, T.G.: Four basic symmetry types in the universal 7-cluster structure of 143 complete bacterial genomic sequences. In Silico Biology **5** (2005) 0025. On-line: http://www.bioinfo.de/isb/2005/05/0025/
16. Staden, R. and McLachlan, A.D.: Codon preference and its use in identifying protein coding regions in long DNA sequences. Nucleic Acids Res **10** (1), 141-56 (1982)
17. Gorban, A.N., Zinovyev, A.Y., and Wunsch, D.C.: Application of The Method of Elastic Maps In Analysis of Genetic Texts, In Proceedings of International Joint Conference on Neural Networks (IJCNN'03), Portland, Oregon (2003)
18. Gorban, A.N., Sumner, N.R., and Zinovyev,A.Y.: Elastic maps and nets for approximating principal manifolds and their application to microarray data visualization, In this book.

Appendix. Program listings

Function LoadSeq Listing

```
function str=LoadSeq(fafile)
fid = fopen(fafile); i=1; str = '';
disp('Reading fasta-file...');
while 1
   if round(i/200)==i/200
   disp(strcat(int2str(i),' lines'));
   end
   tline = fgetl(fid);
   if  ischar(tline), break, end;
   if(size(tline) =0)
   if(strcmp(tline(1),'>')==0)
   str = strcat(str,tline);
   end;
   end;
   i=i+1;
end
nn = size(str); n = nn(2);
disp(strcat('Length of the string: ',int2str(n)));
```

Function PCAFreq Listing

```
function PCAFreq(xx)
% standard normalization
nn = size(xx); n = nn(1) mn = mean(xx);
mas = xx - repmat(mn,n,1); stdr = std(mas);
mas = mas./repmat(stdr,n,1);
% creating PCA plot
[pc,dat] = princomp(mas);
plot(dat(:,1),dat(:,2),'k.'); hold on;
set(gca,'FontSize',16);
axis equal;
title('PCA plot for text fragments','FontSize',22);
set(gcf,'Position',[232 256 461 422]);
hold off;
```

Function CalcFreq Listing

```
function xx=CalcFreq(str,len,wid)
disp('Cutting in fragments...');
i=1; k=1;nn = size(str);
while i+wid<nn(2)
   if round(k/200)==k/200
   disp(strcat(int2str(k),' fragments'));
   end
   frag = str(i:i+wid-1); vf(k) = calcf(frag,len);
   i = i+wid; k=k+1;
end
disp('Merging into table...');
names = java.util.Vector; n = 0;
for i=1:size(vf)
   if size(vf(i)) =0
        keys = vf(i).keys;
        while keys.hasMoreElements
            key = keys.nextElement;
            if names.indexOf(key)==-1
            names.add(key);
            end
        end, n=n+1; end, end
xx = zeros(n,size(names));
for i=1:size(vf)
   if size(vf(i)) =0
   if round(i/200)==i/200
   disp(strcat(int2str(i),' points'));
   end
   for j=1:size(names)
   xx(i,j) = getwf(names.elementAt(j-1),vf(i));
   end, end, end

function vf=calcf(str,num)
vf = java.util.Hashtable; i = 1; nn = size(str);
while i+num<nn(2)
wrd = str(i:i+num-1); i = i+num; addwf(wrd,vf,1);
end

function addwf(word,hash,fr)
wf = hash.get(word);
if size(wf)==0 hash.put(word,fr); else
hash.put(word,fr+wf); end

function fr=getwf(word,hash)
wf = hash.get(word);
if size(wf)==0 r=0; else fr=wf; end
```

Function ClustFreq Listing

```
function fragn = ClustFreq(xx,k)
% centralization and normalization
nn = size(xx); n = nn(1)
mn = mean(xx);
mas = xx - repmat(mn,n,1);
stdr = std(mas);
mas = mas./repmat(stdr,n,1);
% calculating principal components
[pc,dat] = princomp(mas);
% k-means clustering
[fragn,C] = kmeans(mas,k);
% projecting cluster centers into the PCA basis
XTP = C; temp = size(XTP); nums = temp(1);
X1c = XTP-repmat(mn,nums,1);
X1r = X1c./repmat(stdr,nums,1);
X1P = pc'*X1r'; X1P = X1P';
% marking the central cluster black
cnames = ['k','r','g','b','m','c','y'];
for i=1:k no(i) = norm(X1P(i,1:3)); end
[m,mi] = min(no);
for i=1:size(fragn)
   if fragn(i)==mi fragn(i)=1;
   elseif fragn(i)==1 fragn(i)=mi; end
end
% plotting the result using PCA
for i=1:n
   plot(dat(i,1),dat(i,2),'ko',
   'MarkerEdgeColor',[0 0 0],'MarkerFaceColor',
   cnames(fragn(i)));
   hold on;
end
set(gca,'FontSize',16); axis equal;
title('K-means clustering','FontSize',22);
set(gcf,'Position',[232 256 461 422]);
```

Function GenBrowser Listing

```
function GenBrowser(str,wid,fragn,startp)
% we will show 100 fragments in the detailed view
endp = startp+wid*100; nn = size(fragn); n = nn(1);
xr1 = startp/(n*wid); xr2 = endp/(n*wid);
cnames = ['k','r','g','b','m','c','y'];
subplot('Position',[0 0 1 0.1]);
for i=1:size(fragn)
  plot(i/n,0,strcat(cnames(fragn(i)),'s'),'MarkerSize',2);
  hold on;
end
plot([xr1 xr1],[-1 1],'k'); hold on;
plot([xr2 xr2],[-1 1],'k'); axis off;
subplot('Position',[0 0.1 1 0.1]);
for i=floor(startp/wid)+1:floor(endp/wid)+1
  plot([(i-0.5)*wid (i+0.5)*wid],[0 0],
  strcat(cnames(fragn(i)),'-'),'LineWidth',5);
  hold on;
end
axis off;
subplot('Position',[0 0.25 0.98 0.75]);
xlim([0,1]); ylim([0,1]); twid = 100; nlin = 24;k=startp;
for j=1:nlin
for i=1:twid
  col = cnames(fragn(floor(k/wid)+1));
  h=text(i/twid,1-j/nlin,str(k),'FontSize',8,
  'FontName','FixedWidth');
  set(h,'Color',col);
  k=k+1;
end end
axis off;
set(gcf,'Position',[64 356 879 195]);
```